Fullstack Java Development

Trainer : Shekher

Working professional

Experience : 17+ years

Timing : 9 to 10:30 AM IST

Duration: 5 to 6 months

Backup videos(1 year validity) + class material + Live classes + placement assistance

Fee : 30k ( 15k + 15k)

Fullstack Java = Front-end + Back-end + Database + Tools



Front-end : 1) It is nothing but, designing and developing webpages of

a web application.

2) Users interacts with the webpages and webpages will show

the data/information to the users.

3) To develop the webpages, front-end technoloiges like

HTML(Hyper Text Markup Language), CSS(Cascading Style Sheet),

Javascript, Bootstrap, Angular, React, etc.. are used.

4) Front-end is also called UI (User Interface).

Back-end : 1) It is nothing but, developing a program to handle the user

requests.

2) Back-end programs interacts with the database, loads the

data from the database, performs any calculations, provides

the data to the web page.

3) Back-end programs contains business logic of an application.

4) To develp the business logic, back-end technologies like Java,

Python, .Net, etc.. are used.

Database : 1) A Database is used to store the data permanently.

2) You can perform CRUD operations on the Database.

C - Creating/inserting the data

R - Reading the data

U - Updating the data

D - Deleting the data

3) Databases are Oracle, MySQL, PostgreSQL, MongoDB, etc..

Tools : Tools are used to manage the project.

ex: Maven, JUnit, Jenkins, GitHub, Sonar, etc..

=================================================================================

Programming language:

\* language is used for communication.

\* human languages are for communication between people.

\* programming languages are for developing software applications.

\* System/computer can understand only binary language(0's and 1's).

\* When you write a program in programming languages like Java/C/C++ etc..

then the translator converts that program into binary language.

\* You have to learn a programming language in the below pattern.

1. first learn the tokens

2. learn the syntax

3. write the statements

4. write the programs.

\* tokens are 5 types.

1. keywords

2. identifiers

3. operators

4. literals

5. strings

\* statements are also 5 types.

1. input statement

2. output statement

3. memory statement

4. arithmetic and logic statement

5. control statement

Introduction JAVA:

* Java is a high-level and open source programming language.
* Java is an Object Oriented Programming language(OOP).



* It was developed by Sun Microsystems and released in 1995.



* The initial name for the programming language was Oak, later it was renamed to Java in 1995.
* Java was created by a team of 5 persons, headed by James Gosling.
* The primary goal of Java was to develop a platform-independent language, that could run on any operating system or any device.
* Java was released to the public in 1996, with a slogan

“Write Once, Run Anywhere” (WORA).

* Java is widely used for developing Web applications, enterprise applications and mobile applications development(Android).
* Java is also used in IoT(Internet of Things) applications, game development and data analysis.
* Java is extensively used in the financial industry for building banking systems and trading applications, etc.



* In 2010, Oracle Corporation has acquired Sun Microsystems company. So, from 2010, Java software maintainence and releases are handled by Oracle corporation.



Java Editions:

-------------

* Java is divided into three main editions.

1. Java SE (Java Standard Edition)
2. Java EE (Java Enterprise Edition)
3. Java ME (Java Micro Edition)

* Java SE was formerly called J2SE.
* Java EE was formerly called J2EE.
* Java ME was formerly called J2ME.
* Recently, Java EE has been renamed to Jakarta EE.

Java SE:

-------

* Java SE is the core of the Java Programming language.
* Java SE, we call it as Core JAVA.
* Java SE provides the fundmental libraries for creating general-purpose applications.
* Using Java SE, we can develop Desktop applications, also called stand-alone applications.
* Desktop applications are called single user applications, and a user has to download and install the application in their computer.
* For example, Media players, calculators, Anti-virus, IDE’s, etc.

Java EE:

-------

* Java EE was built on Java SE and it provides additional libraries for developing large-scale applications.
* Java EE provides Servlets and JSP(Java Server Page), JPA(Java Persistence API), RESTful webservices, Messaging, etc..
* With Java EE, we can develop web applications and enterprise applications.
* Both web and enterprise applications runs on a server. So, you no need to install them on your computer.
* For example, social media applications like facebook or twitter are enterprise applications, and you no need to install in your computer. You can access them from your browser.
* web applications are small applications like restaurant applications or online tutorials or hotel booking applications.
* enterprise applications are large applications like banking applications, e-commerce applications, CRM applications,etc..

Java ME:

-------

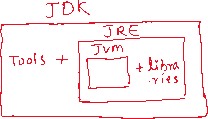
* Java ME is provided for developing applications for Mobile phones, embedded systems, IoT devices
* Using Java ME, you can develop applications for smart appliances, sensors, disply modules, car infotainment systems, etc.
* For a Full Stack Java Developer, Java SE and Java EE are required, but not Java ME.

JVM, JRE and JDK:

* JVM stands for Java Virtual Machine.
* JVM executes a Java program, by converting the Java’s byte code into machine code.



* JRE stands for Java Runtime Environment.
* JRE is a package which contains JVM and other Java libraries to execute the Java programs.
* With JRE, we can only execute the Java programs, but we can’t develop the programs and we can’t compile the programs.
* JDK stands for Java Development Kit.
* JDK provides tools for compiling the programs, and also provides JRE for executing the Programs.
* So, with JDK, we can develop the programs and we can execute the programs.
* JDK provides JRE, and JRE provides JVM.



Q) 2I want to execute the Java program, but I don’t want to develop the program in my computer. What is requied in my computer JDk or JRE?

A) JRE

Q) I want to develop the Java program, and also I wan’t to execute the program in my computer. what is required in my computer JDK or JRE?

A) JDK

Analogy:

. JVM is like a DVD player, which plays a movie by converting the code available on the DVD.

. JRE is like a home theater, which is package with DVD player and speakers and other components.

. JDK is like a film studio, which is a package with home theater, and other tools like cameras, editing tools etc.

Installing Java SE 17:

1. visit <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>
2. download Windows x64 MSI Installer
3. A file jdk-17.0.12\_windows-x64\_bin.msi is downloaded.
4. Double click on the downloaded file, and click next buttons, then finally close button.
5. Now Java 17 software is successfully installed in your computer and you can find the software at,

C:\Program Files\Java\jdk-17

Path setting:

1. Goto windows search and type environment
2. select Edit system environment variables
3. click on Environment variables button.
4. Under system variables, select path variable and click edit.
5. click on New button and enter the jdk’s bin folder location into the textbox. (copy paste the location, don’t type manually).

(C:\Program Files\Java\jdk-17\bin)

1. Move this path upto the top.
2. click on OK, to close the windows.

Note: To verify the path setting is done correctly or not.

1. open command prompt
2. excute the below command.

* java -version

structure of a Java program:

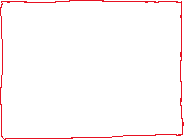
----------------------------

package statement;

import statement;

class <classname>

{



variables;

methods;

}

* package statement is used to specify the name of the package, in which this class has to be stored.
* we can write 0 or 1 package statement in a java program.
* import statement is used to import the one java program into another java program.
* we can write 0 or multiple import statements in a java program.
* class statement is used to create a class in a program.
* minimum one class is required to write a java program.
* You can write multiple classes also if required.
* So, without a class we can’t develop the Java programs.
* Every Java program should contain a file extention “.java”.

program development steps:

1. write the program
2. compile the program
3. execute the program

* To write the Java programs, we can use

1. text editor, or



1. IDE software



* text editors are notepad, notepad++, Edit Plus, etc..



* IDE(Integrated Development Environment) software’s are Eclipse, IntelliJ IDEA, STS(Spring Tool Suite), etc...



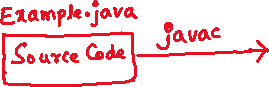
* To compile the Java program, we have to use a tool called ‘javac’(java compiler).



* When you compile the Java program, your source code is converted to byte code, and this byte code will be stored into a file with extention “.class”.
* To execute the Java program, we have to use a tool called “java”.



* This “java” tool internally starts the JVM, to execute the program.



First program:

1. open notepad
2. write the program

class Hello

{

public static void main(String[] args)



{

System.out.println("Hello world");

}



}

1. save the program as Hello.java

Note: In my system, I have saved the program in

D drive.

1. open the command prompt

C:\Users\Windows>D: (then press enter)

D:\>javac Hello.java ( compiling the program)

D:\>java Hello ( executing the program)

output: Hello world

second program:

1. open notepad
2. write the program

class Demo

{

public staic void main(String[] args)

{

System.out.print("Hello");

System.out.print("World");

}

}

1. save the program as Demo.java
2. open the command prompt

C:\Users\Windows>D:

D:\>javac Demo.java

D:\>java Demo

output: HelloWorld

Q) what is the difference between print() and println() methods?

A) print() method, after printing the output, it will place the

cursor on the same line.

println() method, after printing the output, it will move the

cursor to the new line.

Q) what is the output of the below statements?

System.out.print(“Hello”);

System.out.println(“World”);

A) a) Hello

World

b) HelloWorld

c) Hello World

d) None

answer: b

Q) what is the output of the below statements?

System.out.println(“Hello”);

System.out.print(“ World”);

System.out.println(“Java”);

A) a) Hello WorldJava

b) Hello

World

Java

c) Hello

WorldJava

d) Hello

WorldJava

answer: c

Q) If I want to execute a Java program for 2 times, how many times I need to compile the program?

A) one time

Q)If I do any changes in the program, do I need to compile the program again?

A) Yes.

keyword:

-------

* A keyword is also called as a reserved word, which has a pre-defined meaning in the programming language.
* Every programming language provides keywords, but the number of keywords may be different.
* In Java, there are 53 core keywords and in Java 17 version, 14 new keywords are included. So, Java 17 has 67 keywords.
* The keywords should not be used as variable names, method names, class names, etc..

ex:

abstract, final, class, import, extends, catch, break, continue, for , if , while, switch, etc..

identifiers:

-----------

* An identifier is a name given to the variable or a name given to a method or a name given to a class or an interface, etc..
* The rules to define an identifier are,

1. Don’t use a keyword as identifier.
2. identifier can have letters(uppercase & lowercase), digits(0-9), underscore( \_ ) or a dollar($). Other special characters are not allowed.
3. Don’t start an identifier with a digit.
4. an identifier must not contain a whitespace in the middle.
5. identifier is a case-sensitive.

ex1:



class abstract



{



}

ex2:

int myVar; //correct

int my Var; // error

int my\_Var; //correct

int my-Var; //error

int my@Var; //error

int my$Var; //correct

int \_my$Var; //correct

int 4Seater; //error

int row4; // correct

int \_\_id = 101; // correct

int $pid = 1765; // correct

ex3:

class Customer Service //error

{

}

class Customer\_Service //correct

{

}

ex4:

int MAX USERS = 100; //error

int MAX\_USERS = 100; //correct

memory statement and datatypes:

--------------------------------

* memory statement tells the JVM about how much memory should be allocated for a variable.

datatype variable; //variable declaration

variable = value; //variable initialization

(or)

datatype variable = value; //variable declaration and initialization

For ex:



int x = 10;

. The above memory statement



tells the JVM that allocate

4 bytes memory, give the name as x

and store the value as 10.

double y = 19.99;



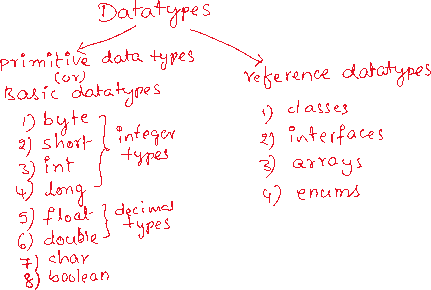
.The above memory statement tells

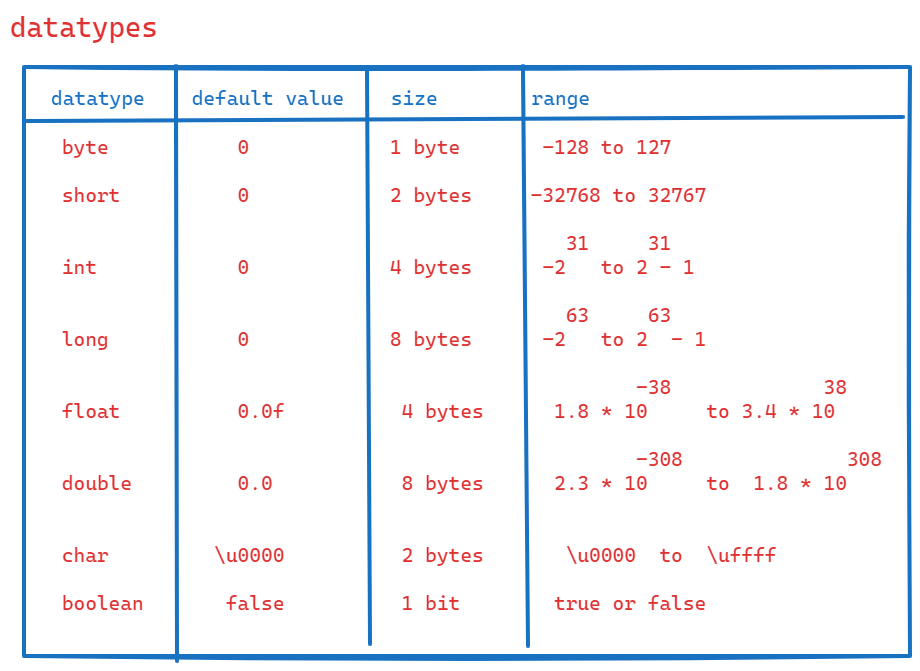
the JVM that allocate 8 bytes memory,



give the name as y and store the value

as 19.99





ex1:

byte a = 120; //correct

byte b = 120; //correct

byte c = a + b; //error

ex2:

short s1 = 25777; //correct

short s2 = 15100; //correct

short s3 = s1 + s2; //error

ex3:

boolean status = true; //correct

boolean flag = null; //error

ex4:

float f1 = 71.56; //error

float f2 = 71.56f; //correct

float f3 = 71.56F; //correct

Note: when you store a float value, the value must be

suffixed with either f or F.

Installing Eclipse IDE:

----------------------

1. visit <https://www.eclipse.org/downloads/>
2. click on download X86\_64
3. a file eclipse-inst-jre-Win64.exe is downloaded.
4. double click on the downloded file
5. choose Eclipse IDE for Enterprise Java and Web developers
6. click on install.

Note: After installation successful, you will get eclipse shortcut on desktop.

Q) Do we have any error in the below statement?

int x = 10, int y = 20;

1. Yes, we have an error.

In a single statement, we can use the datatype for only once.

Q) Do we have any erro in the below statement?

int x = 10, y = 20;

1. No, we don’t have error. Here we have created two variables with the datatype as int.

Q) Do we have any error in the below statement?

int x = 10, double y = 2.45;

1. Yes, we have an error.

Q) Do we have any error in the below statement?

int x = 10; double y = 2.45;

1. No, we don’t have any error.

Here, it is two lines of Java code, because semicolon(;) is a line termination in Java.

reading the input from the user:

* To read the input values from a user, a Java program has to use Scanner class.
* Scanner class is a pre-defined class given by Java only in java.util package.
* To use the Scanner class, first you have to create the object like below.

Scanner scanner = new Scanner(System.in);

* You have to import the Scanner class into your program like below.

import java.util.Scanner;

* In the Scanner class, multiple methods are provided to read the input values.

nextInt() 🡪 reads int value

nextByte() 🡪 reads byte value

nextShort() 🡪 reads short value

nextLong() 🡪 read long value

nextFloat() 🡪 reads float value

nextDouble() 🡪 reads double value

nextBoolean() 🡪 reads boolean value

nextLine() 🡪 reads string value

Note: we don’t have nextChar() method.

Q) what is static program and what is dynamic program?

A) a static program provides the same result, every time when it is executed.

a dynamic program provides the result, based on the input values given by the user.

steps to develop a Java program in eclipse:

* First we need to create a folder in our system, and it will be used as a workspace in eclipse.
* A workspace is a directory where the Java projects are stored by the eclipse.
* create a folder FSJD-9AM at C:\Users\Windows\Documents.

1. launch eclipse from the desktop.
2. select the workspace created, by clicking on browse button then launch.
3. click on File menu -> New -> Project... -> Java Project -> Projectname: Example01 -> choose execution environment JRE :Java SE 17 -> Next -> Finish.
4. Expand the project Example01 -> src -> module-info.java. Delete this file.
5. Right click on src folder -> New -> class -> enter Name:Sum -> choose public static void main(String[] args) checkbox -> finish
6. write the program code.

**import** java.util.Scanner;

**public** **class** Sum {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Hey! Enter first number");

**int** a = scan.nextInt();

System.***out***.println("Hey! Enter second number");

**int** b = scan.nextInt();

**int** c = a + b;

System.***out***.println("Addition : " + c);

scan.close();

}

}

1. Right click on Sum.java -> RunAs -> Java Application

How to write comments?

---------------------

* comment is a meta data
* meta data is the data which tells something about other data.
* The comments will help a developer to understand the code written by the other developer.
* We can write 3 types of comments in a Java program.

1. single line comment
2. multiple lines comment
3. documentation comment

* a single line comment, you can start with //
* a multiline comment, you can start with /\* and end with \*/
* a documentation comment, you can start with /\*\* and end with \*/

/\*

\* write a program to convert the given forenheit

\* value into celsius value.

\* formula:

\* c = (f - 32) \* 5 / 9

\*/

**import** java.util.Scanner;

**public** **class** ForenheitToCelsius {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Hey! Enter forenheit value");

**double** f = scan.nextDouble();

**double** c = (f - 32) \* 5 / 9;

System.***out***.println("Foreheit value : " + f);

System.***out***.println("Celsius value : " + c);

}

}

/\*

\* write a program to convert celsius value

\* to the forenheit value

\* formula:

\* f = (c \* 9) / 5 + 32

\*/

**import** java.util.Scanner;

**public** **class** CelsiusToForenheit {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Hey! Enter celsius value");

**double** c = scan.nextDouble();

**double** f = (c \* 9) / 5 + 32;

System.***out***.println("Celsius value : " + c);

System.***out***.println("Forenheit value : " + f);

scan.close();

}

}

/\*

\* write a program to calculate the power of

\* a number.

\* ex:

\* a = 2

\* b = 3 3

\* output: 2 = 8

\*

\* Note: In Java, Math class is provided with methods to

\* perform mathematical operations.

\* Math class is given in java.lang package

\* java.lang package is by default imported into java program.

\* So no need to write import statement.

\* To calculate the power, the method is pow().

\*/

**import** java.util.Scanner;

**public** **class** Power {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter first number");

**int** a = scan.nextInt();

System.***out***.println("Enter second number");

**int** b = scan.nextInt();

**double** c = Math.*pow*(a, b);

System.***out***.println("power = " + c);

scan.close();

}

}

/\*

\* write a program to calculate the emi for the given input

\* formula:

\* n

\* emi = p \* r \* (1 + r)

\* --------

\* n

\* (1 + r) - 1

\* p -- is the principle amount

\* r -- is the rate of interest per month

\* n -- is the number of months

\*/

**import** java.util.Scanner;

**public** **class** EmiCalculator {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the principle amount");

**double** p = scan.nextDouble();

System.***out***.println("Enter the rate of interest per annaum");

**double** pa = scan.nextDouble();

//convert rate of interest from per annaum to per month

**double** r = pa / 12 / 100;

System.***out***.println("Enter the number of months");

**int** n = scan.nextInt();

**double** x = Math.*pow*(1+r , n);

**double** emi = p \* r \* x / (x - 1) ;

System.***out***.println("Emi to Pay : " + emi);

scan.close();

}

}

Q)what is the difference between float and double?

A) Both can store decimal point values.

float can store upto 6 digits after decimal point.

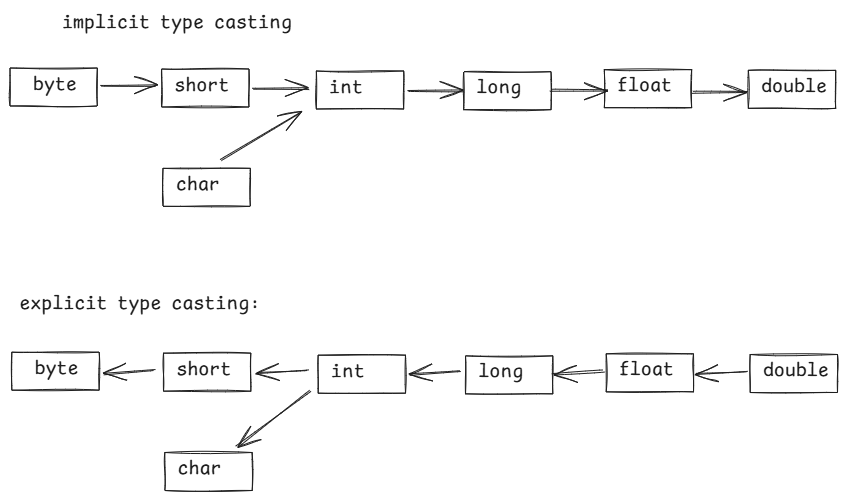
double can store upto 15 digits after decimal point.

while assigning the float value, the value must be suffixed with f or F.

Type casting:

------------

* Type casting is nothing but converting a variable from one datatype to another datatype.
* Type casting is of 2 types.
* 1. implicit type casting/up-casting
* 2. explicit type casting/down-casting
* implicit type casting means, converting a variable from low memory type to high memory type.
* explicit type casting means, converting a variable from high memory type to low memory type.



Note: boolean variable can’t be type casted to any other data type or any other variable can’t be type casted to boolean data type.

* In explicit type casting, we should specify the target data type within the paranthesis.

For example:

long a = 175L;

int b = a; //error

int b =(int)a; //correct

/\*

\* write a program for implicit type casting

\*/

**public** **class** ImplicitCasting {

**public** **static** **void** main(String[] args) {

//type casting from byte to int

**byte** x1 = 125;

**int** x2 = x1;

System.***out***.println("byte value : " + x1);

System.***out***.println("int value : " + x2);

//type casting from char to int

**char** x3 = 'A';

**int** x4 = x3;

System.***out***.println("char value : " + x3);

System.***out***.println("int value : " + x4);

//type casting from long to float

**long** x5 = 49L;

**float** x6 = x5;

System.***out***.println("long value : " + x5);

System.***out***.println("float value : " + x6);

}

}

/\*

\* write a program for explicit type casting

\*/

**public** **class** ExplicitCasting {

**public** **static** **void** main(String[] args) {

//converting from long to int

**long** x1 = 175L;

**int** x2 = (**int**)x1;

System.***out***.println("long value : " + x1);

System.***out***.println("int value : " + x2);

//converting from int to char

**int** x3 = 107;

**char** x4 = (**char**)x3;

System.***out***.println("int value : " + x3);

System.***out***.println("char value : " + x4);

//converting from float to short

**float** x5 = 15.29f;

**short** x6 = (**short**)x5;

System.***out***.println("float value : " + x5);

System.***out***.println("short value : " +x6);

}

}

Q) is the following code is correct?

boolean flag = 1;

A) No, it is an error.

Q) is the following code is correct?

boolean flag = true;

A) Yes.

Q) is the following code is correct?

boolean flag = true;

int x = flag;

A) No, it is an error. Because boolean can’t be type casted.

==================================



operators:

* operator is a symbol which performs a well-defined task.
* operators are 3 types.

1. unary operators
2. binary operators
3. ternary operator

* unary operators works with a single operand.
* binary operators works with two operands.
* ternary operator works with 3 operands/parts.

unary operators:

----------------

1. ++ ----> increment operator
2. -- ----> decrement operator

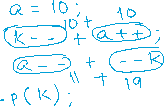
variable++ -->post increment(increment after the use)

++variable -->pre increment(increment before the use)



variable-- -->post decrement (decrement after the use)

--variable -->pre decrement(decrement before the use)



Binary operators:

-----------------

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Bitwise operators

Arithmetic operators:

--------------------

+ --- addition

* --- subtraction

\* --- multiplication

/ --- division

% --- modulus

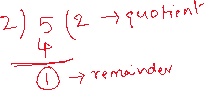
* division operator returns the quotient value and modulus operator returns the remainder value.



ex1:

int x = 5 / 2;

int y = 5 % 2;



S.o.println(x);

S.o.println(y);

output: 2

1

ex2:



int x = 42 / 7;



int y = 42 % 7;

S.o.println(x);



S.o.println(y);

output: 6

0

ex3:



double x = 5.0 / 2;



double y = 5.0 % 2;



S.o.p(x);



S.o.p(y);

output: 2.5

1.0

Note: division operator(/) performs floating point division, if either

numerator or denominator is a decimal value. If both are integers

then division operator performs integer division.

modulus operator(%) performs integer division only, for integers or

for decimal point values.

ex4:

char ch1 = ‘A’;

char ch2 = ‘B’;

int x = ch1 / ch2;

int y = ch1 % ch2;

S.o.p(x);

S.o.p(y);



Relational operators:

---------------------

< --- less than

> --- greater than

<= --- less than or equals

>= --- greater than or equals

== --- equals

!= --- not equals

ex1:

int a = 10, b = 30;

System.out.println(a != b); //true

System.out.println(a == b); //false

System.out.println(a < b); // true

System.out.println(a <= b); // true

System.out.println(a > b); //false

System.out.println(a >= b); //false

ex2:

int x = 10;

double y = 20.0;

S.o.println(x == y); //false

S.o.println(x != y); //true

ex3:

int i = 65;

char ch = ‘A’;

S.o.println( i == ch); //true

S.o.println( i != ch); //false

ex4:

boolean flag = true;

int k = 1;

S.o.println(flag == k); //error

ex5:

boolean f1 =true;

boolean f2 =true;

S.o.println(f1 == f2); //true

S.o.println(f1 != f2); //false

S.o.println(!f1 != f2); //true

S.o.println(!f1 == !f2); //true

ex6:

char ch1 = ‘A’;

char ch2 = ‘a’;

S.o.println(ch1 == ch2); //false

S.o.println(ch1 != ch2); //true

S.o.println(ch1 < ch2); //true

Note: we can compare integer types, floating-point types and char type as any with any. But we can’ compare boolean type with any other type.

Logical operators:

-----------------

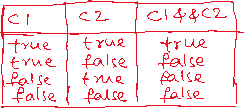
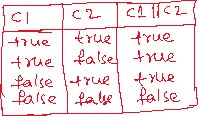
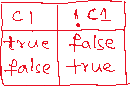
* relational operators are used to define a condition.
* logical operators are used to combine multiple conditions together as a single condition.

&& ---- Logical AND

|| ---- Logical OR

! ---- Logical NOT

* Logical AND (&&) operator returns true, if both the conditions are true. Otherwise, returns false.
* Logical OR(||) operator returns false, if both the conditions are false. Otherwise, returns true.
* Logical NOT(!) operator will inverse the value of a condition. If the condition is true, it will convert the value to false and if the condition is false, it will convert the value to true.



ex1:

int x = 10, y = 12;

S.o.println(x == y || x < y); //true

S.o.println(x > y && y > 10); //false

S.o.println(!(x < y)); // false

ex2:

int x = 10;

if(x++ >= 11 && ++x < 15) {

++x;

}

S.o.println(x);

output: 11

ex3:

int x = 5;

if( ++x > 5 || x-- < 5 ) {

x++;

}

S.o.println(x);

output: 7



ex4:

**int** x = 5;

**if** (++x > 8 || x-- < 5) {

x++;

}

System.***out***.println(x);

output: 5

ex5:

int a = 8, b = 5;

if( --a <= b++ || --a > 8 ) {

++a;

--b;

}

S.o.println(a);

S.o.println(b);

output:

6

6

Note: Logical AND(&&) operator executes the second condition, only if the first condition is true.

Logical OR(||) operator executes the second condition, only if the first condition is false.

Assignment operators:

--------------------

= --- assign

+= --- add and assign

-= --- subtract and assign

\*= --- multiply and assign

/= --- divide and assign

%= --- modulus and assign

ex1:

int a;

a = 10 //assigning the value

a == 10 //checking the value, it is a condition

ex2:

int a = 50;

a += 10; (a = a + 10)

S.o.println(a);

output: 60

ex3:

int x = 5;

x %= 2; (x = x % 2)

S.o.println(x);

output: 1

ex4:

double a = 5.0;

a /= 0; (a = a / 0)

S.o.println(a);

output: Infinity

ex5:

int a = 5;

a /= 0;

S.o.println(a);

output: ArithmeticException

Bitwise operators:

-----------------

* Bitwise operators works on the bits of the operands.
* Bitwise operators are mostly used in the cryptography applications development, embedded systems development, device drivers development,etc.

**& ---- Bitwise AND**

**| ---- Bitwise OR**

**^ ---- Bitwise XOR**

**<< ---- left shift**

**>> ----- right shift**

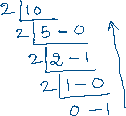
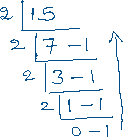
**Bitwise AND(&):**

**\* It returns a bit 1, if the corresponding bits of the operands is 1. Otherwise, it returns 0.**

**ex:**

**int a = 10;**

**int b = 15;**



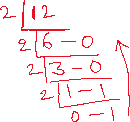
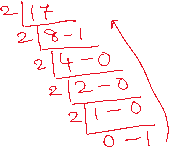
**S.o.println( a & b);**



ex2:

int a = 12;

int b = 17;



S.o.println( a & b);

output:



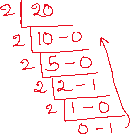
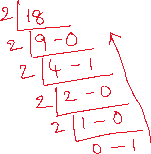
Bitwise OR(|):

\* It returns a bit 1, if either both corresponding bits are 1 or atleast one bit is 1. Otherwise it returns a bit 0.

ex:

int a = 20;

int b = 18;



S.o.println(a|b);

output: 22



ex:



int x = 5;

int y = 10;

S.o.println( x | y);

output: 15

Bitwise XOR(^):

* It returns a bit 1, if the corresponding bits are opposite. Otherwise, it returns a bit 0.

ex:



int a = 10;



int b = 20;



S.o.println( a ^ b);



output: 30

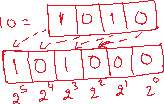
Bitwise left shift(<<):

\* It will shift the bits of an operand by the given positions to the left and fills the right most bits with 0.

ex:

int a = 10;

S.o.println( a << 2);



output: 40

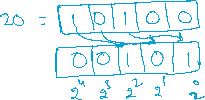
Bitwise right shift:

* It will shift the bits of an operand by the given positions to the right and fills the left most bits with 0.

ex:

int a = 20;

S.o.println(a >> 2);



output: 5

=========================================================================

Ternary operator:

* Ternary operator works with 3 parts and the symbol is ?:

syntax:

result = condition ? value1 : value2;

* if the condition is true, then value1 will be returned. Otherwise value2 will be returned.

ex:

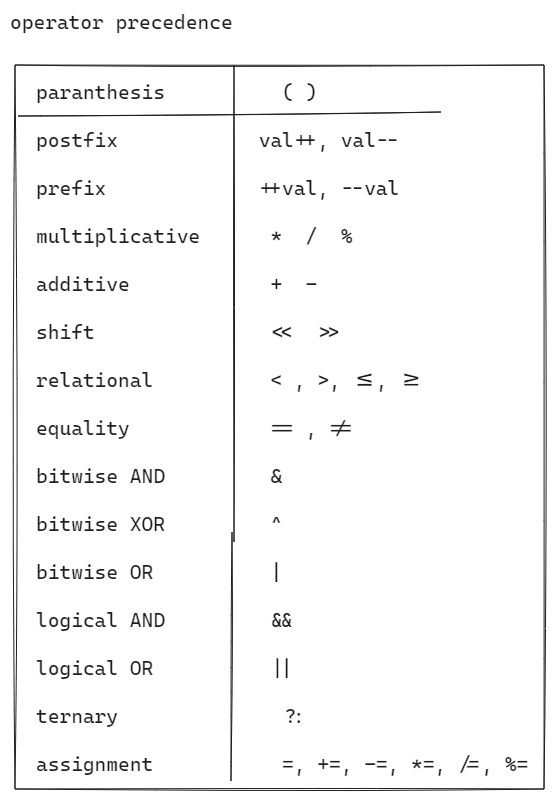
int a = 15;

int b = 17;

int c = a > b ? a : b;

S.o.println(c);

output: 17



ex1:

int x = 9 \* 3 – 12/ (2 + 2) – 4;

S.o.println(x);

output: 20

ex2:

int x = **6 – 7 \* 8 + 9 / 5 – 6 % 3 – 7 / 2 - 3;**

**System.out.println(“ x = “ + x);**

**output: x = -55**

**ex3:**

int x = **3 \* 5 – 7 + 7 \* 7 – 7 / 7 + 7 \* 7 % 7;**

**S.o.p(“ x = “ + x);**

**output: x = 56**

**ex4:**

**int x = 10 ^ 15 & 10 | 5;**

**S.o.println(“x = “ + x);**

**output: x = 5**



control statements

* control statements are 3 types.

1. selection control statements (or)

conditional control statements

1. iterational control statements (or)

loop statements

1. jump control statements

selection control statements:

-----------------------------

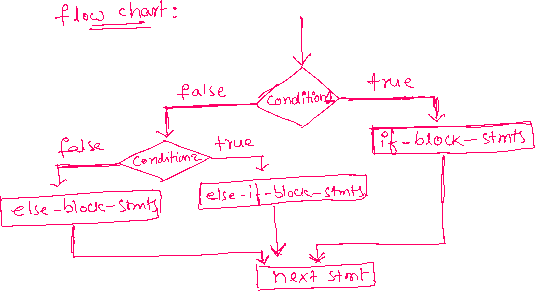
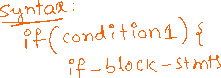
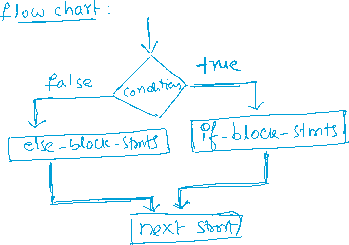
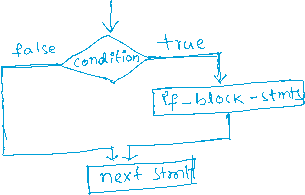
* selection control statements are 2 types.

1. if statement
2. switch statement

if statement:

------------

1. simple if
2. if else
3. if else ladder
4. nested if



syntax:

if(condition1) {

if(condition2) {

nested-if-block-stmts;

}

else {

nested-else-block-stmts;

}

}

else {

else-block-stmts;

}

next statement;

/\*

\* write a program to check whether a given number is even or odd.

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**if**( n % 2 == 0) {

System.***out***.println(n + " : is even number");

}

**else** {

System.***out***.println(n + " : is odd number");

}

scan.close();

}

}

/\*

\* write a program to take the two numbers as input and

\* do the following.

\* --> if the two numbers are same then display the sum of them.

\* --> if they are different then display, double of their product.

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter first number");

**int** a = scan.nextInt();

System.***out***.println("Enter second number");

**int** b = scan.nextInt();

**if**(a == b)

System.***out***.println("sum = " + (a + b));

**else**

System.***out***.println("double of product = " + 2 \* (a \* b));

scan.close();

}

}

/\*

\* write a program to take a number as input and implement the

\* following.

\* --> if the number is divisible by 3 then display "Zip"

\* --> if the number is divisible by 5 then display "Zap"

\* --> if the number is divisible by 3 and 5 then display "Jar"

\* --> Otherwise, display "Rar"

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**if**(n % 3 == 0 && n % 5 == 0)

System.***out***.println("Jar");

**else** **if**(n % 3 == 0)

System.***out***.println("Zip");

**else** **if**(n % 5 == 0)

System.***out***.println("Zap");

**else**

System.***out***.println("Rar");

scan.close();

}

}

/\*

\* write a program to take the distance value as input and apply

\* the following conditions to calculate the delivery fee.

\* ---> For first 3 kms free delivery

\* ---> For next 3 kms, Rs 12 per km

\* ---> For the remaining kms, Rs 15 per km

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the distance");

**double** distance = scan.nextDouble();

**if**(distance <= 3) {

System.***out***.println("Free delivery");

}

**else** **if**(distance <= 6) {

**double** deliveryFee = (distance - 3) \* 12;

System.***out***.println("delivery fee = " + deliveryFee);

}

**else** {

**double** deliveryFee = 3 \* 12 + (distance - 6) \* 15;

System.***out***.println("delivery fee = " + deliveryFee);

}

scan.close();

}

}

/\*

\* write a program to find the biggest of 3 numbers using

\* nested if conditions.

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter first number");

**int** a = scan.nextInt();

System.***out***.println("enter second number");

**int** b = scan.nextInt();

System.***out***.println("enter third number");

**int** c = scan.nextInt();

**if**(a > b) {

**if**(a > c) {

System.***out***.println(a + " : is biggest number");

}

**else** {

System.***out***.println(c + " : is biggest number");

}

}

**else** {

**if**(b > c) {

System.***out***.println(b + " : is biggest number");

}

**else** {

System.***out***.println(c + " : is biggest number");

}

}

scan.close();

}

}

/\*

\* write a program to take 3 sides of a triangle as input and do the following.

\* ---> first check the 3 sides can form a triangle or not.

\* ---> if yes, then verify the 3 sides can form which type of triangle

\* (equilateral, isosceles or scalane)

\* ---> if all the 3 sides are equal then it is equilateral triangle

\* ---> if any 2 sides are equal then it is isosceles triangle

\* ---> if 3 sides are different then it is scalane triangle

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter side1 of triangle");

**double** side1 = scan.nextDouble();

System.***out***.println("Enter side2 of triangle");

**double** side2 = scan.nextDouble();

System.***out***.println("Enter side3 of triangle");

**double** side3 = scan.nextDouble();

**if**( side1 + side2 > side3 && side2 + side3 > side1 && side1 + side3 > side2)

{

**if**(side1 == side2 && side2 == side3)

System.***out***.println("Equilateral triangle");

**else** **if**(side1 == side2 || side2 == side3 || side3 == side1)

System.***out***.println("Isosceles triangle");

**else**

System.***out***.println("Scalane triangle");

}

**else**

{

System.***out***.println("The given sides can't form a triangle");

}

scan.close();

}

}

========================================================================

/\*

\* write a program to check whether a given year is a leap year or not

\* ---> leap year conditions:

\* 1. year divisible by 4 and not divisible by 100, is a leap year

\* 2. year divisible by 4, divisible by 100 then it must be divisible by

\* 400, then it is leap year. Otherwise not a leap year.

\* 4. year not divisible 4, it is not a leap year.

\*/

**import** java.util.Scanner;

**public** **class** LeapYearTest {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a year");

**int** year = scan.nextInt();

**if**(year % 4 == 0) {

**if**(year % 100 == 0) {

**if**(year % 400 == 0) {

System.***out***.println(year + " : is a leap year");

}

**else** {

System.***out***.println(year + " : is not leap year");

}

}

**else** {

System.***out***.println(year + " : is a leap year");

}

}

**else** {

System.***out***.println(year + " : is not a leap year");

}

scan.close();

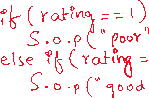
}

}

switch statement:

---------------

* Suppose, you want to define multiple conditions on the same variable and the condition is equality then you can use switch statement, instead of if statement, to make the code more readable.



syntax:

switch(variable/expression)

{

case value1: statements;

break;

case value2: statements;

break;

. . .

. . .

case valueN: statements;

break;

default: statements;

}

next statement;

* switch statement compares the variable value with first case value. If they are equal, the corresponding statements are executed and the break statement will move the control to the next statement.
* If the variable and the first case are not matched, then switch will compare with the second case value. If matched then the corresponding statements are executed, then break will move the control to the next statement.
* If the variable value is not matched with any case then default case will be executed.
* writing the default case is optional.

ex1:

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter your rating between 1 to 3");

**int** rating = scan.nextInt();

**switch**(rating) {

**case** 1: System.***out***.println("Poor");

**break**;

**case** 2: System.***out***.println("Good");

**break**;

**case** 3: System.***out***.println("Excellent");

**break**;

}

scan.close();

}

}

Q) is break statement compulsory in a case?

A) No. If break is not included then the control will execute the further cases also until break statement occurs.

ex:

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter your rating between 1 to 3");

**int** rating = scan.nextInt();

**switch**(rating) {

**case** 1: System.***out***.println("Poor");

**case** 2: System.***out***.println("Good");

**case** 3: System.***out***.println("Excellent");

**break**;

}

* suppose if input is 1, the output is,

Poor

Good

Excellent

* suppose if input is 2, the output is,

Good

Excellent

* suppose if input is 3, the output is,

Excellent

Q) can we use a float or double variable in switch statement?

A) No. We can use only any integer type(byte/short/int/long) or char type or a string type variable in switch statement.

Q) can we use a boolean variable in switch statement?

A) No.

Q) can we write a case without any statements?

A) Yes. If the case matches then the control moves to further cases also until break.

ex:

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a lower case or upper case letter");

**char** ch = scan.next().charAt(0);

**switch**(ch) {

**case** 'a':

**case** 'e':

**case** 'i':

**case** 'o':

**case** 'u':

**case** 'A':

**case** 'E':

**case** 'I':

**case** 'O':

**case** 'U': System.***out***.println("It is a vowel!");

**break**;

**default**: System.***out***.println("It is a consonent!");

}

scan.close();

* suppose if input is E, the output is,

It is a vowel!

* suppose if input is x, the output is.

It is a consonant!

example with String type:

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a day of week");

String today = scan.next().toLowerCase();

**switch**(today) {

**case** "monday": System.***out***.println("Monday's are lazy!");

**break**;

**case** "tuesday":

**case** "wednesday":

**case** "thursday": System.***out***.println("Work in progress!");

**break**;

**case** "friday": System.***out***.println("Friday's are so-so");

**break**;

**case** "saturday":

**case** "sunday": System.***out***.println("Week-end, enjoy!");

}

scan.close();

Q) can we write the default case above of all the cases?

A) Yes. We can write the default case at above all the cases or in between the cases also, but we also must add break statement for it.

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter rating between 1 to 3");

**int** rating = scan.nextInt();

**switch**(rating) {

**default**: System.***out***.println("Invalid input");

**break**;

**case** 1: System.***out***.println("Poor");

**break**;

**case** 2: System.***out***.println("Good");

**break**;

**case** 3: System.***out***.println("Excellent");

**break**;

}

scan.close();

for loop:

syntax:

for (initialization; condition; increment/decrement)

{

statements;

}

next statements;

\* when control enters into for statement, first initialization

part is executed.

\* after that the condition is evaluated and if it is true then

the statements in the for loop body are executed.

\* after that the control goes to increment/decrement. After increment

or decrement, the control goes back to the condition.

\* If the condition is true, then the statements in the for loop

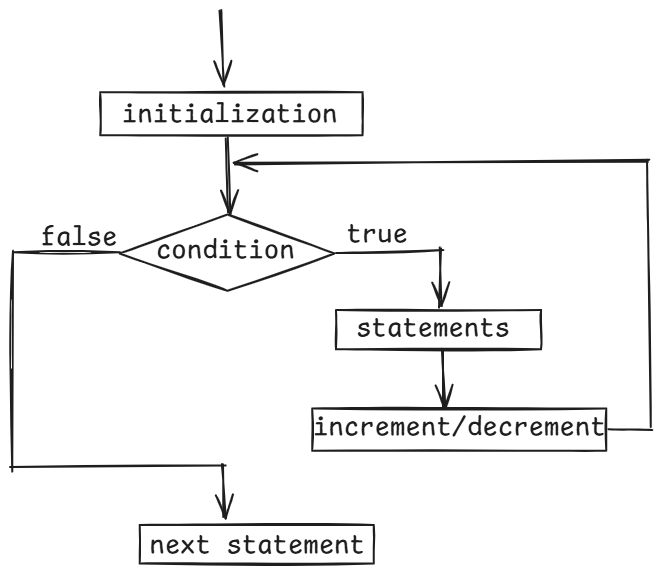
body are executed and then the control goes to increment/decrement.

\* This process will repeat, until the condition is false.

\* If the condition is false, then the control jumps to the next

statement, after the for loop.

Flow chart:



ex1:

for(int i=1; i<=3; i++) {

S.o.println(“i = “ + i);

}

output:



i = 1

i = 2

i = 3

ex2:

for(int j = 3; j >= 1; j--) {

S.o.println(“j = “ + j);

}

output:

j = 3

j = 2

j = 1

ex3:

for(int i = 1, j = 5; i <= j; i++, j--)

{

S.o.println(i + j);

}

output:

6

6

6

ex4:

int i = 1;

for(; i <= 3; i++)

{

S.o.println(“i = “ + i);

}

S.o.println(“i = “ + i);

output:

i = 1

i = 2

i = 3

i = 4

ex5:

int i=1;

for(;;)

{

S.o.println(“i=”+i);

i++;

if(i<=3)

break;

}

output:

i = 1

/\*

\* write a program to find the sum of n-natural numbers

\* for example,

\* if n = 5

\* output: 15

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

*sumOfNaturalNumbers*(n);

}

**private** **static** **void** sumOfNaturalNumbers(**int** n)

{

**int** sum = 0;

**for**(**int** i = 1; i <= n; i++)

{

sum = sum + i;

}

System.***out***.println("sum = " + sum);

}

}

/\*

\* write a program to check whether a given number is

\* a prime number or not.

\* prime number --> A number which has only 2 factors(1 and itself) is

\* called a prime number.

\* example:

\* n = 6

\* output: 6 is not a prime number

\* n = 11

\* output: 11 is a prime number

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

*checkPrime*(n);

}

**private** **static** **void** checkPrime(**int** n)

{

**int** count = 0;

**for**(**int** i=1; i<=n; i++)

{

**if**(n % i == 0) {

count++;

}

}

**if**(count == 2)

System.***out***.println(n + " : is a prime number");

**else**

System.***out***.println(n + " : is not a prime number");

}

}

//prime number program (way-2)

/\*

\* write a program to check whether a given number is

\* a prime number or not.

\* prime number --> A number which has only 2 factors(1 and itself) is

\* called a prime number.

\* example:

\* n = 6

\* output: 6 is not a prime number

\* n = 11

\* output: 11 is a prime number

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

*checkPrime*(n);

}

**private** **static** **void** checkPrime(**int** n)

{

**boolean** flag = **true**;

**for**(**int** i=2; i<=Math.*sqrt*(n); i++)

{

**if**(n % i == 0) {

flag = **false**;

**break**;

}

}

**if**( flag == **true** )

System.***out***.println(n + " : is a prime number");

**else**

System.***out***.println(n + " : is not a prime number");

}

}

/\*

\* write a program to check whether a given number is a perfect number

\* or not.

\* perfect number ---> If sum of the factors of a number excluding the given number

\* matches with the same number, then it is a perfect number.

\* example:

\* n = 6

\* The factors of 6(excluding) are, 1, 2, 3

\* sum of the factors = 1 + 2 + 3 = 6. So, 6 is a perfect number.

\* n = 28

\* The factors of 28(excluding) are, 1, 2, 4, 7, 14

\* sum of the factors = 1 + 2 + 4 + 7 + 14 = 20. So, 28 is a perfect number.

\*/

**import** java.util.Scanner;

**public** **class** PerfectNumber {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**boolean** b = *checkPerfect*(n);

**if**(b == **true**)

System.***out***.println(n + " : is a perfect number");

**else**

System.***out***.println(n + " : is not a perfect number");

}

**private** **static** **boolean** checkPerfect(**int** n) {

**int** sum = 0;

**for**(**int** i = 1; i <= n/2; i++) {

**if**(n % i ==0) {

sum += i;

}

}

**if**(sum == n)

**return** **true**;

**else**

**return** **false**;

}

}

/\*

\* write a program to find the factorial of a given number

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

*factorial*(n);

}

**private** **static** **void** factorial(**int** n) {

**long** fact = 1;

**for**(**int** i=1; i<=n; i++) {

fact = fact \* i;

}

System.***out***.println("Factorial = " + fact);

}

}

/\*

\* write a program to print multiplication table

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

*printMulitplicationTable*(n);

}

**private** **static** **void** printMulitplicationTable(**int** n) {

**for**(**int** i = 1; i <= 10; i++) {

System.***out***.println(n + " \* " + i + " = " + (n \* i));

}

}

}

/\*

\* write a program to print the n terms of fibonacci series

\* fibonacci series ----> The first two terms are 0 and 1

\* The next term is a sum of previous 2 terms.

\* example: The 5 terms of fibonacci series are,

\* 0 1 1 2 3

\* The 10 terms of fibonacci series are,

\* 0 1 1 2 3 5 8 13 21 34

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter n value for fibonacci series");

**int** n = scan.nextInt();

*printFibonacciSeries*(n);

}

**private** **static** **void** printFibonacciSeries(**int** n) {

**int** firstTerm = 0, secondTerm = 1;

**for**(**int** i = 1; i <= n; i++) {

System.***out***.print(firstTerm + " ");

**int** nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

}

}

Q) what is the output of the below code?

for(int i = 1; i <= 2; i++)

{

for(int j = 1; j <= 3; j++)

{

S.o.print(j + “ “);

}

S.out.println();

}

1. 1 2 3



1. 2 3

Q) what is the output of the below code?

for(int i = 1; i <= 3; i++)

{

for(int j = 1; j <= i; j++)

{

S.o.print(j + “ “);

}

S.out.println();

}

A) 1

1 2

1 2 3

pattern programs

* while writing the pattern programs, follow the below steps.
* 1. we always use nested for loops to print the pattern

2. we repeat the outer loop for rows/lines.

* 3. we repeat the inner loop for columns, by somehow finding
* the relationship between rows and columns.
* 4. we always print star(\*) in the inner loop.

/\*

\* write a program to print right angle triangle star pattern

\* for the given no of rows/lines.

\* example:

\* if n = 5

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows to display right angle star pattern");

**int** n = scan.nextInt();

*printPattern*(n);

}

**private** **static** **void** printPattern(**int** n) {

**for**(**int** i=1; i<=n; i++)

{

**for**(**int** j=1; j<=i; j++)

{

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to print inverted right angle

\* triangle star pattern.

\* For example,

\* n = 5,

\* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows to display inverted right angle star pattern");

**int** n = scan.nextInt();

*printPattern*(n);

}

**private** **static** **void** printPattern(**int** n) {

**for**(**int** i=1; i <= n; i++)

{

**for**(**int** j=1; j <= n - i + 1; j++)

{

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to display left angle star pattern

\* if n = 5,

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows to display left angle star pattern");

**int** n = scan.nextInt();

*printPattern*(n);

scan.close();

}

**private** **static** **void** printPattern(**int** n) {

//outer loop

**for**(**int** i = 1; i <= n; i++) {

//inner loop1 : spaces

**for**(**int** j = 1; j <= 2 \* (n - i); j++) {

System.***out***.print(" "); //single space

}

//inner loop2: stars

**for**(**int** k = 1; k <= i; k++) {

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

}

}

===================================================

/\*

\* write a program to print inverted left angle star pattern.

\* if n = 5,

\* \* \* \* \* \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*/

**import** java.util.Scanner;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows to display inverted left angle star pattern");

**int** n = scan.nextInt();

*printPattern*(n);

scan.close();

}

**private** **static** **void** printPattern(**int** n) {

//outer loop

**for**(**int** i = 1; i <= n; i++) {

//inner loop1 : spaces

**for**(**int** j = 1; j <= 2 \* (i - 1); j++) {

System.***out***.print(" "); //single space

}

//inner loop2: stars

**for**(**int** k = 1; k <= n - i + 1; k++) {

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to print pyramid star pattern

\* if n = 5,

\* \*

\* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows to display pyramid star pattern");

**int** n = scan.nextInt();

*printPattern*(n);

scan.close();

}

**private** **static** **void** printPattern(**int** n) {

//outer loop

**for**(**int** i = 1; i <= n; i++) {

//inner loop1 : spaces

**for**(**int** j = 1; j <= 2 \* (n - i); j++) {

System.***out***.print(" "); //single space

}

//inner loop2: stars

**for**(**int** k = 1; k <= 2 \* i - 1; k++) {

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to print the diamond pattern

\* if n = 5,

\* \*

\* \* \* \*

\* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \*

\* \* \* \*

\* \*

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the number of rows to display diamond star pattern");

**int** n = scan.nextInt();

*printPattern*(n);

scan.close();

}

**private** **static** **void** printPattern(**int** n) {

// outer loop

**for** (**int** i = 1; i <= n; i++) {

// inner loop1 : spaces

**for** (**int** j = 1; j <= 2 \* (n - i); j++) {

System.***out***.print(" "); // single space

}

// inner loop2: stars

**for** (**int** k = 1; k <= 2 \* i - 1; k++) {

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

**for**(**int** i = 2; i <= n; i++)

{

**for**(**int** j = 1; j <= 2 \*(i-1); j++)

{

System.***out***.print(" "); //single space

}

**for**(**int** k=1; k <= 2\*(n-i)+1; k++)

{

System.***out***.print("\*" + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to print the hollow square pattern.

\* For example,

\* if n = 5,

\* \* \* \* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \* \* \* \*

\*/

**import** java.util.Scanner;

**public** **class** TestClass {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the n value to display hollow square pattern");

**int** n = scan.nextInt();

*printPattern*(n);

}

**private** **static** **void** printPattern(**int** n) {

**for**(**int** i = 1; i <= n; i++ )

{

**for**(**int** j = 1; j <= n; j++)

{

**if**(i == 1 || i == n || j == 1 || j == n)

System.***out***.print("\*" + " ");

**else**

System.***out***.print(" "); //2 spaces

}

System.***out***.println();

}

}

}

/\*

\* write a program to print the pascal's triangle

\* if n = 5,

\* 1

\* 1 1

\* 1 2 1

\* 1 3 3 1

\* 1 4 6 4 1

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the n value to display pascal's triangle");

**int** n = scan.nextInt();

*printPascalTriangle*(n);

}

**private** **static** **void** printPascalTriangle(**int** n) {

**for**(**int** i = 0; i < n; i++ )

{

**for**(**int** j = 1; j <= n - i - 1; j++)

{

System.***out***.print(" "); //single space

}

**for**(**int** k = 0; k <= i; k++)

{

**int** element = *factorial*(i) / ( *factorial*(k) \* *factorial*(i-k) );

System.***out***.print(element + " ");

}

System.***out***.println();

}

}

**private** **static** **int** factorial(**int** i) {

**int** fact = 1;

**for**(**int** x =1; x <= i; x++) {

fact = fact \* x;

}

**return** fact;

}

}

/\*

\* write a program to print the below pattern.

\* For example,

\* if n = 5,

\* A

\* A B

\* A B C

\* A B C D

\* A B C D E

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the no of rows to display character pattern");

**int** n = scan.nextInt();

*printPattern*(n);

}

**private** **static** **void** printPattern(**int** n) {

**for**(**int** i = 1; i <= n; i++) {

**for**(**char** ch = 'A'; ch <= 'A' + i - 1; ch++) {

System.***out***.print(ch + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to print the below pattern.

\* For example,

\* if n = 5,

\* A

\* B A

\* C B A

\* D C B A

\* E D C B A

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter the no of rows to display character pattern");

**int** n = scan.nextInt();

*printPattern*(n);

}

**private** **static** **void** printPattern(**int** n) {

**for**(**int** i = 1; i <= n; i++) {

**for**(**int** j = 1; j <= 2 \*(n-i); j++) {

System.***out***.print(" ");

}

**for**(**char** ch = (**char**)('A'+i-1); ch >= 'A'; ch--)

{

System.***out***.print(ch + " ");

}

System.***out***.println();

}

}

}

/\*

\* write a program to print the prime numbers

\* with in a given range.

\* For example,

\* if range is 10 to 20, then

\* output:

\* 11

\* 13

\* 17

\* 19

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter lower bound");

**int** x = scan.nextInt();

System.***out***.println("Enter upper bound");

**int** y = scan.nextInt();

*printPrimeNumbers*(x, y);

}

**private** **static** **void** printPrimeNumbers(**int** x, **int** y) {

System.***out***.println("printing the prime numbers between " + x + " and " + y);

**for**(**int** i = x; i <= y; i++) {

**boolean** flag = **true**;

**for**(**int** j = 2; j <= Math.*sqrt*(i); j++) {

**if**( i % j == 0) {

flag = **false**;

**break**;

}

}

**if**(flag == **true**) {

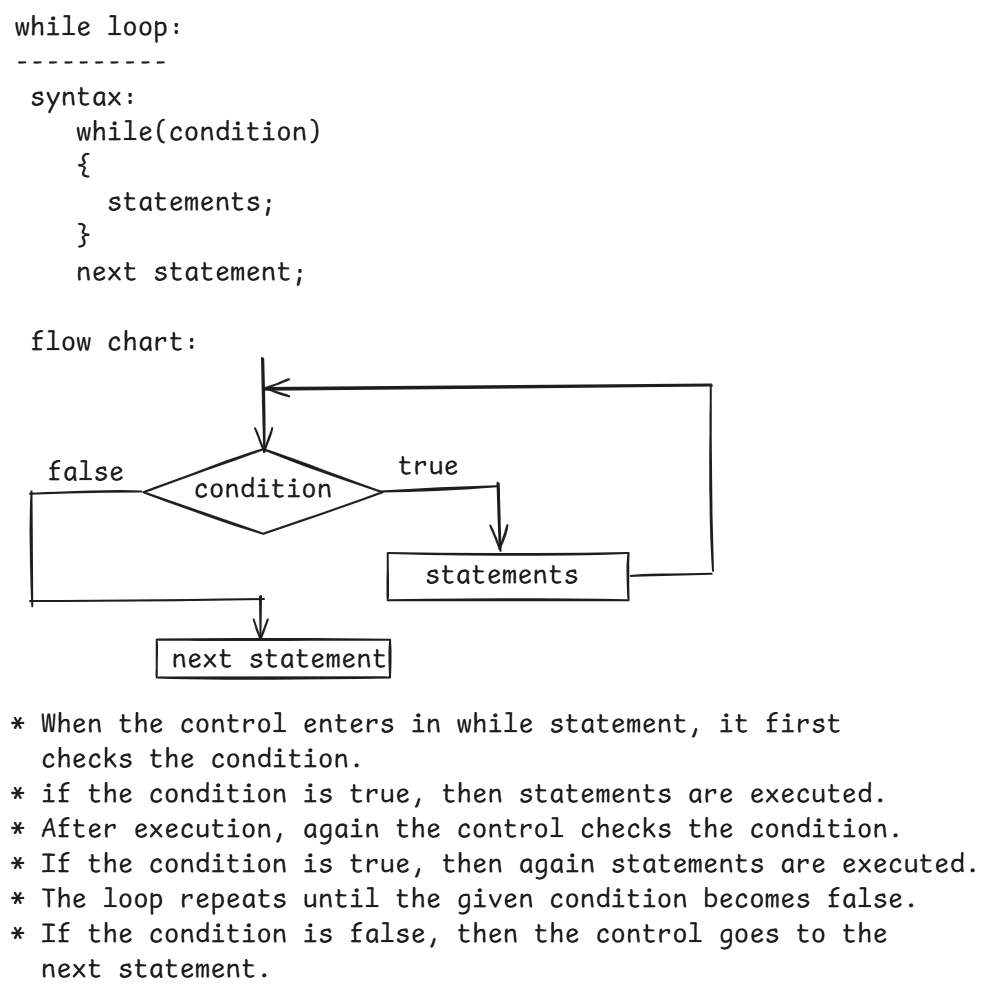
System.***out***.println(i);

}

}

}

}



/\*

\* write a program to find the sum of the digits of a given number

\* Example:

\* n = 158

\* output: 14

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n =scanner.nextInt();

*findSumOfDigits*(n);

}

**private** **static** **void** findSumOfDigits(**int** n) {

**int** sum = 0;

**while**(n != 0) {

**int** d = n % 10;

sum += d;

n = n / 10;

}

System.***out***.println("Sum of the digits = " + sum);

}

}

/\*

\* write a program to check whether a given number

\* is Armstrong number or not.

\* th

\* Armstrong number : If sum of the n power of each digit of a number

\* is equal to the same number, then it is called

\* Armstrong number.

\* example:

\* n = 153

\* 3 3 3

\* = 1 + 5 + 3 = 153

\* output: 153 : is Armstrong number

\*

\* n = 1634

\* 4 4 4 4

\* = 1 + 6 + 3 + 4 = 1634

\* output: 1634 : is Armstrong number

\*

\* n = 15

\* 2 2

\* = 1 + 5 = 26

\* output: 15 : is not an Armstrong number

\*/

**import** java.util.Scanner;

**public** **class** Armstrong {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**boolean** flag = *checkArmstrong*(n);

**if**(flag == **true**)

System.***out***.println(n + " : is Armstrong number");

**else**

System.***out***.println(n + " : is not Armstrong number");

}

**private** **static** **boolean** checkArmstrong(**int** n) {

**int** sum = 0;

//convert the number to string and find the length

String str = String.*valueOf*(n);

**int** length = str.length();

//store the number into another temp variable.

**int** temp = n;

**while**( n > 0) {

**int** d = n % 10;

sum = sum + (**int**)Math.*pow*(d, length);

n = n / 10;

}

**if** ( sum == temp )

**return** **true**;

**else**

**return** **false**;

}

}

/\*

\* write a program to check whether a given number

\* is a strong number or not.

\*

\* Example:

\* n = 2

\* = 2! = 2

\* output:

\* 2 : is a strong number

\*

\* n = 3

\* = 3! = 6

\* output:

\* 3 : is not a strong number

\*

\* n = 145

\* = 1! + 4! + 5! = 145

\* output:

\* 145: is a strong number

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**boolean** flag = *checkStrong*(n);

**if**(flag == **true**)

System.***out***.println(n + " : is strong number");

**else**

System.***out***.println(n + " : is not strong number");

}

**private** **static** **boolean** checkStrong(**int** n) {

**int** sum = 0;

**int** temp = n;

**while**(n > 0) {

**int** d = n % 10;

**int** fact = 1;

**for**(**int** i=1; i<=d; i++)

{

fact = fact \* i;

}

sum = sum + fact;

n = n / 10;

}

**if**(sum == temp)

**return** **true**;

**else**

**return** **false**;

}

}

/\*

\* write a program to check whether a given

\* number is palindrome or not.

\* palindrome: If reverse of a number is equal to the same number

\* then it is palindrome number.

\* example:

\* number = 121

\* reverse = 121

\* output: 121 : is a palindrome

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**boolean** flag = *checkPalindrome*(n);

**if**(flag == **true**)

System.***out***.println(n + " : is palindrome number");

**else**

System.***out***.println(n + " : is not palindrome number");

}

**private** **static** **boolean** checkPalindrome(**int** n) {

**int** reverse = 0;

**int** temp = n;

**while**(n > 0) {

**int** d = n % 10;

reverse = reverse \* 10 + d;

n = n / 10;

}

**if**(reverse == temp)

**return** **true**;

**else**

**return** **false**;

}

}

/\*

\* write a program to check a given number is

\* spy number or not.

\* spy number : If sum of the digits and product of the digits of

\* a number are equal, then it is a spy number.

\* example:

\* n = 123

\* sum = 1 + 2 + 3 = 6

\* product = 1 \* 2 \* 3 = 6

\* output:

\* 123: is a spy number

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a number");

**int** n = scan.nextInt();

**boolean** flag = *checkSpy*(n);

**if**(flag == **true**)

System.***out***.println(n + " : is spy number");

**else**

System.***out***.println(n + " : is not spy number");

}

**private** **static** **boolean** checkSpy(**int** n) {

**int** sum = 0, product = 1;

**while**(n > 0) {

**int** d = n % 10;

sum = sum + d;

product = product \* d;

n = n / 10;

}

**if**(sum == product)

**return** **true**;

**else**

**return** **false**;

}

}

/\*

\* write a program to convert binary number to the

\* decimal number.

\* example:

\* number = 1010

\* output: 10

\* number = 1111

\* output: 15

\*/

**import** java.util.Scanner;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a binary number");

**int** n = scan.nextInt();

*convertToDecimal*(n);

}

**private** **static** **void** convertToDecimal(**int** n) {

**int** sum = 0;

**int** i = 0;

**while**(n > 0)

{

**int** d = n % 10;

**int** t = d \* (**int**)Math.*pow*(2, i);

sum = sum + t;

i++;

n = n / 10;

}

System.***out***.println("Decimal number = " + sum);

}

}

do while loop:

-------------

* When you want to execute the statements of a loop for at least once, before checking the condition, then use do while loop.
* do while loop is also called exit control loop, because the condition in the do while loop tells when to exit from the loop.

syntax:

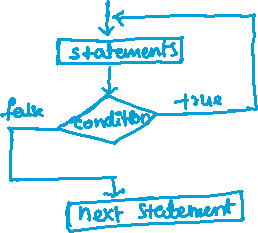
do {

statements;

}

while(condition);

next statement;



ex1;

int i=1;

do {

S.o.println(i);

i++;

}

while(i <= 5);

output:

1

2

3

4

5

ex2:

int n = 156;

int sum = 0;

do {

int d = n % 10;

sum = sum + d;

n = n / 10;

} while( n > 0);

S.o.println(sum);

output: 12

/\*

\* do while loop example with a menu

\*/

**import** java.util.Scanner;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

**int** choice;

**do** {

System.***out***.println("\*\*\*\*\*\*\*\*\*Menu\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("1. add");

System.***out***.println("2. multiply");

System.***out***.println("3. exit");

System.***out***.println("enter your choice");

choice = scan.nextInt();

**switch**(choice) {

**case** 1:

System.***out***.println("enter first number");

**int** a = scan.nextInt();

System.***out***.println("enter second number");

**int** b = scan.nextInt();

System.***out***.println("addition : " + (a+b));

**break**;

**case** 2:

System.***out***.println("enter first number");

**int** x = scan.nextInt();

System.***out***.println("enter second number");

**int** y = scan.nextInt();

System.***out***.println("multiplication : " + (x \* y));

**break**;

}

} **while**(choice != 3);

}

}

Jumping statements:

-------------------

break

continue

return

exit

break: This statement will move the control to the out of switch statement

or out of a loop.

If you don’t write break statement in a case of switch statement

then the controll executes further cases also.

ex1:

for(int i = 1; i <= 20; i++)

{

S.o.println(i);

if( i == 4)

break;

}

output:

1

2

3

4

ex2:

for(int i=1; i<=3; i++)

{

for(int j=1; j<=5; j++)

{

S.o.print(j + “ “);

if(j == 3)

break;

}

}

output:

1 2 3 1 2 3 1 2 3

ex3:

outer: **for**(**int** i=1; i<=3; i++)

{

inner: **for**(**int** j=1; j<=5; j++)

{

System.***out***.print(j + " ");

**if**(j == 3)

**break** outer;

}

}

output: 1 2 3

continue: This statement will skip the execution of remaining statements

and will move the control for the next iteration.

ex1:

for(int i=1; i<=5; i++)

{

if(i==3)

{

continue;

}

S.o.print(i + “ “);

}

output: 1 2 4 5

ex2:

outer: for(int i=1; i<=5; i++)

{

inner: for(int j=1; j<=5; j++)

{

if(i + j > 7)

continue outer;

S.o.print(i+j + “ “);

}

}

output:

2 3 4 5 6 3 4 5 6 7 4 5 6 7 5 6 7 6 7

return and exit :

* return statement will move the control from the current method to the calling point.
* if return type of a method is void, then the method should not return any value.
* when a method is called from another method, the control jumps to the called method, after execution the control goes back to the caller method.
* exit statement will terminate the current running JVM. So, the program execution will be terminated.
* You can write exit statement as System.exit(1);

Arrays

=========

* We use a variable to store the data
* We need to create multiple variables to store the multiple values.
* If mulitple variables are created, then the complexity will increase in a program. Identifying which has variable has which value is difficult.
* The variables are allocated at different places in memory. So, JVM has to search for each variable in memory, either to update the value or read the value. So, the performance of the application will be decreased.
* As a solution, we can arrays in the programs.
* An array is group of elements/items which are stored together under a single name.
* Each element of an array can be identified by using index and the index starts at 0.
* For the array elements, the memory is allocated in continuous memory locations, so JVM can fastly read or write the values.

array declaration & creation:

-----------------------------

datatype[] array\_name; //declaration

(or)

datatype array\_name[]; //declaration

array\_name = new datatype[size]; //creation

. we can combine declaration and creation of an array together.

datatype[] array\_name = new datatype[size];

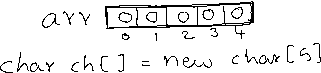
(or)

datatype array\_name[] = new datatype[size];

ex:

int[] arr = new int[5];

* when an array is created, it will initially store the elements as default values based on the datatype of the array.



ex:

int[] arr = new int[5];

arr[0] = 17;

arr[1] = 32;



arr[4] = 5;

* To find the length of an array, you can call length attribute.

System.out.println(arr.length); // output: 5

storing dynamic values into an array:

int[] arr = new int[5];

Scanner scan = new Scanner(System.in);

for(int i=0; i<arr.length; i++)

{

System.out.println(“enter element for index : “+ i);

arr[i] = scan.nextInt();

}

storing static values into an array:

int[] arr = new int[] { 10, 23, 18, 4, 16};

(or)

int[] arr = {10, 23, 18, 4, 16};

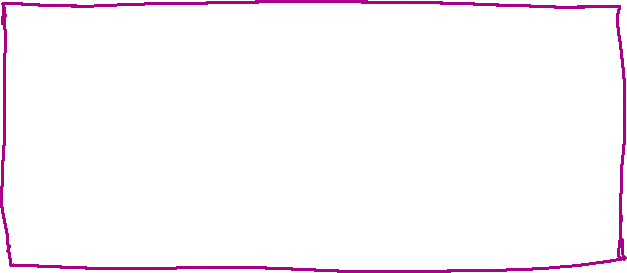
Q) what are the drawbacks of an array?

A) 1. The size of an array is fixed. It cannot grow or shrink dynamically.

2. inserting or deleting elements in an array in the middle is

inefficient. Because, it requires shifting the elements to either

left or right side, to maintain the order.



3. Arrays only support homogeneous data(same type), so arrays are less

flexible.

4. Arrays does not have built-in methods for common tasks like sorting,

searchig, resizing etc..

/\*

\* write a program to find the sum of the elements of an array

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter the size of an array");

**int** size = scan.nextInt();

// create an array with the given size

**int**[] arr = **new** **int**[size];

// storing the elements

**for** (**int** i = 0; i < arr.length; i++) {

System.***out***.println("enter element for index : " + i);

arr[i] = scan.nextInt();

}

*findSum*(arr);

}

**private** **static** **void** findSum(**int**[] arr) {

**int** sum = 0;

**for** (**int** i = 0; i < arr.length; i++) {

sum = sum + arr[i];

}

System.***out***.println("sum = " + sum);

}

}

/\*

\* write a program to search for an element in the given

\* array with linear search.

\*

\* linear search: It means, compare the each array element with searching element.

\* If they are equal then search is successful.

\* If no element in the array matches to the searching element then

\* search is unsuccessful and element is not found.

\*/

**import** java.util.Scanner;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter the size of an array");

**int** size = scan.nextInt();

// create an array with the given size

**int**[] arr = **new** **int**[size];

// storing the elements into array

**for** (**int** i = 0; i < arr.length; i++) {

System.***out***.println("enter element for index : " + i);

arr[i] = scan.nextInt();

}

System.***out***.println("enter searching element");

**int** searchingElement = scan.nextInt();

*linearSearch*(arr, searchingElement);

}

**private** **static** **void** linearSearch(**int**[] arr, **int** searchingElement) {

**boolean** flag = **false**;

**for**(**int** i=0; i<arr.length; i++)

{

**if**(arr[i] == searchingElement) {

System.***out***.println("Element is found at index : " + i);

flag = **true**;

**break**;

}

}

**if**(flag == **false**)

System.***out***.println("Element is not found.");

}

}

/\*

\* write a program to search for an element in the given

\* array using binary search.

\* Binary search:

\* steps:

\* 1. sort the array elements in ascending order

\* 2. initialize low and high variables with indexes.

\* 3. calculate mid point (low+high)/2

\* 4. if searchingElement > mid element then change low=mid+1

\* 5. if searchingElement < mid element then change high=mid-1

\* 6. if searchingElement == mid element then display "element found".

\* 7. the steps 3 to 6 should repeat until low <= high

\* 8. if low > high then display "element not found".

\*/

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter the size of an array");

**int** size = scan.nextInt();

// create an array with the given size

**int**[] arr = **new** **int**[size];

// storing the elements into array

**for** (**int** i = 0; i < arr.length; i++) {

System.***out***.println("enter element for index : " + i);

arr[i] = scan.nextInt();

}

System.***out***.println("enter searching element");

**int** searchingElement = scan.nextInt();

*binarySearch*(arr, searchingElement);

}

**private** **static** **void** binarySearch(**int**[] arr, **int** searchingElement) {

// sort the array elements in ascending order

Arrays.*sort*(arr);

**int** low = 0;

**int** high = arr.length - 1;

**boolean** flag = **false**;

**while** (low <= high) {

**int** mid = (low + high) / 2;

**if** (searchingElement > arr[mid])

low = mid + 1;

**else** **if** (searchingElement < arr[mid])

high = mid - 1;

**else** {

System.***out***.println("element found!");

flag = **true**;

**break**;

}

}

**if** (flag == **false**) {

System.***out***.println("element not found!!");

}

}

}

/\*

\* write a program to find the max consecutively repeated

\* element of the array.

\* For example:

\* int[] arr = {2, 1, 1, 2, 3, 3, 3};

\* output: element is : 3 and repeated for : 3 times consecutively.

\*/

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

**int**[] arr = { 2, 2, 1, 1, 1, 3, 4, 4, 4, 4, 4, 2, 2, 2, 2};

*findMaxConsecutive*(arr);

}

**private** **static** **void** findMaxConsecutive(**int**[] arr) {

**int** maxi = 0, count = 1, element = 0;

**for**(**int** i=0; i<arr.length-1; i++)

{

**if**(arr[i] == arr[i+1])

count++;

**else**

count=1;

**if**(count > maxi)

{

maxi=count;

element = arr[i];

}

}

System.***out***.println("Element : " + element + ", is max consecutively repeated for : "+ maxi + " times");

}

}

/\*

\* write a program to remove the duplicate elements from the array.

\* example:

\* int[] arr = { 4, 2, 4, 1, 5, 2};

\* output:

\* 4 2 1 5

\*/

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

**int**[] arr = { 4, 2, 1, 4, 3, 5, 2, 1 };

*removeDuplicates*(arr);

}

**private** **static** **void** removeDuplicates(**int**[] arr) {

**int**[] newArr = **new** **int**[arr.length];

**int** count = 0;

**for** (**int** i = 0; i < arr.length; i++) {

**boolean** flag = **false**;

**for** (**int** j = 0; j < newArr.length; j++) {

**if** (arr[i] == newArr[j]) {

flag = **true**;

**break**;

}

}

**if** (flag == **false**) {

newArr[count] = arr[i];

count++;

}

}

System.***out***.println("Orginal array ");

**for** (**int** i = 0; i < arr.length; i++) {

System.***out***.print(arr[i] + " ");

}

System.***out***.println();

System.***out***.println("After removing the duplicates ");

**for** (**int** i = 0; i < count; i++) {

System.***out***.print(newArr[i] + " ");

}

}

}

finding the uninion of two arrays:

---------------------------------

* You can find the union of two arrays in 2 approaches.

approach-1:

1. remove the duplicate elements from array1
2. remove the duplicate elements from array2
3. add the array1 elements to the array3
4. add the array2 elements to the array3, if they are not exist.
5. display array3.

approach-2:

1. create HashSet class object.
2. add array1 elements to HashSet object
3. add array2 elements to HashSet object
4. copy the elements from HashSet object to array3.
5. display array3

/\*

\* The below code is based on approach-2

\*/

**import** java.util.HashSet;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

**int** array1[] = {2, 9, 0, 1, 2};

**int** array2[] = {3, 7, 1, 9, 5};

**int** array3[] = *findUnion*(array1, array2);

System.***out***.println("The union of array1 and array2 : ");

**for**(**int** element : array3)

{

System.***out***.print(element + " ");

}

}

**private** **static** **int**[] findUnion(**int**[] array1, **int**[] array2) {

//create HashSet class object

HashSet<Integer> hashSet = **new** HashSet<>();

//add array1 elements to HashSet object

**for**(**int** element : array1)

{

hashSet.add(element);

}

//add array2 elements to HashSet object

**for**(**int** element : array2)

{

hashSet.add(element);

}

//create array3

**int**[] array3 = **new** **int**[hashSet.size()];

**int** i = 0;

//copy the elements from HashSet object to array3

**for**(**int** x : hashSet)

{

array3[i] = x;

i++;

}

**return** array3;

}

}

Note: In Java, we have two types of for loop.

1. for loop 2. for each loop.

. The both types, the keyword is for only.

. sytax of for each loop,

for(datatype variable : array/collection)

{

statements;

}

finding the intersection of two arrays:

--------------------------------------

1. create HashSet class object
2. add array1 elements to the HashSet object
3. take each element from array2 and check is it exist in HashSet object
4. if exist then display, otherwise don’t display it.

**import** java.util.HashSet;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

**int** array1[] = { 4, 1, 7, 5, 3 };

**int** array2[] = { 9, 2, 1, 4, 5 };

*findIntersection*(array1, array2);

}

**private** **static** **void** findIntersection(**int**[] array1, **int**[] array2) {

//create HashSet class object

HashSet<Integer> hashSet = **new** HashSet<>();

//add array1 eleements to HashSet object

**for**(**int** element : array1) {

hashSet.add(element);

}

//take element from array2

**for**(**int** element : array2) {

//check is it exist in HashSet object

**if**(hashSet.contains(element))

{

//print the element

System.***out***.print(element + " ");

}

}

}

}

How to sort the array elements?

------------------------------

* we can sort the array elements with the below statement, by without writing any manual logic.

Arrays.sort(arr);

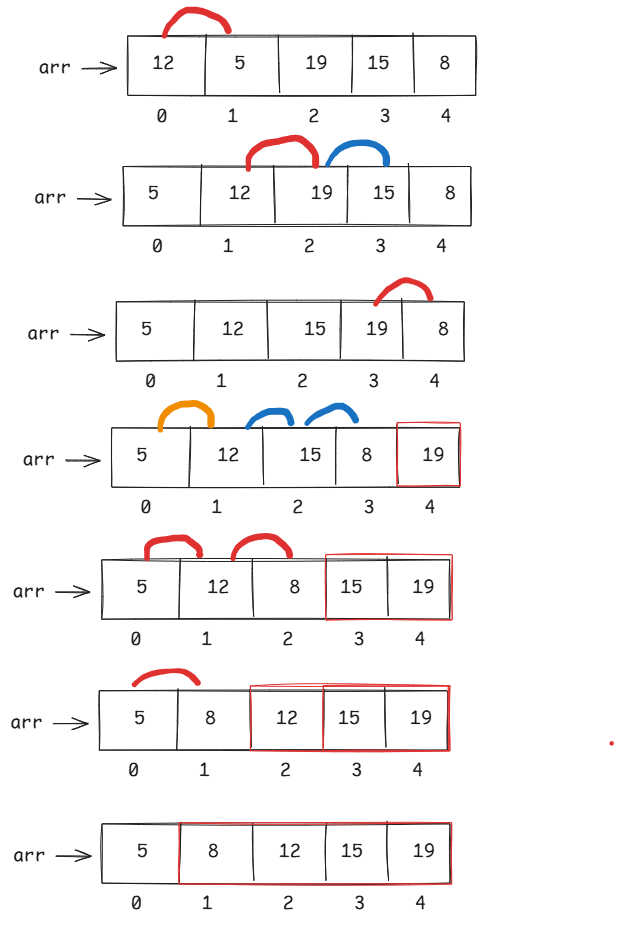
* we can manually write the sorting logic with a sorting algorithm like bubble sort or insertion sort or selection sort or merge sort, etc..

How bubble sort works?

---------------------

* The bubble sort technique works like below.

1. start from the beginning of the array
2. compare first element with second element and if first element is greater then swap them.
3. Move to the next pair, and repeat the same for all the array elements.
4. After one pass, the largest element is moved to the end of the array. It means, the largest element is bubbled.
5. Now again start from the beginning of the array, repeat the same process upto last but one element, so that second largest element is bubbled.
6. Repeat the process until n-1 elements are bubbled.



/\*

\* Bubble sort program

\*/

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

**int**[] arr = { 14, 8, 45, 7, 9 };

*bubbleSort*(arr);

}

**private** **static** **void** bubbleSort(**int**[] arr) {

**int** n = arr.length;

System.***out***.println("Array elements before sorting : ");

**for** (**int** i = 0; i < n; i++) {

System.***out***.print(arr[i] + " ");

}

System.***out***.println();

System.***out***.println("\*=".repeat(20));

**for** (**int** i = 0; i < n - 1; i++) {

**for** (**int** j = 0; j < n - 1 - i; j++) {

**if** (arr[j] > arr[j + 1]) {

**int** temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

System.***out***.println("Array elements after sorting : ");

**for** (**int** i = 0; i < n; i++) {

System.***out***.print(arr[i] + " ");

}

}

}

Two-dimentional array:

* A two-dimentional array contains rows and columns.
* A two-dimentional array is a collection of one-dimention arrays.
* A two-dimentional array is also called as a matrix.
* A two-dimentional array can be created like below.

datatype[][] arrayname = new datatype[rows][cols];

(or)

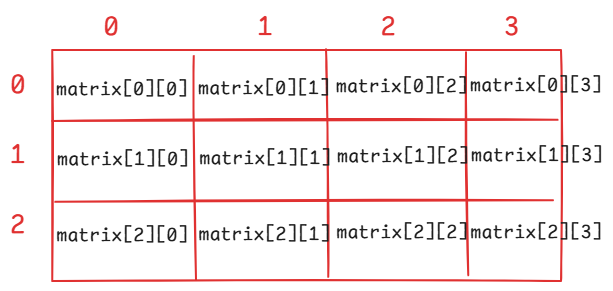
datatype arrayname[][] = new datatype[rows][cols];

(or)

datatype[] arrayname[] = new datatype[rows][cols];

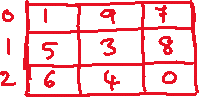
* example:

int[][] matrix = new int[3][4];



* A static two-dimentional array can be created as,

int[][] matrix = { {1, 9, 7}, {5, 3, 8}, {6, 4, 0} };



How to display the elements of this matrix?

for(int i = 0; i < matrix.length; i++)

{

for(int j = 0; j < matrix[i].length; j++)

{

System.out.print(matrix[i][j] + “ “);

}

System.out.println();

}

/\*

\* program to find the addition of two matrices

\*/

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

**int**[][] matrix1 = { {2, 1}, {3, 0}, {4, 9} };

**int**[][] matrix2 = { {8, 3}, {5, 2}, {7, 0} };

*addMatrices*(matrix1, matrix2);

}

**private** **static** **void** addMatrices(**int**[][] matrix1, **int**[][] matrix2) {

**int**[][] matrix3 = **new** **int**[matrix1.length][matrix1[0].length];

**for**(**int** i=0; i < matrix1.length; i++) {

**for**(**int** j=0; j < matrix1[0].length; j++) {

matrix3[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

**for**(**int** i=0; i < matrix3.length; i++) {

**for**(**int** j=0; j < matrix3[i].length; j++) {

System.***out***.print(matrix3[i][j] + " ");

}

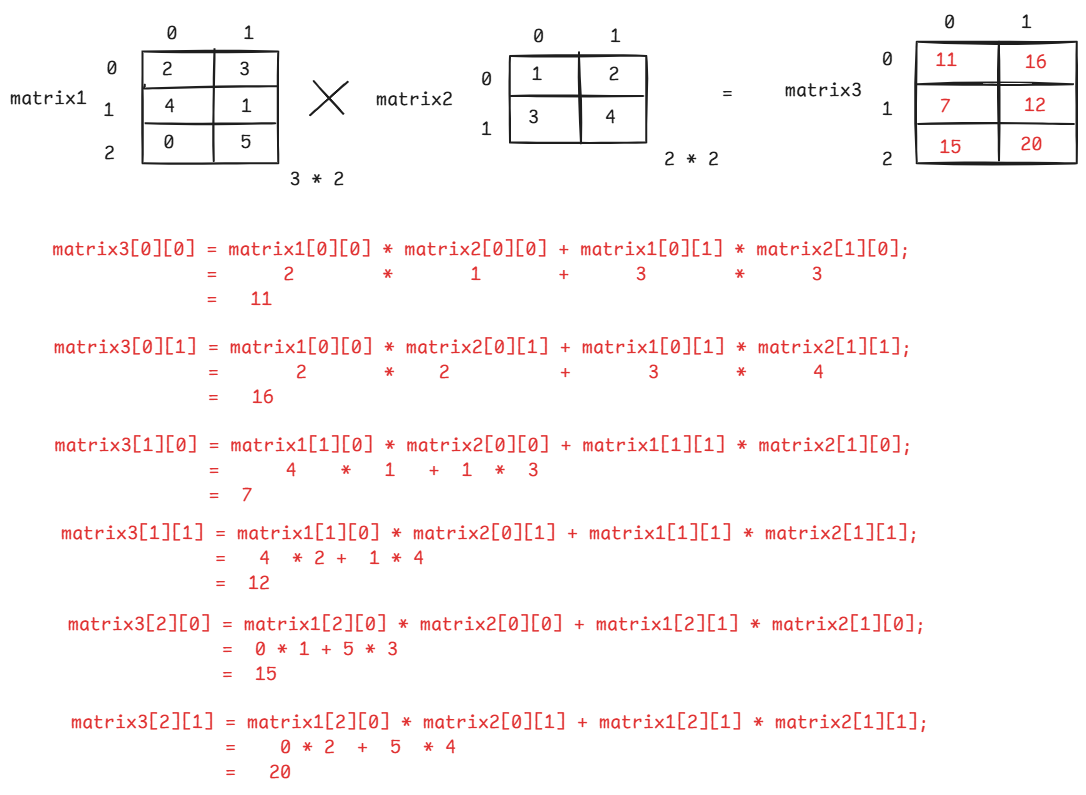
System.***out***.println();

}

}

}

matrix multiplication:



/\*

\* program to multiply the matrices

\* Note: rule is ,

\* no of columns in first matrix = no of rows in second matrix

\*/

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

**int**[][] matrix1 = { {2, 3}, {1, 1}, {0, 2} };

**int**[][] matrix2 = { {0, 0}, { 4, 5} };

*multiplyMatrices*(matrix1, matrix2);

}

**private** **static** **void** multiplyMatrices(**int**[][] matrix1, **int**[][] matrix2) {

**int**[][] matrix3 = **new** **int**[matrix1.length][matrix2[0].length];

**for**(**int** i=0; i<matrix3.length; i++) {

**for**(**int** j=0; j<matrix3[i].length; j++) {

**int** r = 0;

**for**(**int** k=0; k < matrix2.length; k++) {

r = r + matrix1[i][k] \* matrix2[k][j];

}

matrix3[i][j] = r;

}

}

**for**(**int** i=0; i<matrix3.length; i++) {

**for**(**int** j=0; j<matrix3[i].length; j++) {

System.***out***.print(matrix3[i][j] + " ");

}

System.***out***.println();

}

}

}

String handling

* Generally, a string is a sequence of characters enclosed within double quotes(“ “).
* In Java, to perform any operations on a string value, we should create an object for java.lang.String class.
* java.lang is the default package which is imported automatically into a Java program.
* In Java, two ways are provided, to create an object for a String class.

1. with literal form
2. with new keyword

* example:

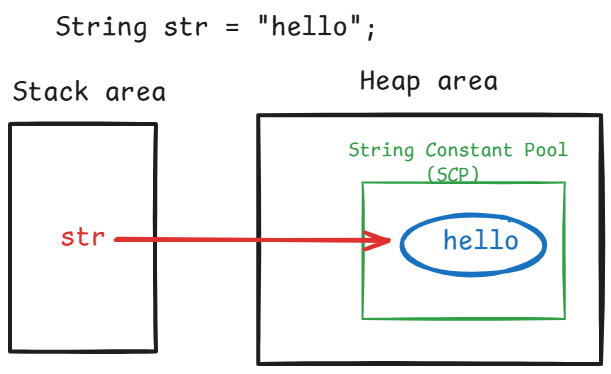
String str1 = “ashokit”; // with literal form

String str2 = new String(“john”); //with new keyword

* when you create a String object in the literal form, JVM will do the following.

1. First it checks for the string object in String Constant Pool(SCP).
2. If already exists then JVM simply creates the reference variable in the stack area and it links the reference variable with the object in the SCP.
3. If not exists, then JVM creates a new object in the SCP, then creates reference variable in the stack area and it links the reference variable with object in the SCP.

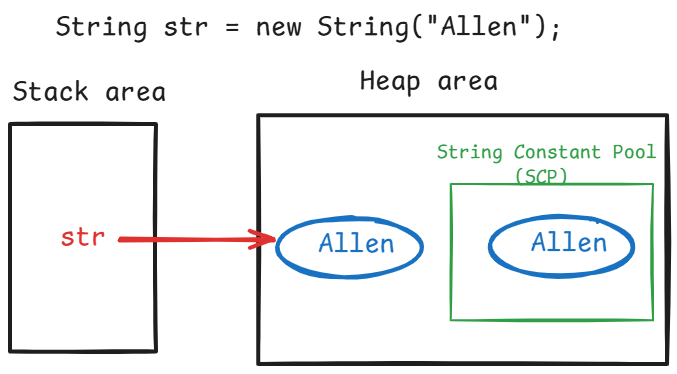
example:

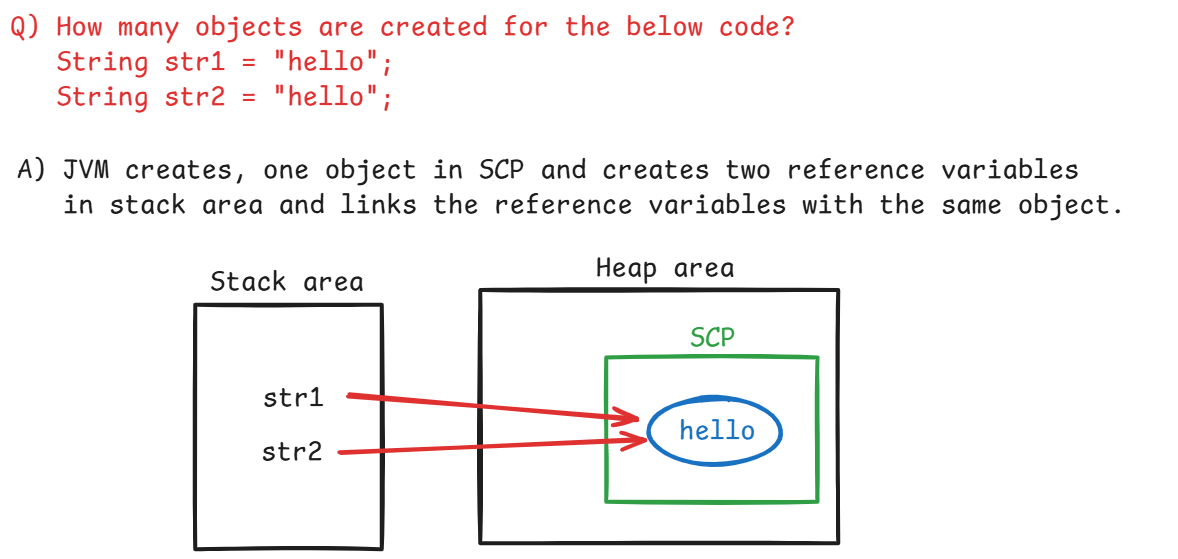


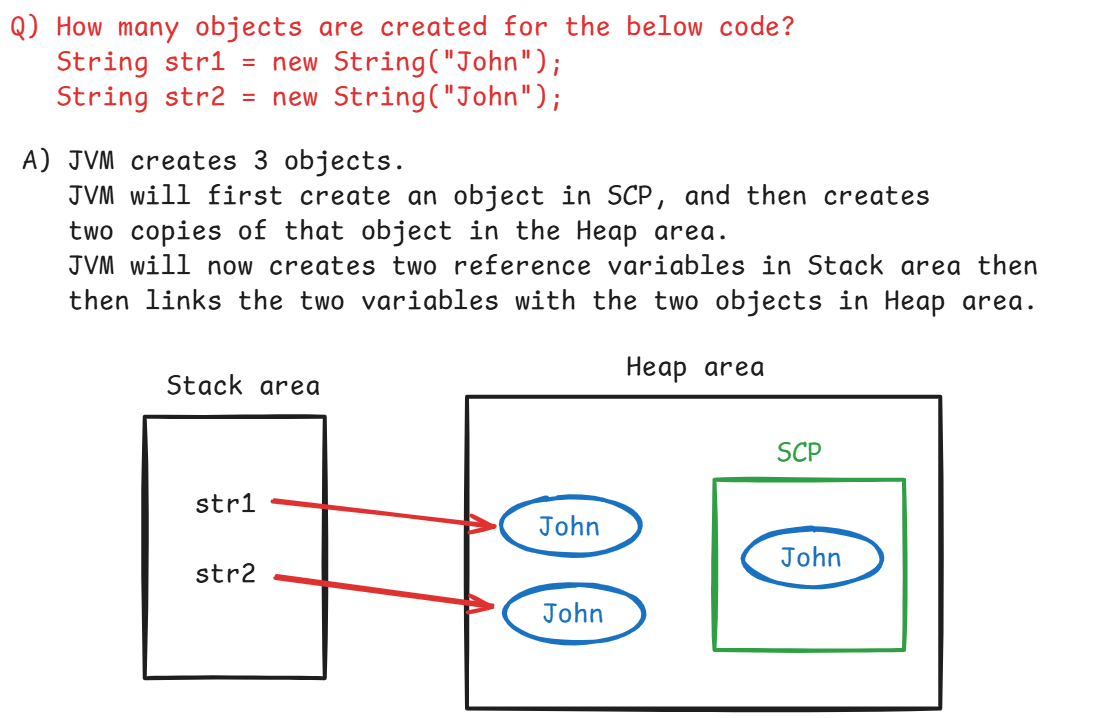
* When you create a String object with new keyword, then JVM will do the following.

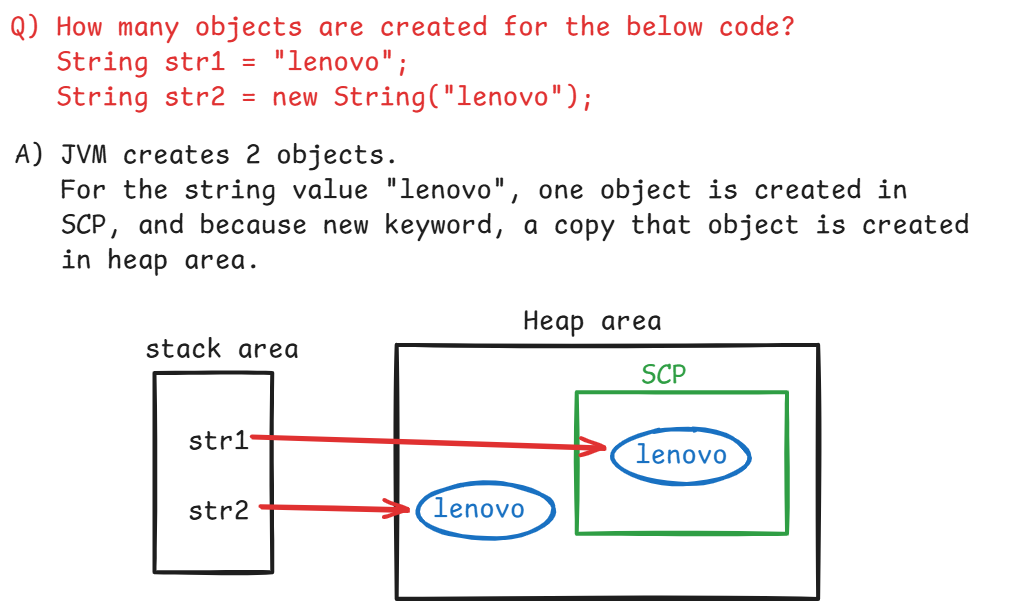
1. First it checks for the string object in the String Constant Pool.
2. If already exist, JVM creates a copy of that object in the Heap area, then creates reference variable in the stack area.
3. JVM links the reference variable with the object in Heap area.
4. If not exist, JVM create a new object in SCP, then a copy of that object in Heap area, then create reference variable in the stack area.
5. After that, JVM links the reference variable with the object in Heap area.

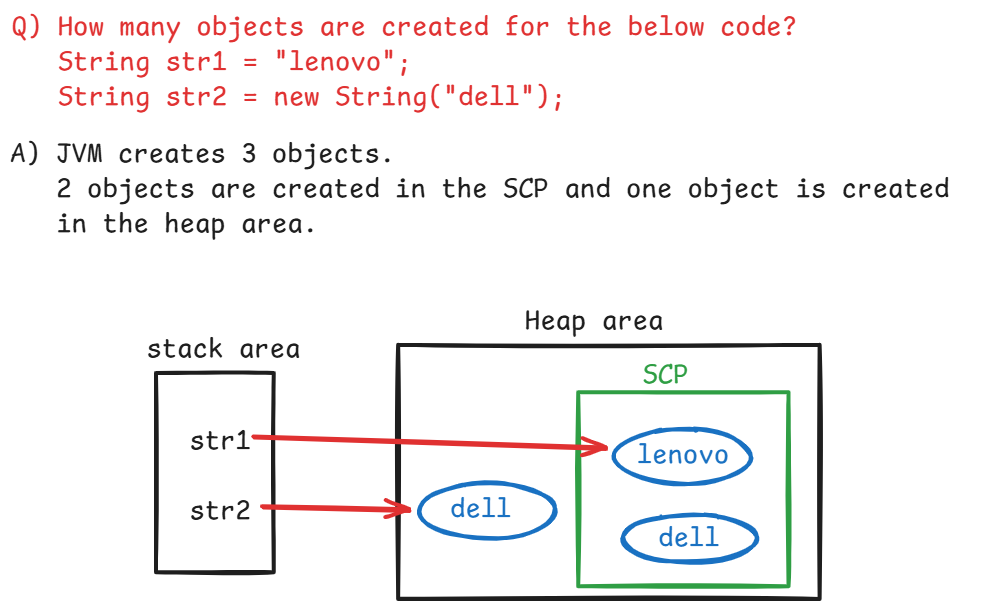
example:

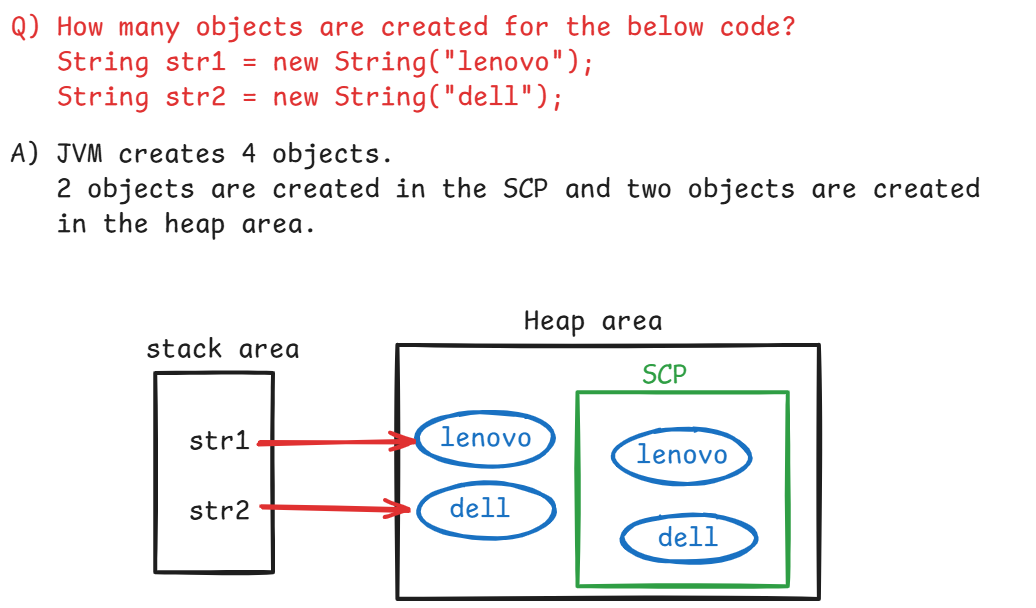


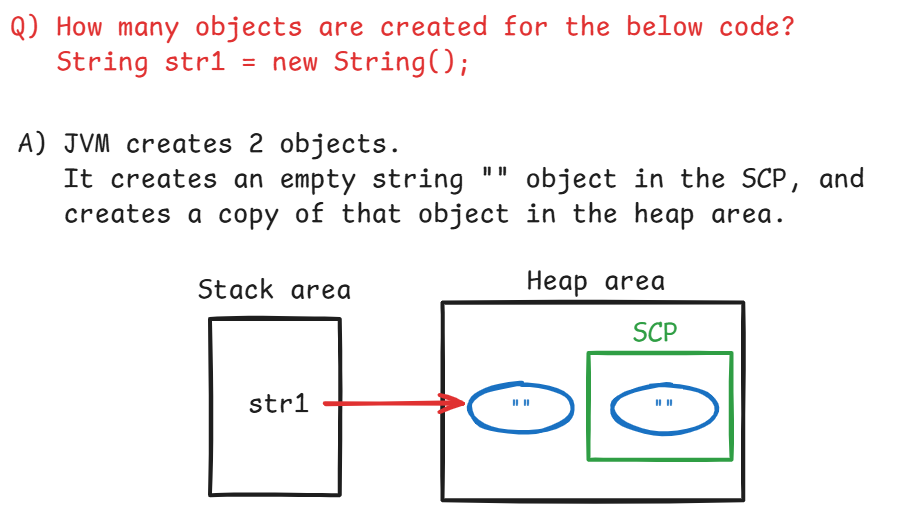












Q) How many objects are created for the below code?

String str1 = null;

1. no objects are created.

Q) String str1 = null;

System.out.println(str1);

A) Since str1 is null, it prints the string literal “null” to the console.

Finding the length of a string:

------------------------------

length() : returns the number of characters in the string value.

ex1:

String str1 = “Ashokit”;

System.out.println(str1.length()); // 7

String str2 = “Ashok it”;

System.out.println(str2.length()); // 8

ex2:

String[] fruits = { “Banana”, “Apple”, “Mango”, “Grapes” };

System.out.println(fruits.length()); // error

System.out.println(fruits.length); // 4

System.out.println(fruits[2].length); // error

System.out.println(fruits[2].length()); // 5

comparing the two strings:

------------------------

* There are 4 ways to compare the two strings.

1. equality (==) operator.
2. equals() method
3. equalsIgnoreCase() method
4. compareTo() method

equality(==) operator:

. It checks whether the two string reference variables are pointing

to the same object or not. If yes then returns true, otherwise

returns false.

ex1:

String str1 = new String(“apple”);

String str2 = new String(“apple”);

System.out.println(str1 == str2); // false

ex2:

String str1 = new String(“sony”);

String str2 = str1;

System.out.println(str1 == str2); // true

ex3:

String str1 = “Samsung”;

String str2 = “samsung”;

System.out.println(str1 == str2); // false

ex4:

String str1 = null;

String str2 = null;

System.out.println(str1 == str2); // true

equals() method:

. It compares the content in the two string objects. If the content is same, returns true. Otherwise, returns false.

ex1:

String s1 = “Lenovo”;

String s2 = “Lenovo”;

System.out.println(s1.equals(s2)); // true

ex2:

String s1 = new String(“dell”);

String s2 = new String(“dell”);

System.out.println(s1.equals(s2)); //true

ex3:

String s1 = new String(“Dell”);

String s2 = new String(“dell”);

System.out.println(s1.equals(s2)); //false

ex4:

String s1 = null;

String s2 = “null”;

System.out.println(s1.equals(s2)); // NullPointerException

System.out.println(s2.equals(s1)); // false

equalsIgnoreCase() method:

. It compares the content of the two string object by without considering the case.

ex1:

String s1 = “Lenovo”;

String s2 = “lenovo”;

System.out.println(s1.equalsIgnoreCase(s2)); //true

compareTo() method:

. It returns a negitive or zero or positive integer value.

. If string1 is less than string2, returns negitive integer.

. If string1 is equal to string2, returns zero.

. If string1 is greater than string2, returns positive integer.

. This is also called as “lexicographical” comparision.

ex1:

String s1 = “ABC”;

String s2 = “BCD”;

System.out.println(s1.compareTo(s2)); // -1

System.out.println(s2.compareTo(s1)); // 1

ex2:

String str1 = “ashokit”;

String str2 = “ashokit”;

System.out.println(s1.compareTo(s2)); // 0

charAt(), indexOf(), lastIndexOf() methods:

. charAt(index) returns the character at the given index.

ex:

String s1 = “hello”;

System.out.println(s1.charAt(4)); // o

System.out.println(s1.charAt(5)); // IndexOutOfBoundsException

. indexOf(char) returns the index of the first occurrence of the

specified character. If does not exist, returns -1.

ex:

String str = “hello”;

System.out.println(str.indexOf(‘l’)); // 2

System.out.println(str.indexOf(‘k’)); // -1

. indexOf(char, fromIndex) returns the index of the first occurrence

of the specified character, starting the search from

the specified index.

ex:

String str = “cat sat on mat”;

System.out.println(str.indexOf(‘a’,3)); // 5

. lastIndexOf(char) returns the index of the last occurrence of the

specified character. If does not exist, returns -1.

ex:

String str = “cat sat on mat”;

System.out.println(str.lastIndexOf(‘a’)); //12

substring() method:

. It returns a part/ a portion of a given string.

. If you specify only start index, then it returns the string from start index to the end of the string.

. If you specify both start index and end index then it returns

the string from start index to end index – 1.

ex:

String s1 = “The sky is blue”;

System.out.println(s1.substring(4, 7)); //sky

System.out.println(s1.substring(4)); // sky is blue

split() method:

. It will split/divide a string into a string array of multiple

strings at the specified character.

ex:

String str = “The cat sat on mat”;

String[] strArr = str.split(“ “);

for(int i=0; i < strArr.length; i++)

{

System.out.println(strArr[i]);

}

output:

The

cat

sat

on

mat

join() method:

. It will join multiple strings of a string array into a single

string with the specified joining character.

ex:

String[] strArr = { “My”, “Cat”, “is”, “Cute” };

String newString = String.join(“--“, strArr);

System.out.println(newString);

output:

My--Cat--is--Cute

toCharArray() method:

. It converts a given string into a character array.

ex:

String str = “listen”;

char[] ch = str.toCharArray();

trim() method:

. It removes the white space characters before the start of the

first letter or after the end of the last letter.

ex:

String str = “ hello”;

System.out.println(str.trim()); //hello

ex:

String str = “cat sat”;

System.out.println(str.trim()); //cat sat

immutable object?

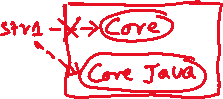
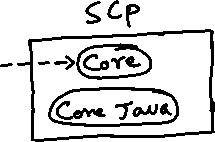
\* An object, which does not allow any changes to its data is called

immutable object.

\* A String object is also an immutable object. So, when you make any

changes to the data, JVM creates a new object for the modified

value.



toUpperCase() & toLowerCase():

-----------------------------

ex1:

String str1 = “sky is Blue”;

str1 = str1.toUpperCase();

S.o.println(str1); //SKY IS BLUE

ex2:

String str1 = “Dog IS BLACK”;

str1 = str1.toLowerCase();

S.o.println(str1); //dog is black

replace() method:

------------------

* It replaces the each substring of a string with a replacement string.

replace(substring, replacement)

ex1: String str1 = “The cat sat on mat”;

str1.replace(“cat”, “dog”);

S.o.p(str1);

output: The cat sat on mat

ex2: String str1 = “The cat sat on mat”;

str1 = str1.replace(“cat”, “dog”);

S.o.p(str1);

output: The dog sat on mat

replaceAll(): replaces each substring of a string that matches the specified regular expression with a given replacement string.

replaceAll(String regex, String replacement)

ex1:

String str = “My contact number : 123-456-7890”;

String result = str.replaceAll(“\\d”, “#”);

S.o.println(result);

output: My contact number : ###-###-####

Note: [\\d](file:///\\d) pattern matches to any digit(0-9)

ex2:

String str = “Spring is a framework”;

String result = str.replaceAll(“\\s”, “\_”);

S.o.println(result);

output: Spring\_is\_a\_framework

Note: [\\s](file:///\\s) pattern matches to a space.

ex3:

String str = “Java is a Programming language”;

String result = str.replaceAll(“\\s+”, “ “);

S.o.println(result);

output: Java is a Programming language

Note: [\\s](file:///\\s)+ pattern matches one or more spaces

ex4:

String str = “Java#@is$great!&”;

String result = str.replaceAll(“[^a-zA-Z0-9]”, “”);

S.o.println(result);

output: Javaisgreat

ex5:

String str = “Ashokit Solutions”;

String result = str.replaceAll(“[aeiouAEIOU]”, “”);

S.o.println(result);

output: shkt Sltns

ex6:

String creditCard=”1234-5678-9012-3456”;

String result = creditCard.replaceAll(“[^-](?=.{4})”, “X”);

S.o.println(result);

output: XXXX-XXXX-XXXX-3456

Note: (?=.{4}) pattern checks if there are 4 characters following the current character or not.

ex7:

String str = “<h1>Hello, <b>Shekher</b>!</h1>”;

String result = str.replaceAll(“<[a-zA-Z0-9/]>”, “”);

S.o.println(result);

output: Hello, Shekher!

isEmpty() & isBlank():

----------------------

* isEmpty() : returns true, if the string is empty. Otherwise returns false.
* isBlank(): returns true, if the string is empty after trim operation. Otherwise returns false.

ex1:

String str1 = “”;

S.o.println(str1.isEmpty()); //true

S.o.println(str1.isBlank()); //true

ex2:

String str1 = “ “;

S.o.println(str1.isEmpty()); //false

S.o.println(str1.isBlank()); //true

ex3:

String str1 = “Java”;

S.o.println(str1.isEmpty()); //false

S.o.println(str1.isBlank()); //false

startsWith() & endsWith():

-------------------------

* startsWith() returns true, if a string starts with the specified value. Otherwise, returns false.
* endsWith() returns true, if a string ends with the specified value, Otherwise, returns false.

ex:

String str1 = “admin@ashokit.com”;

System.out.println(str1.startsWith(“admin”)); //true

System.out.println(str1.startsWith(“ashok”)); //false

System.out.println(str1.endsWith(“.com”)); //true

System.out.println(str1.endsWith(“.in”)); //false

Converting a value from string type to primitive type:

-----------------------------------------------------

ex1:

String str = “1214”;

int k = Integer.parseInt(str); //converts a string to int

ex2:

String str = “3.78”;

double d = Double.parseDouble(str); //converts a string to double

ex3:

String str = “2.89f”;

float f = Float.parseFloat(str); //converts a string to float

ex4:

String str = “true”;

boolean b = Boolean.parseBoolean(str); //converts a string to boolean

converting a value from a primitive type to a string type:

-----------------------------------------------------------

* In String class, a static method is provided called valueOf(), which converts a primitive type to a string type.

ex1:

int x = 149;

String str = String.valueOf(x); //converts an int to string

ex2:

double y = 3.26;

String str = String.valueOf(y); //converts a double to string

ex3:

S.o.println(String.valueOf(10)+String.valueOf(12.55));

output:

1012.55

//demo program

**public** **class** TestClass {

**public** **static** **void** main(String[] args) {

//equality operator(==)

String str1 = **new** String("ashokit");

String str2 = **new** String("ashokit");

System.***out***.println("str1 == str2 : " + (str1 == str2));

//equals() method

String str3 = "hello";

String str4 = **new** String("hello");

System.***out***.println("str3.equals(str4) : " + str3.equals(str4));

//replaceAll() method

String creditCard="1234-5678-9012-3456";

String result = creditCard.replaceAll("[^-](?=.{4})", "X");

System.***out***.println("masked credit card : " + result);

//substring() method

String str5 = "The sky is blue";

String str6 = str5.substring(4, 7);

System.***out***.println("substring : " + str6);

//join() method

String[] str7 = { "Spring", "Boot", "with", "Microservices" };

String str8 = String.*join*("-", str7);

System.***out***.println("joined string : " + str8);

}

}

/\*

\* write a program to count the vowels and consonents in a given string

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("enter a string");

String str = scan.nextLine();

*findVowelsConsonentsCount*(str);

}

**private** **static** **void** findVowelsConsonentsCount(String str) {

str = str.toLowerCase();

**int** vowels = 0, consonents = 0;

**for**(**int** i = 0; i < str.length(); i++)

{

**char** ch = str.charAt(i);

**if**(Character.*isLetter*(ch))

{

**if**(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch =='u')

vowels++;

**else**

consonents++;

}

}

System.***out***.println("Vowels count = " + vowels);

System.***out***.println("Consonents count = " + consonents);

}

}

/\*

\* write a program to check whether a given string is

\* palindrome or not.

\* ex:

\* String str = "madam";

\* output: palindrome

\* String str = "madan";

\* output: not a palindrome

\* String str = "malayalam";

\* output: palindrome

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a string");

String str = scan.nextLine();

**boolean** b = *checkPalindrome*(str);

**if**(b == **true**) {

System.***out***.println("The given string is palindrome");

}

**else** {

System.***out***.println("The given string is not a palindrome");

}

}

**private** **static** **boolean** checkPalindrome(String str) {

str = str.toLowerCase();

**char**[] ch = str.toCharArray();

**boolean** flag = **true**;

**for**(**int** i=0, j=ch.length-1; i <= j; i++, j--) {

**if**(ch[i] != ch[j]) {

flag = **false**;

**break**;

}

}

**return** flag;

}

}

/\*

\* write a program to check whether the given two strings

\* are anagrams or not.

\* ex:

\* s1 = "Silent"

\* s2 = "Listen"

\* output: Both are anagrams

\* s3 = "hello";

\* s4 = "hallo";

\* output" both are not anagrams.

\*

\* Anagrams means, both the strings must contain same set of characters.

\*/

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter first string");

String str1 = scan.nextLine();

System.***out***.println("Enter second string");

String str2 = scan.nextLine();

**boolean** flag = *checkAnagrams*(str1, str2);

**if**(flag == **true**) {

System.***out***.println("Both the strings are anagrams");

}

**else** {

System.***out***.println("Both the strings are not anagrams");

}

}

**private** **static** **boolean** checkAnagrams(String str1, String str2) {

**if**(str1.length() != str2.length())

**return** **false**;

str1 = str1.toLowerCase();

str2 = str2.toLowerCase();

**char** ch1[] = str1.toCharArray();

**char** ch2[] = str2.toCharArray();

Arrays.*sort*(ch1);

Arrays.*sort*(ch2);

**boolean** flag = **true**;

**for**(**int** i = 0; i < ch1.length; i++) {

**if**(ch1[i] != ch2[i])

{

flag = **false**;

**break**;

}

}

**return** flag;

}

}

/\*

\* write a program to find the frequency of each character in a given string

\* frequency of each character means, count how many times each character is

\* repeated in the given string.

\* Example:

\* String str = "abcdba";

\* output:

\* a -- 2

\* b -- 2

\* c -- 1

\* d -- 1

\*/

**import** java.util.Arrays;

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a string");

String str = scan.nextLine();

*findFrequency*(str);

}

**private** **static** **void** findFrequency(String str) {

//convert the string to lower case

str = str.toLowerCase();

//convert the string to character array

**char**[] ch = str.toCharArray();

//sort the character array

Arrays.*sort*(ch);

**for**(**int** i = 0; i < ch.length; i++)

{

**int** count = 0;

**int** j;

**for**(j = i; j < ch.length; j++)

{

**if**( ch[i] != ch[j] )

{

i = j-1;

**break**;

}

**else**

{

count++;

}

}

System.***out***.println(ch[i] + " -- " + count);

**if**(j == ch.length)

**break**;

}

}

}

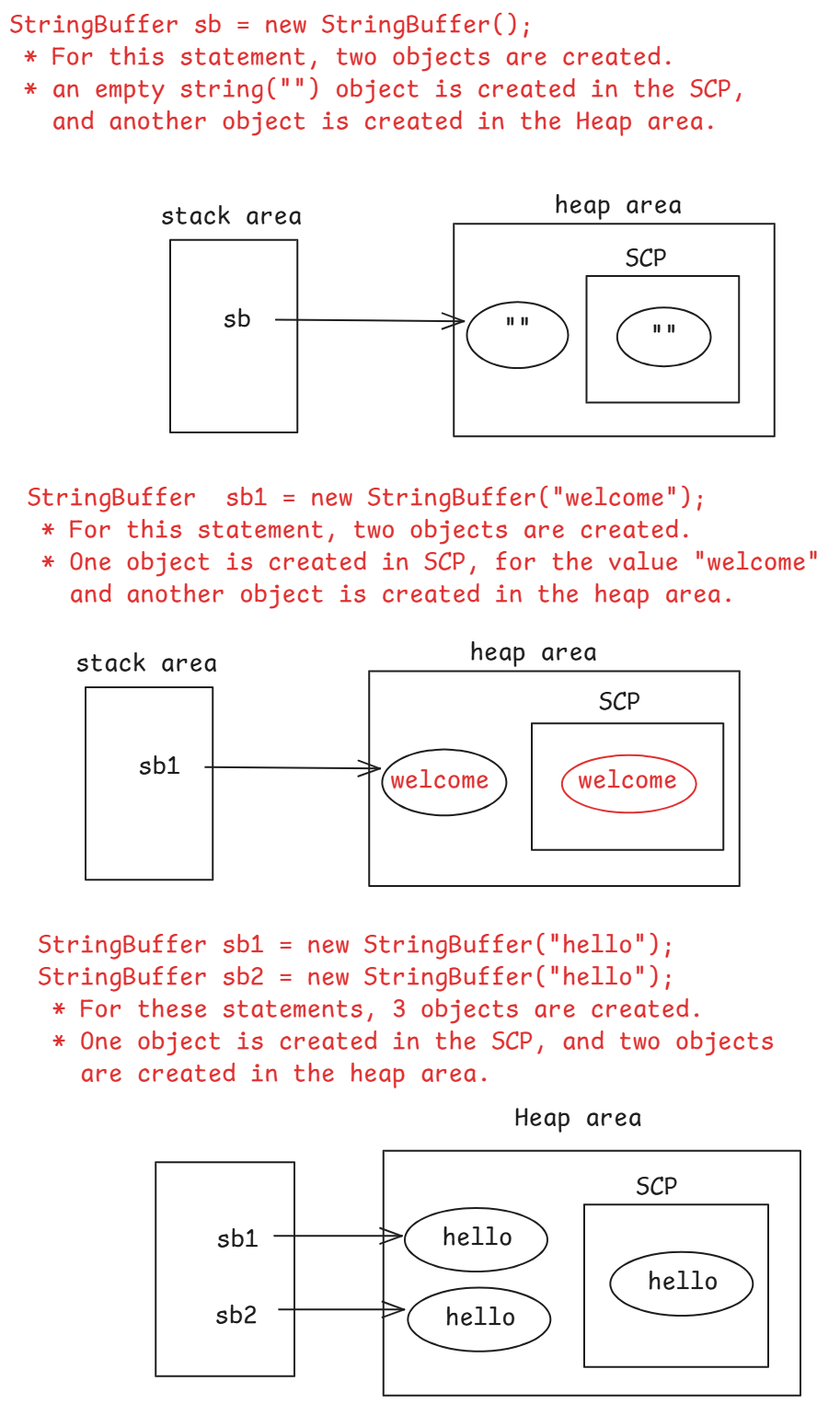
StringBuffer class:

------------------

* StringBuffer is a class provided in java.lang package
* StringBuffer also works on sequence of characters like String class.
* StringBuffer class objects are mutable and thread-safe objects.
* String class objects are immutable and thread-safe objects.
* A mutable object means, the object will allow to make the changes to its data.
* We can create a StringBuffer object, only with new keyword.

StringBuffer sb1 = “welcome”; //error

StringBuffer sb2 = new StringBuffer(“welcome”); //correct



Comparing the two StringBuffer objects:

--------------------------------------

ex:

StringBuffer sb1 = new StringBuffer(“hello”);

StringBuffer sb2 = new StringBuffer(“hello”);

System.out.println(sb1 == sb2); //false

System.out.println(sb1.equals(sb2)); //false

* equals() method of StringBuffer class, internally uses equality operator(==) only. So, the equals() method has returned false.
* So, when you want to compare the data in two StringBuffer objects, you need to convert the StringBuffer objects into String objects, by calling toString() method.

ex:

StringBuffer sb1 = new StringBuffer(“hello”);

StringBuffer sb2 = new StringBuffer(“hello”);

String s1 = sb1.toString();

String s2 = sb2.toString();

System.out.println(s1.equals(s2)); //true

StringBuffer capacity:

---------------------

* When you create a StringBuffer object, JVM allocates some additional space to allow the changes into the StringBuffer object.
* When you create an empty StringBuffer object, JVM allocates capacity for that object as 16.
* StringBuffer sb1 = **new** StringBuffer();
* System.***out***.println(sb1.capacity()); //output: 16
* You can also specify the capacity while creating StringBuffer object.

StringBuffer sb2 = **new** StringBuffer(10);

System.***out***.println(sb2.capacity()); //output: 10

* If you specify a string value, while creating StringBuffer object, then the capacity will be

length of the string + 16.

StringBuffer sb3 = **new** StringBuffer("hello");

//capacity = (length of the string) + 16

System.***out***.println(sb3.capacity()); //output: 21

* If you append more characters, exceeding the capacity then the capacity will be resized with a formula,

new capacity = (old capacity \* 2) + 2

sb3.append("abcdefghijklmnopq"); //appended 17 characters,

length = 22

//capacity = (old capacity) \* 2 + 2

System.***out***.println(sb3.capacity()); //output:44

* The capacity resize will happen for first time with the formula. From the next time, it may happen, or the capacity will be same as the length.
* sb3.append("ababababababababababababab"); //appended 26 characters, length = 48
* //capacity = length of the string
* System.***out***.println(sb3.capacity()); //output: 90 or 48

some key methods of StringBuffer class:

--------------------------------------

1. append() : This method appends an array or a string or a primitive value to the StringBuffer object.

ex:

StringBuffer sb = new StringBuffer(“hello”);

sb.append(“ world!”);

S.o.println(sb); // hello world!

StringBuffer sb = new StringBuffer(“hello”);

sb.append(true);

S.o.println(sb); //hellotrue

1. insert() : This method inserts a value in the middle of the StringBuffer object at a specified index.

ex:

StringBuffer sb = new StringBuffer(“The sat”);

sb.insert(4, “cat “);

S.o.println(sb); // The cat sat

1. delete(star, end): This method deletes the characters from start to end-1.

ex:

StringBuffer sb = new StringBuffer(“The sky is blue”);

sb.delete(3, 10);

S.o.println(sb); //The blue

1. replace(start, end, replacement): It will replace the substring between start to end-1 with replacement.

ex:

StringBuffer sb = new StringBuffer(“The cat sat”);

sb.replace(4, 7, “dog”);

S.o.println(sb); //The dog sat

1. reverse() : It will reverse a string value of the StringBuffer object.

ex:

StringBuffer sb = new StringBuffer(“hello”);

StringBuffer sb2 = sb.reverse();

S.o.println(sb2); //olleh

StringPalindrome program code with StringBuffer:

---------------------------------------------------

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a string");

String str = scan.nextLine();

//convert the string to lowercase

str = str.toLowerCase();

//create a StringBuffer object

StringBuffer sb1 = **new** StringBuffer(str);

StringBuffer temp = **new** StringBuffer(sb1);

//reverse

sb1.reverse();

//compare

**if**(sb1.toString().equals(temp.toString())) {

System.***out***.println("The given string is palindrome");

}

**else** {

System.***out***.println("The given string is not a palindrome");

}

}

}

StringBuilder class:

. StringBuilder class is provided in java.lang package.

. StringBuilder class also works on sequence of

characters.

. StringBuilder object is a mutable and not a thread safe

object.

. StringBuilder class methods are same as StringBuffer

class methods. But, StringBuffer class methods are

synchronized methods and StringBuilder class methods

are not synchronized methods.

. StringBuffer objects are recommended to use in

multithreading applications and StringBuilder objects

are recommended to use in the single thread

applications.

Q) what is the difference between String and StringBuffer?

A) String is an immutable and thread-safe object. StringBuffer is mutable and thread-safe object.

Q) what is the difference between StringBuffer and StringBuider?

A) StringBuffer is mutable and thread-safe object. StringBuilder is mutable and not a thread-safe object.

Q) what is the difference between String and StringBuilder?

A) String is an immutable and thead-safe object. StringBuilder is mutable and not a thread-safe object.

//write a program to check string palindrome with StringBuilder

//without calling reverse() method.

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a string");

String str = scan.nextLine();

str = str.toLowerCase();

StringBuilder builder = **new** StringBuilder();

**for**(**int** i = str.length() - 1; i >= 0; i--)

{

builder.append(str.charAt(i));

}

**if**(str.equals(builder.toString()))

System.***out***.println("The given string is palindrome");

**else**

System.***out***.println("The given string is not palindrome");

}

}

/\*

\* write a program to remove the duplicate characters from

\* a given string.

\* example:

\* String str = "missisipi";

\* output: misp

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a string");

String str = scan.nextLine();

String str2 = *removeDuplicates*(str);

System.***out***.println("After removing duplicate characters : " + str2);

}

**private** **static** String removeDuplicates(String str) {

str = str.toLowerCase();

StringBuilder sb = **new** StringBuilder();

**boolean** seen[] = **new** **boolean**[256];

**for**(**int** i = 0; i < str.length(); i++) {

**char** ch = str.charAt(i);

**if**(seen[ch] == **false**) {

sb.append(ch);

seen[ch] = **true**;

}

}

**return** sb.toString();

}

}

Home work-1:

write a program for string compression.

ex: String str = “aabbbcaaaddef”;

output: a2b3c1a3d2e1f1

Home work-2:

write a program to reverse each word of a given string.

ex: String str = “The cat sat”

output: ehT tac tas

/\*

\* write a program for string compression

\* For example

\* String str = "aabbbccdbc";

\* output: a2b3c2d1b1c1

\*/

**import** java.util.Scanner;

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a string");

String str = scan.nextLine();

String result = *compress*(str);

System.***out***.println("After compression : " + result);

scan.close();

}

**private** **static** String compress(String str) {

**int** count = 1;

StringBuilder builder = **new** StringBuilder();

**for**(**int** i = 0; i < str.length(); i++)

{

**if**( i + 1 < str.length() && str.charAt(i) == str.charAt(i+1))

{

count++;

}

**else**

{

builder.append(str.charAt(i)).append(count);

count = 1;

}

}

**return** builder.toString();

}

}

what is recursion?

-----------------

* Recursion means, calling a method from the body of the same method again.
* /\*
* \* write a program to find the factorial of a number
* \* with recursion.
* \*/
* **import** java.util.Scanner;
* **public** **class** Main {
* **public** **static** **void** main(String[] args) {
* // **TODO** Auto-generated method stub
* Scanner scan = **new** Scanner(System.***in***);
* System.***out***.println("Enter a number");
* **int** n = scan.nextInt();
* **if**( n < 0)
* {
* System.***out***.println("You must enter a number >= 0");
* System.*exit*(1);
* }
* **int** result = *factorial*(n);
* System.***out***.println("factorial value = " + result);
* }
* **private** **static** **int** factorial(**int** n) {
* // **TODO** Auto-generated method stub
* **if**(n == 0 || n == 1)
* **return** 1;
* **else**
* **return** n \* *factorial*(n-1);
* }
* }
* /\*
* \* write a program to print the fibonacci series of given
* \* number of terms with recursion.
* \*/
* **import** java.util.Scanner;
* **public** **class** Solution {
* **public** **static** **void** main(String[] args) {
* Scanner scan = **new** Scanner(System.***in***);
* System.***out***.println("Enter number of terms of fibonacci series");
* **int** terms = scan.nextInt();
* **for**(**int** i = 0; i < terms; i++)
* {
* **int** result = *fibonacci*(i);
* System.***out***.println(result);
* }
* scan.close();
* }
* **private** **static** **int** fibonacci(**int** n) {
* **if**( n == 0 ) **return** 0;
* **if**( n == 1 ) **return** 1;
* **else** **return** *fibonacci*(n - 1) + *fibonacci*(n - 2);
* }
* }

Object Oriented Programming System

( OOPS )

=================================

* To develop a software application, earlier we have POP model.
* POP – Procedure Oriented Programming
* In POP, a software application is developed by creating functions/procedures.
* POP model is good for developing small scale applications.
* To develop a large scale applications, more functions are required and the code becomes more complicated and also identifying and resolving the bugs is also difficult.
* So, Object Oriented Programming model was introduced.
* In OOP, we can map real-world entities to the objects in the applications. So, we can develop large scale applications easily with OOP model.
* With OOP model, to design and implement a software, a set of practices/principles were introduced, and they are called OOPS priciples.

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

Encapsulation: It means, combine/keep the data and its related functionalities at one place in a single unit and don’t allow others to modify the data directly.

This principle can be implemented in the software, by creating a class with variables and methods and with proper access modifiers.

Inheritance: It means, create a new class by inheriting the attributes and behavior from an existing class.

The newly created class is called as a child class and the exisiting class is called as a parent class.

With this principle, the advantages are,

1. reduces code duplication(redundency)
2. code reusability
3. improves productivity of a developer.

Note: productivity means, developing more code in less time.

Polymorphism:

. Polymorphism is a combination of 2 Greek words

“Poly” and “Morphism”.

. Poly means “many” and Morphism means “forms”.

. An operator or a method can perform a task/action

in an application.

. Polymorphism is an ability of an object to perform

an action in multiple forms.

. Polymorphism can be implemented using two

mechanisms called method overloading and method

overriding.

Abstraction:

. Abstraction is a process of providing the

essential information, by hiding the un-essential

information from the user of a system.

. This abstraction can be implemented using abstract

classes and interfaces.

====================================================

classes and objects:

------------------

. class is keyword taken from another word called

classification.

. In OOP terms, everything in the real-world is an object.

. Different groups of objects will have different attributes and

they perform different actions(different behaviour).

. So, we will define the attributes and the actions of a group

of objects into a template, before creating the objects. This

template is called as a class.

. A class is a template or a blueprint with common attributes

and behaviour for a group of objects.

. For example, In e-commerce application, we have employees and

customers as objects. The attributes and behaviour of

employees and customers are different. So, they are different

groups of objects.

. We create a template with attributes and behaviour for

employees and another template with attributes and behaviour

for customers. These templates are called classes.

syntax of creating a class:

--------------------------------

<access modifier> class <classname>

{

variables;

methods;

}

* access modifier is optional. If you don’t specify an access modifier, then by default it is considered as default.

example:

class User

{

String username;

String password;

void signup()

{

//logic

}

void signin()

{

//logic

}

}

* an object is an instance of a class.
* an object is the actual realization of a class.
* an object contains the values for the variables and it can perform the operations defined in the class.
* If two objects are created then they contain different values for the variables, but they can perform the same operations defined in the class.
* For example, if two objects are created for the above User class, then the objects can have different values for username and password, but they can perform same operations called signin() and signup().

syntax of creating an object:

----------------------------

classname objectname = new classname();

ex:

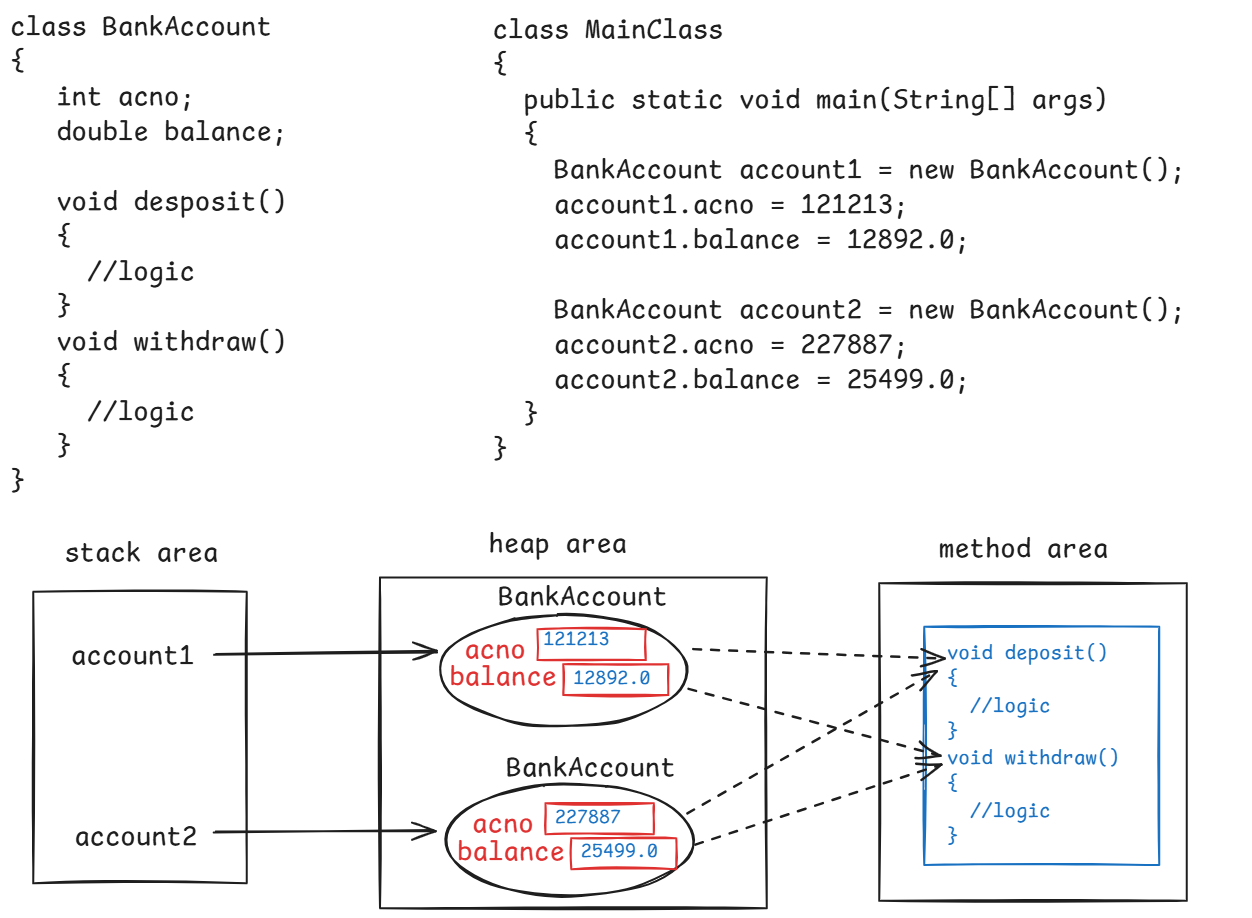
User user1 = new User();

User user2 = new User();

* Here, the object names are also called reference variables.
* The actual objects are created in the heap memory and these reference variables are created in the statck memory.
* When an object is created, the memory for the variables is allocated inside the object.
* Inside an object, the memory is allocated only for the variables, but not for the methods.
* The methods are stored in the method area and an object has links to the methods in the method area.

diagram:

-------



//example on class and object

**class** Example

{

**int** x;

**void** increment()

{

x++;

System.***out***.println("After increment : " + x);

}

**void** decrement()

{

x--;

System.***out***.println("After decrement : " + x);

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Example e1 = **new** Example();

e1.x = 10;

System.***out***.println("Initial value : " + e1.x);

e1.increment();

e1.increment();

e1.decrement();

}

}

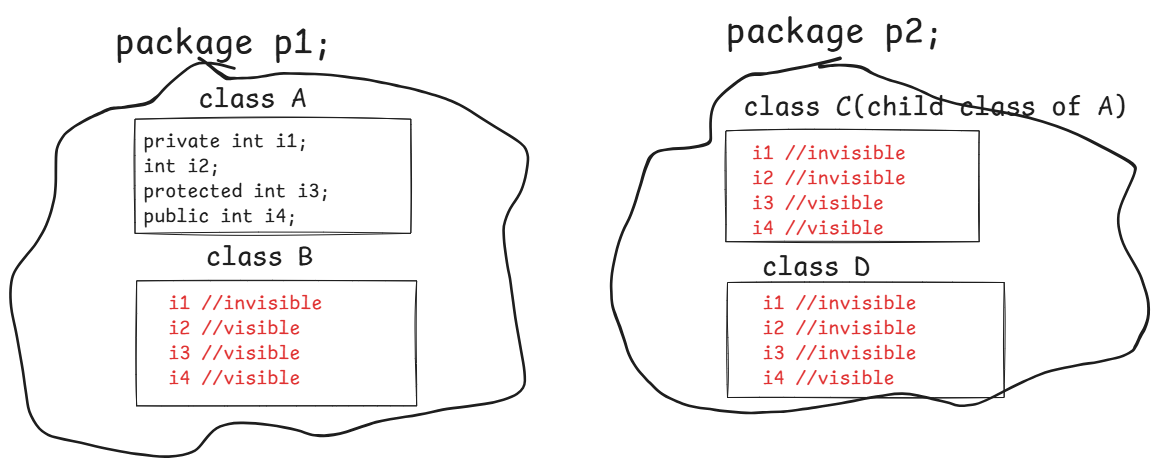
Access modifiers in Java:

------------------------

* The Access modifiers are used to specify the visibility for a class or for a variable or for a method or for a constructor.
* The access modifiers are,

1. private
2. default
3. protected
4. public

* If you add private access modifier to a variable or a method then it is visible within the same class only. At outside of this class, they are not visible.
* The important note is, we can not use the private access modifier for a class.
* If you don’t specify any access modifier for a class or a variable or a method then by default the access modifier is default.
* The default members are visible to the classes of the same package.
* If you add protected access modifier to a variable or a method then it is visible within the same package and also visible to the child classses in another package.
* The important note is, we can not use the protected access modifier also for a class.
* If you add public access modifier to a variable or a method or a class then it is visible to the classes in the same package and also visible to the classes of other packages.
* So, public access modifier will set global visibility for a member.



constructors :

* A constructor is used to initialize an object(assigning the values to the fields), when it is created.
* We can initialize the object manually without a constructor also. But it will increase the code and you may forget to initialize some fields.
* So, we use constructor to initialize an object.
* In a class, if you don’t explicitly define a constructor then JVM will define a default constructor automatically.
* The default constructor will assign the default values to the variables/fields, based on the datatype.
* A constructor is a special method in a class.
* It is a special method, because of the below reasons.

1. constructor name must be same as the classname.
2. constructor doesn’t contain return type
3. constructor is automatically executed by the JVM, when the object is created. So, you no need to call the constructor.

* constructors are 2 types.

1. parameter-less constructor / constructor without arguments
2. parameterized constructor / constructor with arguments.

* If you want to initialize multiple objects of a class with same values then you have to create parameter-less constructor.
* If you want to initialize different objects of a class with different values then you have to create parameterized constructor.

ex:

**class** Point

{

**private** **int** x;

**private** **int** y;

//parameter-less constructor

**public** Point()

{

**this**.x = 10;

**this**.y = 20;

}

//parameterized constructor

**public** Point(**int** x, **int** y)

{

**this**.x = x;

**this**.y = y;

}

**public** **void** display()

{

System.***out***.println("x : " + x);

System.***out***.println("y : " + y);

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Point p1 = **new** Point();

p1.display();

System.***out***.println("===================");

Point p2 = **new** Point(15, 45);

p2.display();

}

}

this keyword:

-------------

* this is a keyword in Java, it refers to the current object of a class.
* Suppose, we are creating a parameterized constructor, and the parameters names(local variables) are matching with the instance variables.
* if we assign the parameter value to the instance variable without this keyword, then the local variable value is again copied to the local variable only.
* So, we use this keyword to copy the local variable value to the instance variable.

ex:

**class** Employee

{

**private** **int** empno;

**private** String ename;

**public** Employee(**int** empno, String ename)

{

**this**.empno = empno;

**this**.ename = ename;

}

**public** **void** show()

{

System.***out***.println("empno : " + empno);

System.***out***.println("ename : " + ename);

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Employee e1 = **new** Employee(7101, "SMITH");

Employee e2 = **new** Employee(7102, "ALLEN");

e1.show();

System.***out***.println("================");

e2.show();

}

}

static keyword:

* static keyword represents that a member belongs to the class, not belongs to an object.
* using static keyword, we can create

1. static variables
2. static methods
3. static blocks
4. static inner classes

static variable:

. When you want to share the data of a variable across all instances of a class then you have to create a static variable.

. For a static variable, JVM creates the memory in the Method Area, and the objects of the class will share that variable.

. In a class, we can create non-static and static variables.

. non-static variables are also called instance variables.

. static variables can be called directly with the classname.

. if you call a static variable with the object name, then internally JVM calls it with the classname only.

example:

**class** Student {

**private** **int** sid;

**private** String sname;

**private** **static** String *collegeName*;

**public** Student(**int** sid, String sname)

{

**this**.sid = sid;

**this**.sname = sname;

Student.*collegeName* = "CBIT";

}

**public** **void** display() {

System.***out***.println("sid : " + sid );

System.***out***.println("sname : " + sname);

System.***out***.println("collegeName : " + *collegeName*);

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Student s1 = **new** Student(101, "Anil");

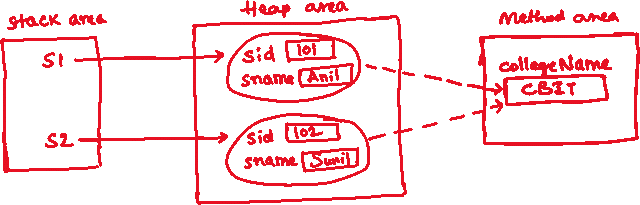
s1.display();

Student s2 = **new** Student(102, "Sunil");

s2.display();

}

}



Note: If you change the value of a static variable with one object, then the changes are effected to the other objects of that class also.

static method:

. When you want to create a method to perform an action, by without using the data of an object of the class, then you have to create that method as a static method.

. A static method performs an action for the given parameters.

* A static method can use only static variables of the class, but not the instance variables.
* A static method can be called directly with the classname. If you call a static method with objectname then internally JVM calls it with classname only.
* A non-static method can be called only with the objectname and it can use both instance variables and static variables.

example code:

**class** ClassA {

**private** **int** x;

**private** **static** **int** *y*;

**public** ClassA(**int** x)

{

**this**.x = x;

ClassA.*y* = 100;

}

**public** **static** **void** m1()

{

**int** z = *y* + *y*;

System.***out***.println(z);

}

**public** **void** m2()

{

**int** z = x + *y*;

System.***out***.println(z);

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

ClassA ca = **new** ClassA(20);

ClassA.*m1*();

ca.m2();

}

}

static block:

-------------

* Suppose, we have a static variable and we are initializing that variable in the constructor.
* For each object creation, the constructor is executed. So, the static variable is re-initialized for each and every object.
* To initialize the static variables for only once, we use static block.
* A static block will be executed for only once when the class is loaded into the JVM.
* Depends on requirement, we can also create multiple static blocks in a class.
* If multiple static blocks are created, then they are executed in the same order in which they have defined in the class.

example:

**class** ClassA {

**private** **int** x;

**private** **static** **int** *y*;

**public** ClassA(**int** x)

{

**this**.x = x;

System.***out***.println("I am constructor");

}

**static** {

*y* = 100;

System.***out***.println("I am static block1");

}

**static** {

System.***out***.println("I am static block2");

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

ClassA ca1 = **new** ClassA(10);

ClassA ca2 = **new** ClassA(20);

ClassA ca3 = **new** ClassA(30);

}

}

output:

I am static block1

I am static block2

I am constructor

I am constructor

I am constructor

Q) How many types of variables are there?

A) 3 types.

1. instance variables

2. static variables

3. local variables

Q) what is the difference between instance variable and static variable?

A) write your answer here

Inner classes:

* suppose, we are creating a class in an application, where only one other class is going to use it, not the remaining classes. Then instead of creating that class as a separate class, we can create it as inner class.
* For example, we are creating classes like Mobile, Laptop and Sim in our application. Here, Sim class is going to be used by Mobile class only. So, we can create Sim class as an inner class under Mobile class.
* If you create one class within the boundary of another class then it is called inner class.
* By creating inner classes, we can improve the code redability and we can get better encapsulation.

ex:

class Mobile {

//variables

//methods

class Sim {

//variables

//methods

}

}

key points:

1. private variables of the outer class are visible to the inner class. But private variables of inner class are not visible to the outer class.
2. If inner class also has same variable like outer class variable, then inner class will use the inner class variable only.
3. outer class can have default/public access modifier. inner class can have default/private/protected/public access modifier.
4. we can create inner class object in outer class.
5. inner classes are 4 types.

* non-static inner class
* static inner class
* local inner class
* anonymous inner class

1. non-static inner class can use static and non-static variables of the outer class. But static inner class can only use the static variables of the outer class.

Example code:

-------------

**class** Outer {

**private** **static** **int** *o1* = 100;

**void** outerMethod() {

System.***out***.println("In Outer :: outerMethod()");

System.***out***.println("o1 = " + *o1*); //prints outer class variable

}

**static** **class** Inner {

**private** **int** i1 = 150;

**private** **static** **int** *o1* = 112;

**void** innerMethod() {

System.***out***.println("In Inner :: innerMethod()");

System.***out***.println("i1 = " + i1);

System.***out***.println("o1 = " + *o1*); //prints inner class variable

System.***out***.println("o1 = " + Outer.*o1*); //prints outer class variable

}

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

//outer class object

Outer outer = **new** Outer();

//inner class object

Outer.Inner inner = **new** Outer.Inner();

outer.outerMethod();

inner.innerMethod();

}

}

setter and getter methods:

-------------------------

* setter method is used to update the value of a private variable
* getter method is used to fetch the value of a private variable
* suppose you have a private variable,

private double balance;

* Now, the setter and getter methods should be,

public void setBalance(double balance)

{

this.balance = balance;

}

public double getBalance()

{

return balance;

}

* suppose you have private variables,

private int accountNumber;

private String accountHolderName;

* Now, the setter and getter methods should be,

public void setAccountNumber(int accountNumber)

{

this.accountNumber = accountNumber;

}

public int getAccountNumber()

{

return accountNumber;

}

public void setAccountHolderName(String accountHolderName)

{

this.accountHolderName = accountHolderName;

}

public String getAccountHolderName()

{

return accountHolderName;

}

Example code:

**class** Employee

{

**private** **int** empno;

**private** String ename;

**private** **double** sal;

**public** **int** getEmpno() {

**return** empno;

}

**public** **void** setEmpno(**int** empno) {

**this**.empno = empno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **double** getSal() {

**return** sal;

}

**public** **void** setSal(**double** sal) {

**this**.sal = sal;

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Employee e1 = **new** Employee();

e1.setEmpno(7788);

e1.setEname("SCOTT");

e1.setSal(25000.0);

System.***out***.println("Empno = " + e1.getEmpno());

System.***out***.println("Ename = " + e1.getEname());

System.***out***.println("Sal = " + e1.getSal());

}

}

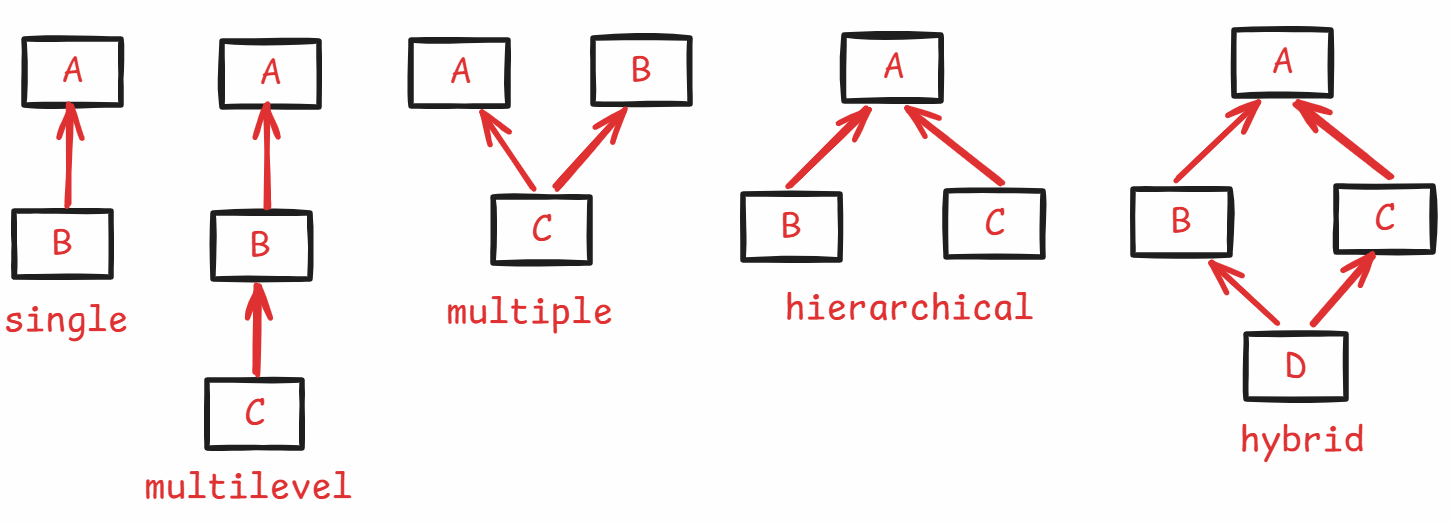
Inheritance

* Inheritance is a way of creating a new class by inheriting the attributes and functionalities from an existing class.
* The existing class is called parent class / super class / base class.
* The new class is called child class / sub class/derived class.
* Suppose, while designing the classes if we find any common attributes or common functionalities in multiple classes then we create a separate class for common things and we extend that class to create the new classes.
* For example, we need classes like FulltimeEmployee and ParttimeEmployee and we have identified that common attributes and common functionalities are available in both the classes. So, we can create a separate class like Employee for common things and we can extend that class into FulltimeEmployee and ParttimeEmployee classes.
* Here, Employee class is called as the Parent class and FulltimeEmployee, ParttimeEmployee classes are called as the child classes.
* With inheritance, we can improve code reusability and we can reduce code redundency.

Types of inheritance:

--------------------

1. single inheritance
2. multilevel inheritance
3. multiple inheritance
4. hierarchical inheritance
5. hybrid inheritance



* hybrid inheritance is a combination of hierarchical and multiple inheritance.
* multiple and hybrid inheritance is not possible with classes in Java. But possible with interfaces.
* child class inherits the behaviour from parent class and also it can add more behaviour to the objects. So we create objects for the child class.
* With parent class object, we can only call the parent class methods. But with child class object, we can call the parent class methods and also the child class methods.

constructors execution in inheritance:

------------------------------------

* In a constructor, the first statement is a super() call.
* If you don’t write super() call, then JVM will add this super() call automatically.
* The super() call, invokes the parameter-less constructor of the parent class.

ex:

**class** ClassA {

**public** ClassA()

{

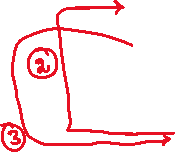


System.***out***.println("ClassA :: parameter-less constructor");



}

}



**class** ClassB **extends** ClassA

{

**public** ClassB()

{

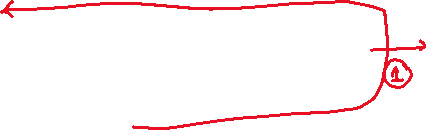


**super**();



System.***out***.println("ClassB :: parameter-less constructor");

}



}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

ClassB cb = **new** ClassB();

}

}

output:

ClassA :: parameter-less constructor

ClassB :: parameter-less constructor

* If the parent class doesn’t have parameter-less constructor, but it has parameterized constructor then super() call gets error.
* To solve this error, we need to call parameterized constructor of the parent class with super() call, by passing parameters.

ex:

**class** ClassA {

**public** ClassA(**int** x, **int** y)

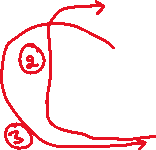
{

System.***out***.println("ClassA :: parameterized constructor");



}

}



**class** ClassB **extends** ClassA

{

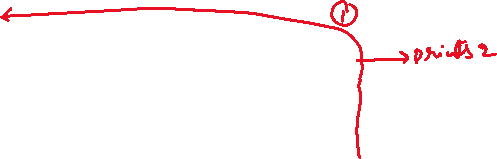
**public** ClassB()

{

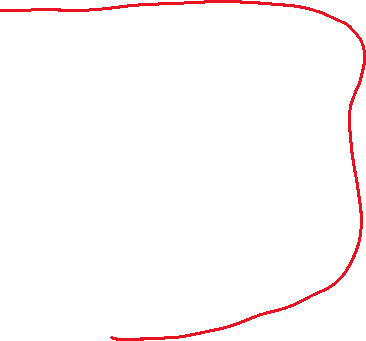
**super**(10, 20);



System.***out***.println("ClassB :: parameter-less constructor");



}



}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

ClassB cb = **new** ClassB();

}

}

output:

ClassA :: parameterized constructor

ClassB :: parameter-less constructor

==============================================

ex:

**class** ClassA {

**private** **int** x;

**private** **int** y;

**public** ClassA(**int** x, **int** y)

{

System.***out***.println("ClassA :: parameterized constructor");

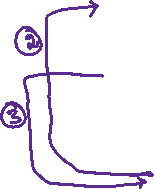


**this**.x = x;

**this**.y = y;

}

}



**class** ClassB **extends** ClassA

{

**private** **int** z;

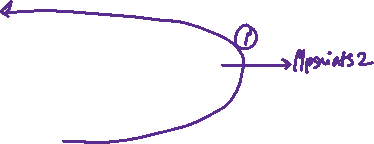
**public** ClassB(**int** x, **int** y, **int** z)

{

**super**(x, y);

**this**.z = z;

System.***out***.println("ClassB :: parameterized constructor");



}

}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

ClassB cb = **new** ClassB(10, 20, 30);

}

}

output:

ClassA :: parameterized constructor

ClassB :: parameterized constructor

this() call:

----------

* this() call is used to call the same class constructor.
* this() call also must be the first statement in a constructor.
* if you add this() call then JVM doesn’t add the super() call.
* we can’t add this() call and super() call at a time.

ex:

**class** ClassA {

**public** ClassA()

{

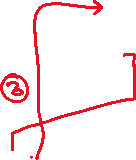


System.***out***.println("ClassA :: parameter-less constructor");



}

}



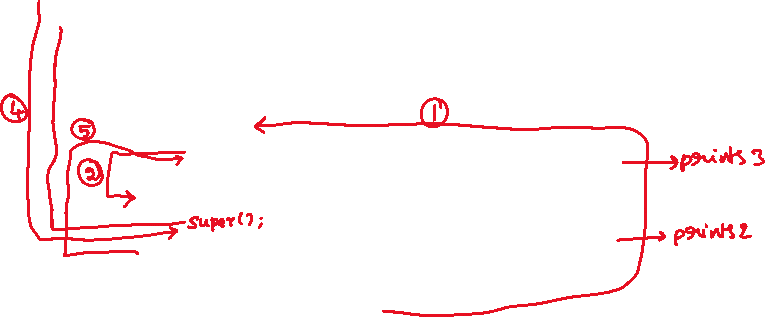
**class** ClassB **extends** ClassA

{

**public** ClassB()

{

**this**(10, 20);



System.***out***.println("ClassB :: parameter-less constructor");

}

**public** ClassB(**int** x, **int** y)

{

System.***out***.println("ClassB :: parameterized constructor");

}

}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

ClassB cb = **new** ClassB();

}

}

output:

ClassA :: parameter-less constructor

ClassB :: parameterized constructor

ClassB :: parameter-less constructor

Q) will static method of a parent class is inherited to the child class?

A) Yes.

Q) can we call instance method directly in a static method?

A) No.

Q) can we call static method directly in instance method?

A) Yes.

Q) can we call static method directly in another static method of the class?

A) Yes.

Q) why Java doesn’t support multiple inheritance with classes?

A) When child class object is created, JVM has to call parent class constructor because of super() call. But in multiple inheritance, there are multiple parent classes for a child class. So, JVM will get ambiguity error, hence multiple inheritance is not supported with classes.

Q) what is the difference between this keyword and this() call?

A) 1. this keyword refers to the current object of a class and it is used to call the variables or the methods of the current object.

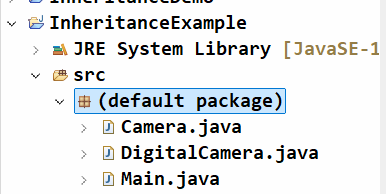
this() call is used to call the another constructor of the same class and it must be the first statement in a constructor.

2. this keyword can be used in constructor and in other methods also. But this() call can be used only in the constructor.

Q) what is the difference between super keyword and super() call?

A) write your answer.

//Inheritance example



Camera.java

//parent class

**public** **class** Camera {

**private** String brand;

**private** String lensType;

**public** Camera(String brand, String lensType) {

**super**();

**this**.brand = brand;

**this**.lensType = lensType;

}

**public** String getBrand() {

**return** brand;

}

**public** **void** setBrand(String brand) {

**this**.brand = brand;

}

**public** String getLensType() {

**return** lensType;

}

**public** **void** setLensType(String lensType) {

**this**.lensType = lensType;

}

**public** **void** capture() {

System.***out***.println(brand + " camera with lens : " + lensType + " captured a photo...");

}

}

DigitalCamera.java

//child class

**public** **class** DigitalCamera **extends** Camera {

**private** **int** resolution;

**private** **int** storage;

**public** DigitalCamera(String brand, String lensType, **int** resolution, **int** storage) {

**super**(brand, lensType);

**this**.resolution = resolution;

**this**.storage = storage;

}

**public** **void** recordVideo()

{

System.***out***.println(getBrand() + " camera with lens : " + getLensType() +

" with resolution : " + resolution + "MP with storage " + storage + "GB, recording video");

}

}

Main.java

**public** **class** Main {

**public** **static** **void** main(String[] args) {

DigitalCamera digitalCamera = **new** DigitalCamera("Nikon", "Wide-Angle", 64, 512);

digitalCamera.capture();

digitalCamera.recordVideo();

}

}

Polymorphism

==================

* Poly means “many” and Morphos means “forms”. So, polymorphism indicates “many forms”.
* In OOP, polymorphism means, a method should be used for different types of data or a method should be used for different actions.
* To use a same method to perform an action for different types of data, you have to overload that method.
* To use a same method to perform different actions, you have to override that method.
* Polymorphism is of 2 types.

1. compile-time polymorphism/static polymorphism.
2. runtime polymorphism/dynamic polymorphism.

* compile-time polymorphism can be achieved with method overloading and runtime polymorphism can be achieved with method overriding.

method overloading:

* Method overloading means, writing the same method in a class for more than once with different parameters.
* The difference must be in,

either number of parameters,

or types of parameters,

or sequence of parameters.

ex1:

class RechargeService

{

void recharge(long mobile, double amoount)

{

//logic

}

void recharge(long mobile, double amount, String coupon)

{

//logic

}

}

* In this example, our class has 2 recharge methods, but the number of parameters are different. So, recharge method is overloaded.

ex:

class DeliveryService {

void deliver(int pincode)

{

//logic

}

void deliver(String city)

{

//logic

}

}

* In this example, our class has 2 deliver methods with single parameter, but the types of parameter is different. So, deliver method is overloaded.

ex:

class ClassA {

void m1(int x, double y)

{

//logic

}

void m1(double x, int y)

{

//logic

}

}

* In this example, our class has 2 m1 methods with 2 parameters. But the sequence of parameters is different. So, m1 method is overloaded.

ex:

class EmployeeService {

int findById(int empid)

{

//logic

}

void findById(int empid)

{

//logic

}

}

* In this example, our class has 2 findById methods with single parameter. But both parameters are same type, there is no difference in parameters.
* The return types are different.
* It is not method overloading and we will get compile-time error. Because, for method overloading, return types of the methods is not important.

method overriding:

----------------

* If a method which is provided by the parent class is not matching with the child class requirements, then the child class override that method.
* overriding the method means, redefining the method again.
* For overriding a method, we need a parent class and a child class. It means, inheritance is required.
* Method overriding rules are,

1. method name must be same
2. the number of parameters, datatype of the parameters, and the sequence/order of the parameters must be same.
3. return type must be same as parent class method, or it can be co-variant type.
4. access modifier must be either same as parent class method, or it could be higher.

ex:

**class** ParentClass {

**public** **void** m1(**int** x) {

System.***out***.println("ParentClass :: m1()");

}

}

**class** ChildClass **extends** ParentClass {

@Override

**public** **void** m1(**int** k) {

System.***out***.println("ChildClass :: m1()");

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

ChildClass cc = **new** ChildClass();

cc.m1(10);

}

}

output:

ChildClass :: m1()

ex:

**class** ParentClass {

**public** **void** m1(**int** x) {

System.***out***.println("ParentClass :: m1()");

}

}

**class** ChildClass **extends** ParentClass {

@Override

**void** m1(**int** k) {



System.***out***.println("ChildClass :: m1()");

}

}

ex:

**class** ParentClass {

**public** **void** m1(**int** x) {



System.***out***.println("ParentClass :: m1()");

}

}

**class** ChildClass **extends** ParentClass {

@Override

**public** **int** m1(**int** k) {



System.***out***.println("ChildClass :: m1()");

return 1;

}

}

ex:

**class** ParentClass {



**public** **void** m1(**int** x) {

System.***out***.println("ParentClass :: m1()");

}

}

**class** ChildClass **extends** ParentClass {

@Override

**public** **void** m1(**int** k, int v) {



System.***out***.println("ChildClass :: m1()");

}

}

ex:

**class** ParentClass {

**public** **Employee find()** {

System.***out***.println("ParentClass :: find()");

}

}

**class** ChildClass **extends** ParentClass {

@Override

**public** **FullTimeEmployee** find() {

System.***out***.println("ChildClass :: find()");

}

}

* Here, assume that Employee class is parent and FullTimeEmployee class is child. So, we can say that a child class as a co-variant type.
* So, the above method overriding is correct.

ex:

**class** ParentClass {

**public** FullTimeE**mployee find()** {

System.***out***.println("ParentClass :: find()");

}

}

**class** ChildClass **extends** ParentClass {

@Override

**public** **PartTimeEmployee** find() {



System.***out***.println("ChildClass :: find()");

}

}

Q) can we override a private method?

A) No

Q) can we override a static method?

A) No

Q) can we override a final method?

A) No.

Q) can we overload a private method?

A) Yes

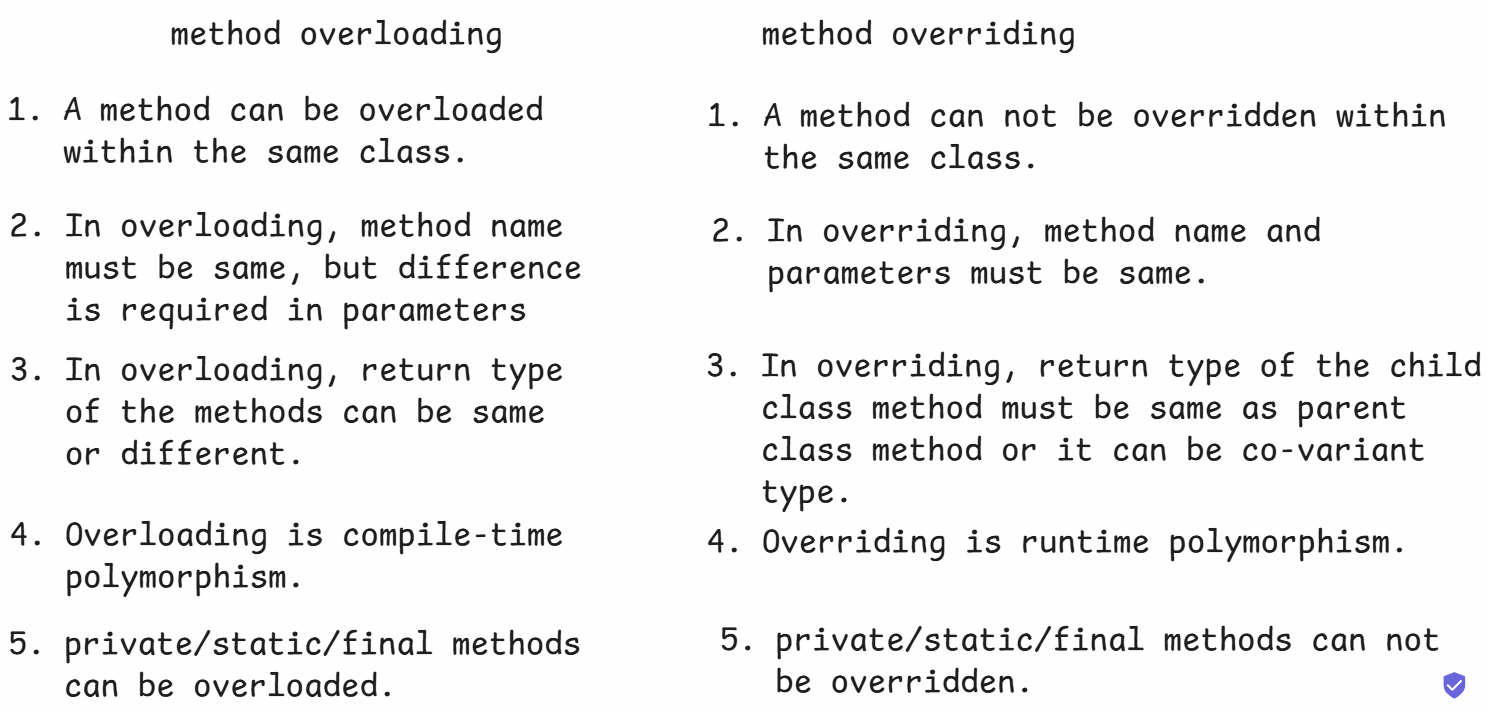
Q) can we overload a static method?

A) Yes

Q) can we overload a final method?

A) Yes

Q) what is the difference between method overloading and method overriding?

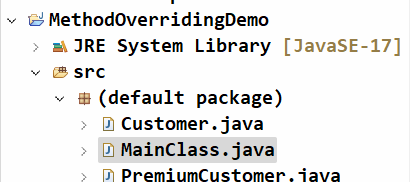


Q) what is the difference between compile-time polymorphism and runtime polymorphism?

A) In Compile-time polymorphism, the Java compiler creates the bindings between the method calling point and method definition point at the compile-time only. (overloading)

In Runtime polymorphism, the bindings between the method calling point and method definition point are created by the JVM at runtime, based on the actual object type. (overriding).

//Method overriding example:



Customer.java

------------

//parent class

//Create Customer class with attributes item, qty and price

**public** **class** Customer {

**private** String item;

**private** **int** qty;

**private** **double** price;

**public** Customer(String item, **int** qty, **double** price) {

**super**();

**this**.item = item;

**this**.qty = qty;

**this**.price = price;

}

**public** String getItem() {

**return** item;

}

**public** **void** setItem(String item) {

**this**.item = item;

}

**public** **int** getQty() {

**return** qty;

}

**public** **void** setQty(**int** qty) {

**this**.qty = qty;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

**public** **void** calculateBill() {

**double** amount = price \* qty;

System.***out***.println("item : " + item);

System.***out***.println("qty : " + qty);

System.***out***.println("amount : " + amount);

}

}

PremiumCustomer.java

--------------------

//child class

**public** **class** PremiumCustomer **extends** Customer {

**private** **int** reedimPoints;

**public** PremiumCustomer(String item, **int** qty, **double** price, **int** reedimPoints) {

**super**(item, qty, price);

**this**.reedimPoints = reedimPoints;

}

**public** **int** getReedimPoints() {

**return** reedimPoints;

}

**public** **void** setReedimPoints(**int** reedimPoints) {

**this**.reedimPoints = reedimPoints;

}

@Override

**public** **void** calculateBill() {

**double** amount = getPrice() \* getQty() - reedimPoints \* 100;

System.***out***.println("item : " + getItem());

System.***out***.println("qty : " + getQty());

System.***out***.println("amount : " + amount);

}

}

MainClass.java

-------------

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

//parent class object

Customer customer = **new** Customer("Mobile", 2, 29999.0);

customer.calculateBill();

System.***out***.println("\*+++++++++++++++++++++++++++\*");

//child class object

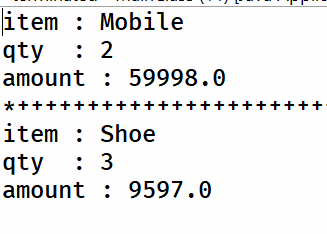
PremiumCustomer premiumCustomer = **new** PremiumCustomer("Shoe", 3, 5999.0, 84);

premiumCustomer.calculateBill();

}

}

output:



final keyword in Java:

--------------------

* final modifier can be used with variables, methods and classes with a distinct purpose.

final variables:

--------------

* To define the constant variables in Java, we use final variables.
* A constant variable means, once a value is assigned to the variable, it can not be changed.
* final variables are 3 types.

1. instance final variable
2. static final variable
3. local final variable

1.instance final variable:

\* instance final variable is a constant variable at object-

level.

\* It means, you can’t modify the value of that final

variable within that object.

\* For an instance final variable, you can assign the value

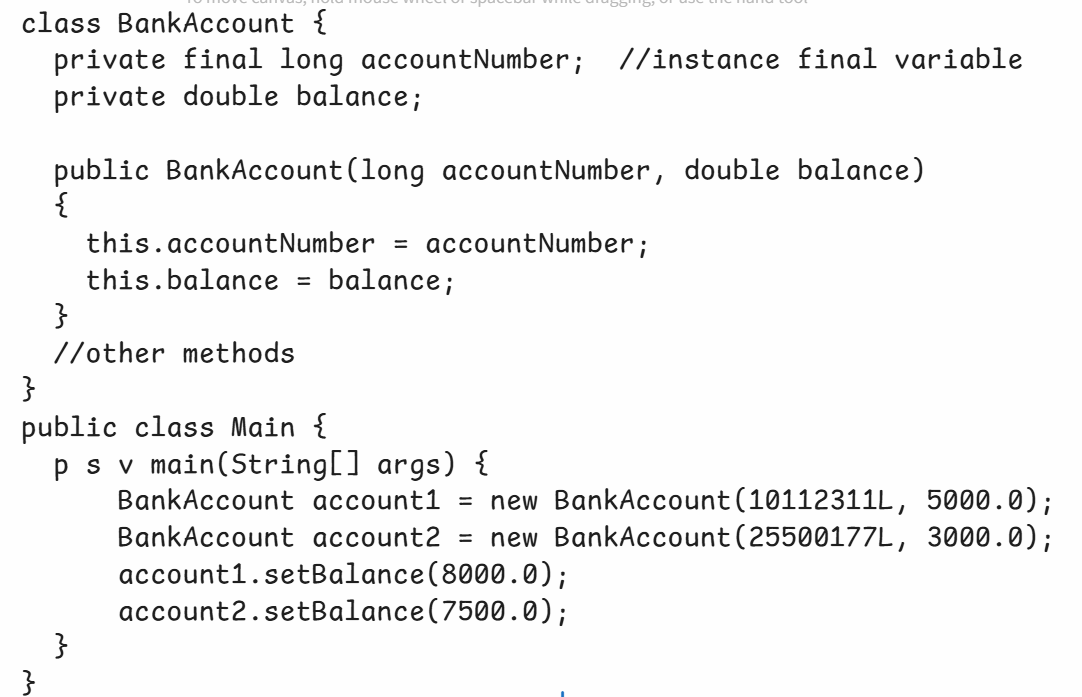
at the time of variable declaration or within the

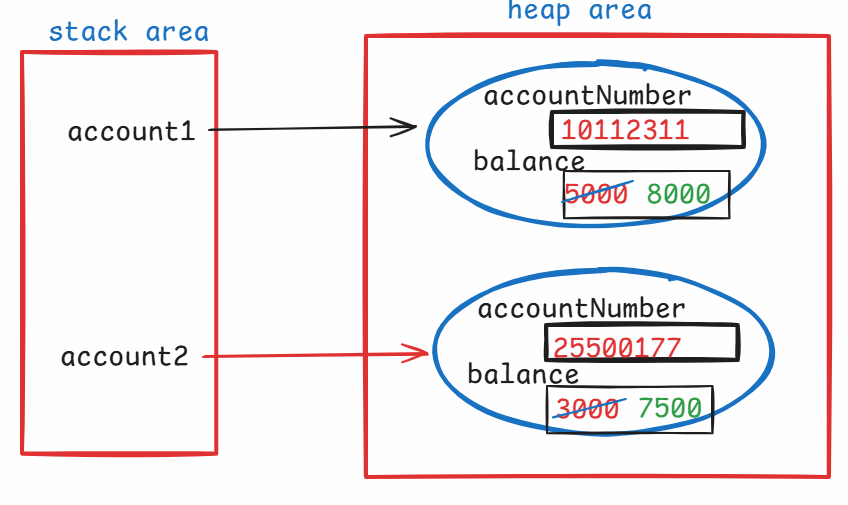
constructor.

\* For a final variable, you can’t define a setter method, but

you can define a getter method.

ex:





2. static final variable:

---------------------

\* static final variable is a class-level constant.

\* The value of a static final variable is constant for all the

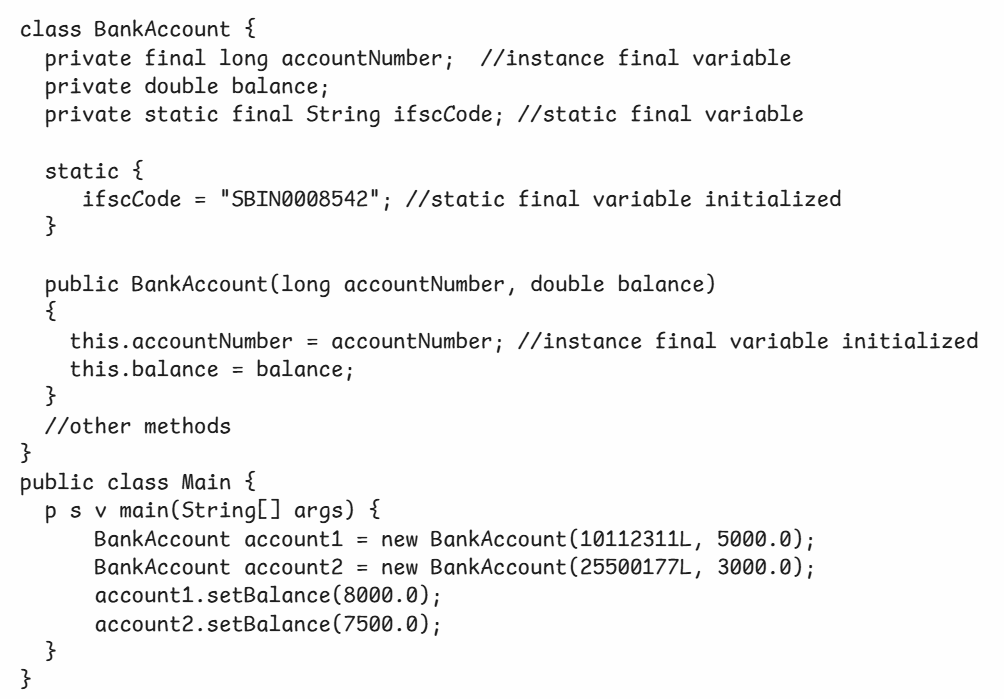
objects of a class.

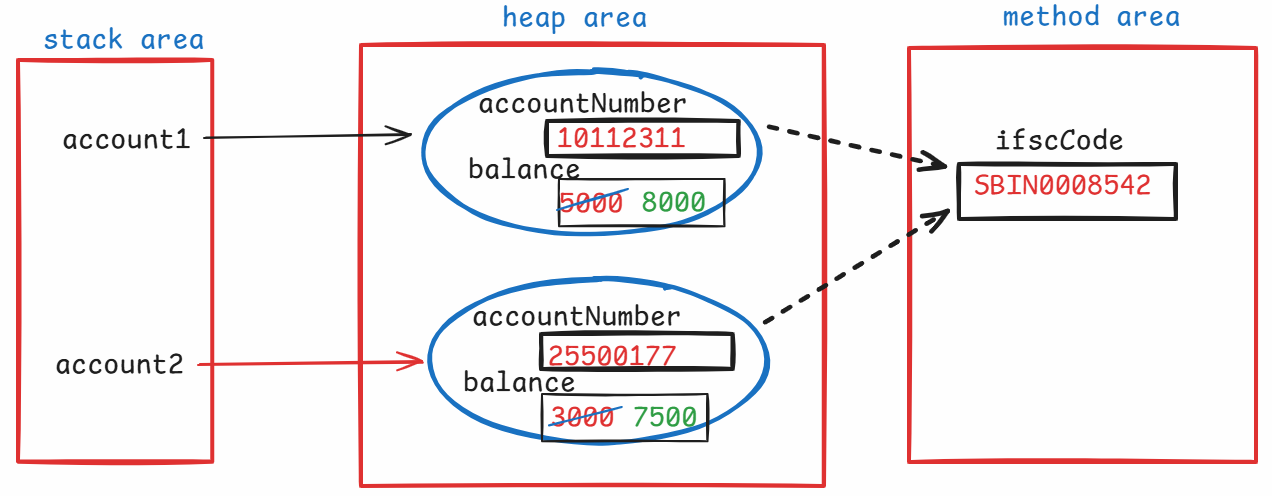
\* For a static final variable, you can assign the value

directly while declaring the variable or in the static

block. But you can’t assign the value in the constructor.

EX:





3.local final variable:

-------------------

\* If you declare a local variable in a method as final, then

it becomes local final variable.

\* a local final variable is a method-level final variable.

\* The variable is visible within that method only and we

can’t modify the value in that method.

\* The memory is allocated to the local variable in the stack

area, when the method execution starts and it is removed

from the stack are, when the method execution is completed.

ex:

class ClassA {

void m1() {

final int k = 80; //local final variable

}

}

final method:

-----------

* If you want to restrict the child classes from overriding a parent class method, then you should declare the parent class method as final.
* In Java, private methods, static methods and final methods of a parent class can’t be overridden in the child class.
* we can make a method as private and final. But technically both are same. Similarly, static and final also.
* A final method can be overloaded. But can’t be overridden.

ex1:

public class ClassA {

final void m1() {

//logic

}

final void m1(int x) { //correct, overloaded

//logic

}

}

ex2:

class ClassA {

final void m1() {

//logic

}

}

class ClassB extends ClassA {

@Override

final void m1() { //error

//logic

}

}

final class:

----------

* Suppose, if you don’t want to allow creating a child class for a class, then you should declare your class as final class.
* For example, String, Scanner, System, etc.. are the final classes, so we can’t create the child classes.

ex:

final class A {

//methods

}

class B extends A { //error.

//methods

}

Q) can we create a final constructor?

A) No

Q) can we create a static constructor?

A) No

Q) what is the difference between instance final variable and static final variable?

A) 1. instance final variable is an object level constant.

static final variable is a class level constant.

2. instance final variable gets memory in the heap area.

static final variable gets memory in the method area.

3. instance final variable can be initialized in a constructor.

static final variable can be initialized in a static block.

Q) can we create an object for a final class?

A) Yes.

Q) can we override the methods of a final class?

A) No. Because, a final class can’t be extended into a child class. So, we can’t override the methods of the final class.

Q) can we create a final object?

A) Yes. But we can’t assign another object of the class to the same reference variable.

ex:

**class** OuterClass {

**int** x;

**public** OuterClass() {

x = 10;

}

}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

**final** OuterClass oc = **new** OuterClass();

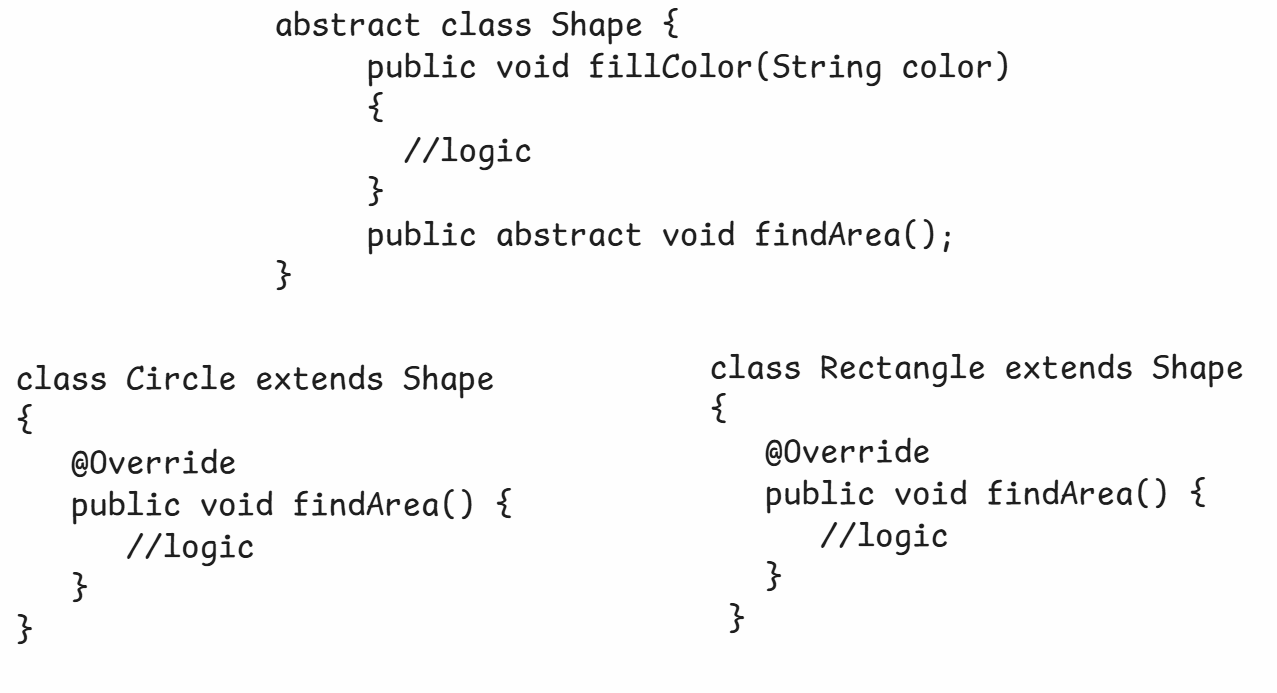
oc = **new** OuterClass(); //error

}

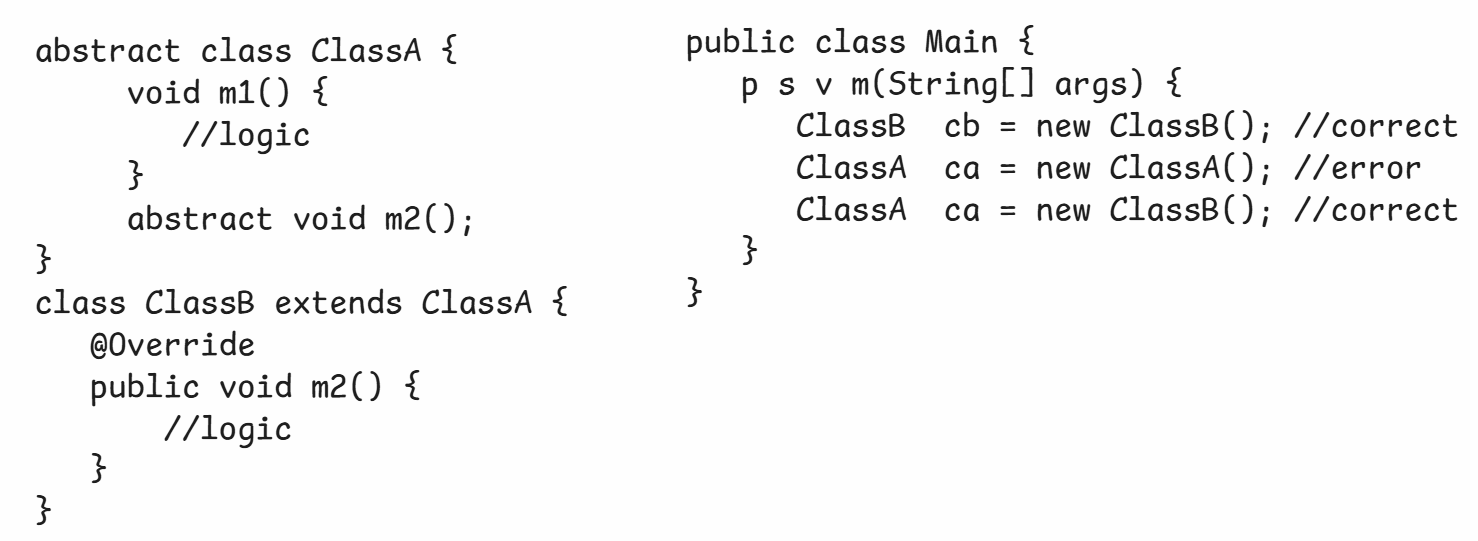
}

Abstract classes

* while designing multiple classes, if you find any common attributes or common functionalities then you have to separate them into another class for code reusability.
* From that class, you have to inherit the other classes. So, we call the common class as parent class and inherited classes as child classes.
* For a common functionality, if the child classes have to implement the logic in their own way, then we have to declare that method as abstract method in the parent class.
* abstract method means, the method should contain only signature/header in the class.
* If you declare any abstract method in a class, then you have to declare that class also as an abstract class.
* The abstract method of the parent should be overridden in the child classes.



* Suppose, if a parent class has 2 abstract methods then child class has to override the 2 abstract methods.
* If a child class is not overriding the 2 abstract methods or if it is overriding one abstract method then we have to declare that child class also as abstract.
* with abstract modifier, we can create abstract methods and abstract classes. But we can’t create any variables or constructors or blocks with the abstract keyword.
* For an abstract class, we can not create an object.
* We can store the child class object in the abstract class reference variable.



Q) can we create private abstract methods?

A) No.

Q) can we create final abstract methods?

A) No.

Q) can we create abstract final class?

A) No.

Q) can we create instance variables/static variables in abstract class?

A) Yes.

Q) can we create an instance for an abstract class?

A) No.

Q) can we define a constructor in abstract class?

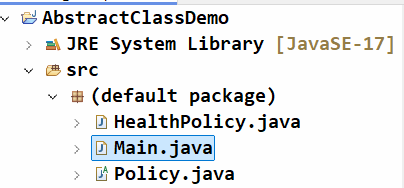
A) Yes. When child class object is created, the child class constructor will call the parent class constructor with super() call.

Q) when child class object is created, will the parent class object is also created internally or not?

A) No. But, as part of the child class object only, the memory is allocated for the parent class variables, by the JVM.

Q) can we make a class as abstract, without abstract methods?

A) Yes. Suppose, if some methods of a class has partial logic or a dummy logic then you can make that class as abstract class.





Policy.java

-----------

**public** **abstract** **class** Policy {

**private** **int** policyId;

**private** **double** premiumAmount;

**public** Policy(**int** policyId, **double** premiumAmount) {

**this**.policyId = policyId;

**this**.premiumAmount = premiumAmount;

}

**public** **abstract** **double** calculateRiskFactor();

**public** **void** calculateFinalAmount() {

**double** riskFactor = calculateRiskFactor();

**double** finalAmount = premiumAmount + riskFactor;

System.***out***.println("Base premium = " + premiumAmount);

System.***out***.println("Risk factor = " + riskFactor);

System.***out***.println("Final amount = " + finalAmount);

}

**public** **int** getPolicyId() {

**return** policyId;

}

**public** **void** setPolicyId(**int** policyId) {

**this**.policyId = policyId;

}

**public** **double** getPremiumAmount() {

**return** premiumAmount;

}

**public** **void** setPremiumAmount(**double** premiumAmount) {

**this**.premiumAmount = premiumAmount;

}

}

HealthPolicy.java

----------------

**public** **class** HealthPolicy **extends** Policy {

**private** **boolean** isSmoker;

**private** **int** age;

**public** HealthPolicy(**int** policyId, **double** premiumAmount, **boolean** isSmoker, **int** age) {

**super**(policyId, premiumAmount);

**this**.isSmoker = isSmoker;

**this**.age = age;

}

@Override

**public** **double** calculateRiskFactor() {

**double** riskFactor = 0;

**if**(age >= 60 && isSmoker == **true**)

{

riskFactor = getPremiumAmount() \* 0.60;

}

**else** **if**(age < 60 && isSmoker == **true**)

{

riskFactor = getPremiumAmount() \* 0.30;

}

**else**

{

riskFactor = getPremiumAmount() \* 0.10;

}

**return** riskFactor;

}

}

Main.java

--------

**public** **class** Main {

**public** **static** **void** main(String[] args) {

HealthPolicy hp = **new** HealthPolicy(2009001, 35000, **true**, 69);

hp.calculateFinalAmount();

System.***out***.println("=======================");

HealthPolicy hp2 = **new** HealthPolicy(188990, 24000, **false**, 30);

hp2.calculateFinalAmount();

}

}

output:

Base premium = 35000.0

Risk factor = 21000.0

Final amount = 56000.0

=======================

Base premium = 24000.0

Risk factor = 2400.0

Final amount = 26400.0

Interfaces

---------------------------

* Suppose, we want to create multiple classes with same methods, and with their own implementations in a project.
* To ensure that the classes should contain same methods, we have to define a contract for the classes.
* This contract is an interface.
* An interface can contain public static final variables and public abstract methods.
* A class has to implement an interface and it has to override all the abstract methods of that interface.

example:

interface MyInter {

void m1();

void m2();

}

class MyClass1 implements MyInter {

@Override

public void m1() {

//logic

}

@Override

public void m2() {

//logic

}

}

class MyClass2 implements MyInter {

@Override

public void m1() {

//logic

}

@Override

public void m2() {

//logic

}

}

ex:

public interface CustomerInter {

int MAX\_CUSTOMERS = 100;

void save();

void find();

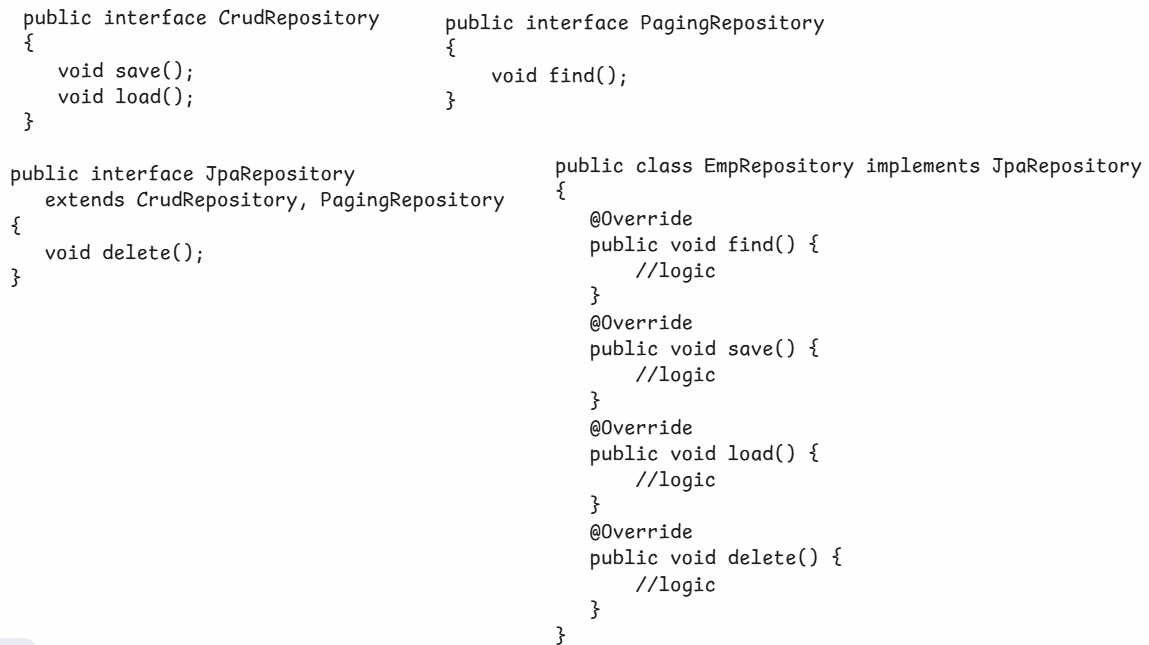
}

* In an interface, by default the variables are public static and final. So, it is optional to write these keywords.
* In an interface, by default the methods are public and abstract. So, it is optional to write these keywords.

Note:

. As per coding standards, if a variable is public static and final then we declare that variable in UpperCase.

* we can create a child interface by extending one or more parent interfaces also. So, we can achieve multiple inheritance with interfaces.



* In Java, one class can implement multiple interfaces.
* If two interfaces have the same abstract method, then the class has to override that abstract method for only once.

ex:

interface MyInter1 { interface MyInter2 {

void m1(); void m1();

} }

class MyClass implements MyInter1, MyInter2 {

@Override

public void m1() {

//logic

}

}

Q1) Do we have constructor in interface?

1. No.

Q2) Do we have constructor in class?

1. Yes.

Q3) Can we create an object for an interface?

1. No.

Q4) Can we create an object for an abstract class?

1. No.

Q4) Who is the default parent for interface?

1. Nothing.

Q5) Who is the default parent for a class?

1. java.lang.Object class

Q6) what is the difference between abstract class and interface?

1. write your answer here.

Types of interfaces:

------------------

1. Normal interface
2. Marker interface
3. Functional interface(From Java8)

* Normal interface means, an interface with more than one abstract method.
* Marker interface means, an interface without anything. It is an empty interface.
* Functional interface means, an interface with a single abstract method.
* For example, java.io.Serializable, java.lang.Cloneable and java.rmi.Remote are the pre-defined marker interfaces.

Q) what is the use of marker interfaces?

A) marker interfaces are used to mark the objects of a specific class to get additional behavior.

. For pre-defined marker interfaces, JVM will provide the additional behaviour for the objects.

. For example, if a class implements java.io.Serializable marker interface, then JVM will allow them to transfer over the network.

. For user-defined marker interfaces, the programmer has to define the additional behaviour.

Q) Can we create user-defined interface?

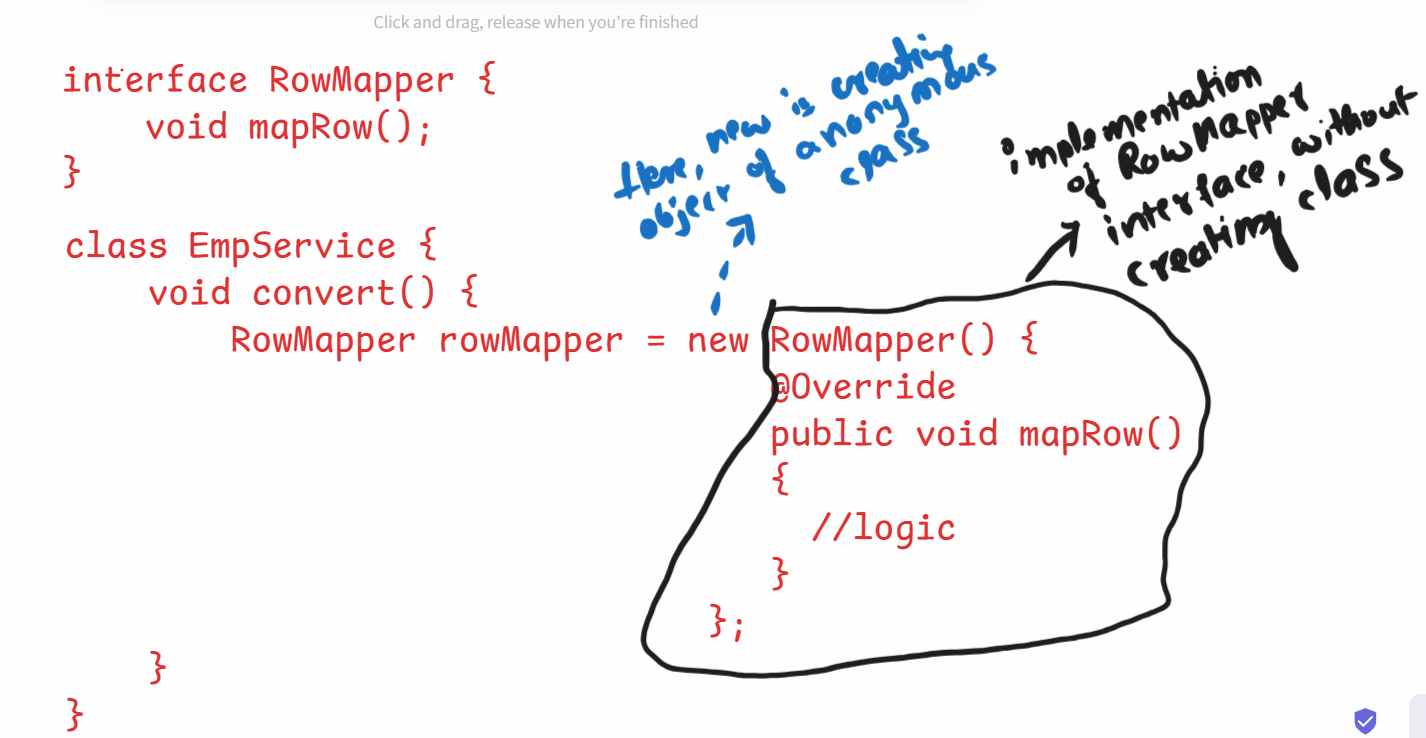
A) Yes.

Q) How many implementation classes can be created for an interface?

A) we can create one or more implementation classes.

Q) what is anonymous inner class?

A) You can provide implementation for an interface, by without creating a class. It is called anonymous inner class.



Q) What is IS-A relationship and HAS-A relationship?

A) 1. Is-A relationship is called Inheritance

Has-A relationship is called Association.

2. If you a create a class by extending another class, then it

is IS-A relationship.

3. If you create object one class in another class, then it is

HAS-A relationship.

class B extends A //IS-A class X {

{ Y y = new Y(); //HAS-A

} }

4. Association(HAS-A) is of 2 types, aggregation and composition.

5. aggregation is weak binding between the classes and composition is strong binding between the classes.

Exception Handling

=====================================

* A program can have,

1. compile-time errors
2. runtime errors
3. logical errors

* Runtime errors are called exceptions.
* Exceptions will occur in a program, because of the user’s input.
* The default behaviour of an application is, when an exception occurs, the application execution will be terminated abnormally.
* Exception handling is a mechanism, which makes an application to execute without termination even though exception occurs.
* To implement exception handling, we have to use the below keywords.

1. try
2. catch
3. finally
4. throw
5. throws

try, catch and finally:

* try, catch and finally are the keywords used to define the blocks.
* try block should contain the statements, from which we are expecting an exception.
* catch block should contain the statements, that should handle the exception.
* finally block should contain the statements, the must execute irrespective of the exception.
* we can write try, catch and finally blocks inside a method.

void m1() {

try {

statements;

}

catch(Exception ex)

{

statements;

}

finally {

statements;

}

}

Q) can we write try and catch blocks without finally block?

A) Yes.



Q) can we write try and finally blocks without catch block?

A) Yes.



Q) can we write the blocks in try, finally then catch order?

A) No. We can write only in try, catch then finally order.

Q) can we write multiple catch blocks for a try block?

A) Yes. Writing catch blocks for specific exceptions followed by catch block for generic exception(Exception class) is valid. But, writing catch block for generic exception(Exception class) followed by catch blocks for specific exceptions is invalid.

example1:

**void** someMethod(**int** a, **int** b)

{

**int**[] arr = **new** **int**[5];

String str = **null**;

**try** {

**int** c = a / b;

System.***out***.println("c = " + c);

arr[4] = 50;

System.***out***.println(str.length());

}

**catch**(Exception ex) { //catch for generic exception

System.***out***.println("3");

System.***out***.println(ex);

}

**catch**(ArithmeticException ex) { //compile-time error

System.***out***.println("1"); //(unreachable code)

System.***out***.println(ex);

}

**catch**(ArrayIndexOutOfBoundsException ex) { //error

System.***out***.println("2");

System.***out***.println(ex);

}

**finally** {

System.***out***.println("I am in finally...");

}

}

* The above code has catch for generic exception followed catch for specific exceptions. So, the code has errors.

example2:

**void** someMethod(**int** a, **int** b)

{

**int**[] arr = **new** **int**[5];

String str = **null**;

**try** {

**int** c = a / b;

System.***out***.println("c = " + c);

arr[4] = 50;

System.***out***.println(str.length());

}

**catch**(ArithmeticException ex) { //specific catch

System.***out***.println("1");

System.***out***.println(ex);

}

**catch**(ArrayIndexOutOfBoundsException ex) { //specific catch

System.***out***.println("2");

System.***out***.println(ex);

}

**catch**(Exception ex) { //generic catch

System.***out***.println("3");

System.***out***.println(ex);

}

**finally** {

System.***out***.println("I am in finally...");

}

}

* The above example has catch blocks for specific exceptions, followed by catch for generic exceptions. So, it is valid.

Q) can we write multiple finally blocks for a try block?

A) No.

Q) can we write a try block in another try block?

A) Yes. It is called nested try blocks.

Q) If an exception occurs in the inner try block, who can handle it?

A) inner catch block can handle it. If it is unable to handle it then the control goes to the outer catch block.

Q) If an exception occurs in the outer try block, who can hande it?

A) outer catch block can handle it.

Q) When to write multiple catch blocks for a try block?

A) If you want to handle/deal the different exceptions in different ways, then you have to write multiple catch blocks.

Q) what are the different ways to display the exception details?

A) In 3 ways we can display.

1. by calling getMessage() method

2. print the excepton object

3. by calling printStackTrace()

. getMessage() method returns the exception message string.

. when you print the exception object, it will display

exception classname and exception message string.

. when you call printStackTrace() method, it will display stack trace of an exception. This stack trace contains 4 details.

1. exception classname 2. exception message string

3.methods call stack 4. location of the exception

example:

**void** someMethod(**int** a, **int** b)

{

**int**[] arr = **new** **int**[5];

String str = **null**;

**try** {

**int** c = a / b;

System.***out***.println("c = " + c);

arr[4] = 50;

System.***out***.println(str.length());

}

**catch**(Exception ex) {

System.***out***.println(ex.getMessage());

System.***out***.println("=================================");

System.***out***.println(ex);

System.***out***.println("=================================");

ex.printStackTrace();

}

}

* If the catch block is executed then it will display the output like below.

/ by zero

=================================

java.lang.ArithmeticException: / by zero

=================================

java.lang.ArithmeticException: / by zero

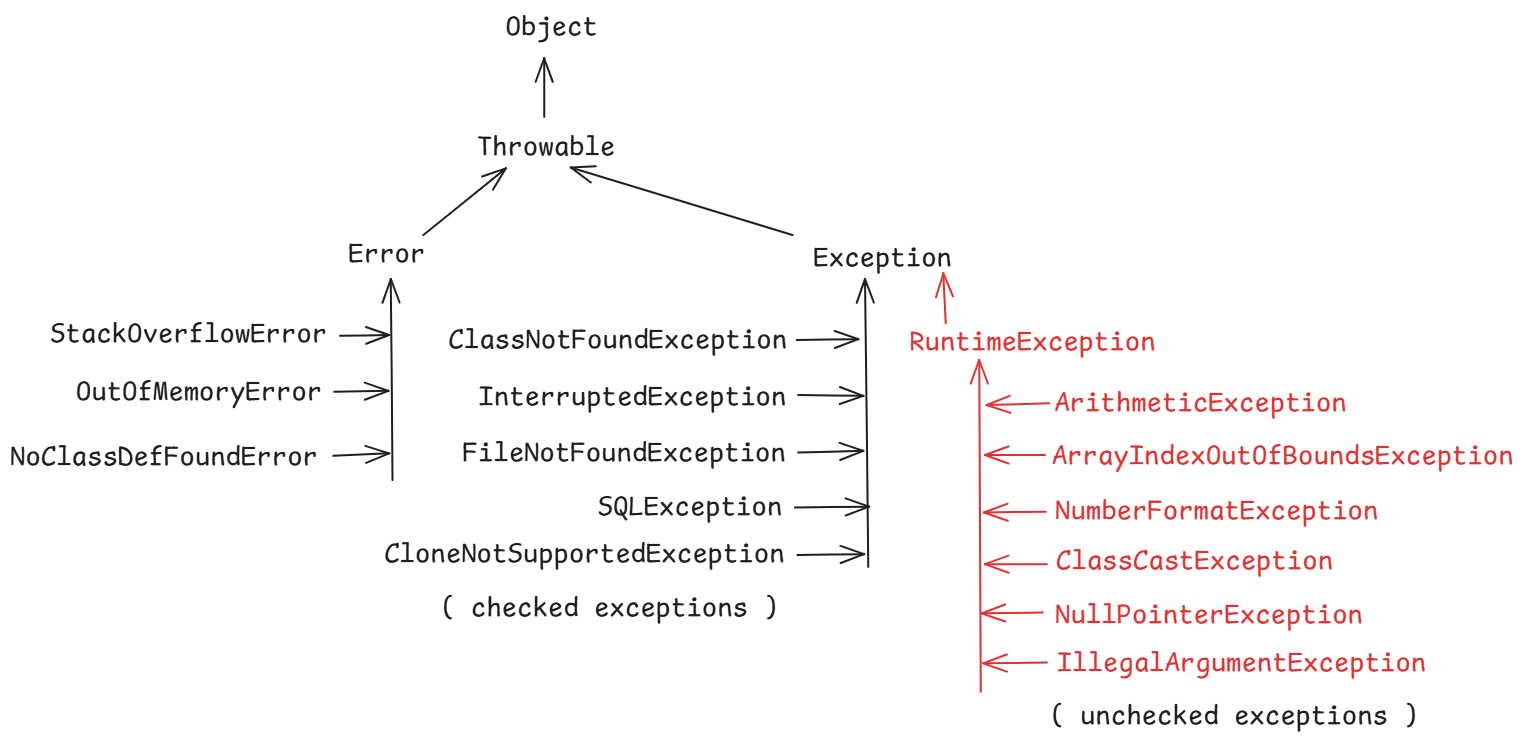
at SomeClass.someMethod(Main.java:7)

at Main.main(Main.java:28)

=============================================================================

Exception class hierarchy:

-----------------------



Q) what is the difference between Error and Exception?

A) Both are the child classes of a Parent class called Throwable.

Errors represents serious problems in the system/application that

a program should not try to catch. These are non-recoverable

problems.

Exceptions represents the problems in the application that a

program can address that problem. These are recoverable problems.

Q) How many types of exceptions are there in Java?

A) 2 types.

1. checked exceptions

2. unchecked exceptions

Q) what is the difference between checked and unchecked exceptions?

A) checked exceptions are known to the compiler and while compiling the code, it can foresee the exception that it might occur and verifies whether programmer is catching it or atleast propagating it with throws keyword. If not done anything, then generates a compile-time error.

unchecked exceptions are unknown to the compiler and while compiling the code, it doesn’t verify whether the programmer is catching it or atleast propagating it with throws keyword. So, the compiler doesn’t generate any compile-time error.

So, the Java compiler will force the programmer to catch or propagate the checked exception. But doesn’t force for the unchecked exception.

User-defined exceptions:

* we can create an user-defined exception, by extending our class from either Exception or from RuntimeException class.
* If you want to make your exception as a checked exception then extend Exception class.
* If you want to make your exception as a unchecked exception then extend RuntimeException class.
* While creating an user-defined exception class, we don’t create any variables or methods in that class, because we want to use it as an exception class, but not as a business class.
* You can create a constructor with a String parameter in the exception class or you can make your class as an empty class.
* While throwing the exception, suppose you want to throw a message also along with the exception then in that exception class you have to create a constructor with String parameter.
* While throwing the exception, suppose you want to throw the exception without a message, then you can create that exception class as empty.

Example:

public class EmployeeNotFoundException extends RuntimeException

{

public EmployeeNotFoundException(String message)

{

super(message);

}

}

public class DuplicateEmployeeException extends RuntimeException

{

}

* The throw keyword is used to raise an user-defined exception from the try block.
* With throw keyword, you can throw the exception object.
* For pre-defined exceptions, JVM will throw the exception object.

throws keyword:

--------------

* When an exception occurs in a method, there are 2 places where it can be handled.

1. in the same method only by using try and catch blocks.
2. in the caller method by using try and catch blocks.

* To delegate/propagate an exception from the current method to its caller method, throws keyword is used.
* If a method is not handling checked exceptions that might occur in that method, then it must propagate those exceptions to the caller method, by using throws keyword. Otherwise, Java compiler will generate an error.
* If a method is not handling unchecked exceptions that might occur in that method, then it is optional to propagate those exceptions to the caller method.
* throws keyword is used in the method signature to propagate/delegate the exceptions.
* With throws keyword, we can declare multiple exceptions also.
* If a caller method is not handling the exceptions, then it can also propagate the exceptions to its caller.

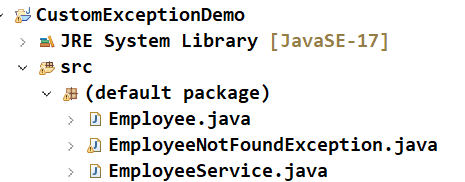
Q) what is the difference between throw and throws ?

A) 1. throw keyword is used inside the method body, but throws keyword is used in the method header.

2. With throw keyword, we can declare only one exception object. But with throws keyword, we can declare multiple exception classes.

3. With throw keyword, we can throw an exception and along with a message also. But with throws keyword, we can propagate only an exception, but not any message.

4. With throw keyword, we can raise custom exceptions. With throws keyword, we can propagate either custom exceptions or pre-defined exceptions.



Employee.java

-----------

**public** **class** Employee {

**private** **int** empno;

**private** String ename;

**public** Employee(**int** empno, String ename) {

**super**();

**this**.empno = empno;

**this**.ename = ename;

}

**public** **int** getEmpno() {

**return** empno;

}

**public** **void** setEmpno(**int** empno) {

**this**.empno = empno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

}

EmployeeNotFoundException.java

---------------------------

**public** **class** EmployeeNotFoundException **extends** Exception {

**public** EmployeeNotFoundException(String message) {

**super**(message);

}

}

EmployeeService.java

------------------

**public** **class** EmployeeService {

Employee[] employees;

**public** EmployeeService() {

employees = **new** Employee[3];

employees[0] = **new** Employee(7655, "Smith");

employees[1] = **new** Employee(7989, "Miller");

employees[2] = **new** Employee(7634, "Chris");

}

**public** Employee findById(**int** empno) {

**boolean** flag = **false**;

**try** {

**for** (Employee e : employees) {

**if** (e.getEmpno() == empno) {

flag = **true**;

**return** e;

}

}

**if** (flag == **false**) {

**throw** **new** EmployeeNotFoundException("Sorry, employee does not exist!");

}

} **catch** (EmployeeNotFoundException ex) {

System.***out***.println(ex);

}

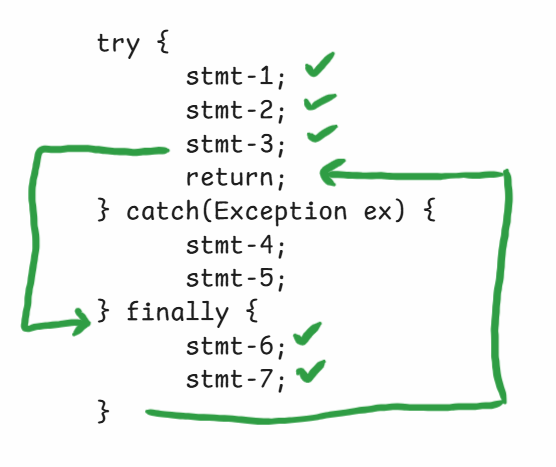
**return** **null**;

}

}

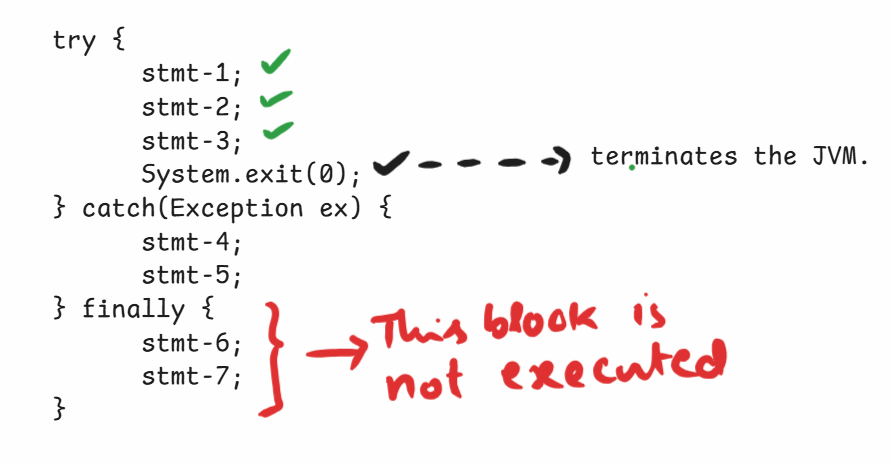
Q) if a try block has return statement, will finally block execute or not?

A) Yes, finally block will execute.



Q) if a try block has exit statement, will finally block executes?

A) No.



Q) can we write a single catch block to handle multiple exception types?

A) Yes. We can separate the exception classes with a pipe(|) symbol.

ex:

try {

stmt-1;

stmt-2;

stmt-3;

} catch(ArithmeticException|NullPointerException ex)

{

stmt-4;

stmt-5;

} finally {

stmt-6;

stmt-7;

}

try with resources statement:

---------------------------

* try with resources statement means, we can pass resources(objects) as parameters to the try statement.
* Generally, to clean up the resources like, Database connections, or Email server connections, or Input/Output streams, etc.., we write a finally block.
* In finally block, if you forget to close any resource, then memory leak issue will occur.
* So, to overcome this issue, Java has provided try with resources statement.
* In this, try with resources, the try statement only will automatically close/clean the resources after its execution, even if exeception occurs or doesn’t occur.
* So, you no need to write the finally block, to close the resources.
* The resources of the try statement must be either Closeable or AutoCloseable type. It means, the class of that object should implement Closeable or AutoCloseable interface.

ex:

try( FileReader fr = new FileReader(“D:\\one.txt”);

FileWriter fw = new FileWriter(“D:\\two.txt”) )

{

stmt-1;

stmt-2;

stmt-3;

}

catch(Exception ex) {

System.out.println(ex);

}

variable arguments(var args):

---------------------------

* If you specify 3 dots(...) as a suffix for a data type, then it is called var args.
* we used var args, for declaring a parameter of a method.
* When you want to create a method to accept variable number of arguments, not a fixed number of arguments then you have to use var args.

ex:

public void add(int a, int b) { //accepts fixed no of args(2)

//logic

}

ex:

public void add(int... arr) { //accepts variable no of args

//logic

}

* When you call a method with var args parameter, Java will internally creates an array, initializes the array with the given values and then Java will pass that array as a parameter to the var args parameter.

ex:

**class** SomeClass {

**static** **void** add(**int**... args)

{

**int** sum = 0;

//Normal for loop

/\*

for(int i = 0; i < args.length; i++)

{

sum = sum + args[i];

}

\*/

//for each loop

**for**(**int** x : args)

{

sum = sum + x;

}

System.***out***.println(sum);

}

}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

SomeClass.*add*(10,21,36,17);

}

}

Q) can we declare var args as a first parameter or as a middle parameter in a method declaration?

A) No. We can declare var args only as a last parameter.

ex:

public void add(int... args, double d) { //compile-time error

}

ex:

public void add(double d, int... arr, char ch) { //error

}

ex:

public void add(double d, char ch, int... arr) { //valid

}

Q) can we declare multiple var args parameter for a method?

A) No. We can declare atmost one var args parameter for a method.

Q) is the below main method of Java is correct?

static public void main(String... args)

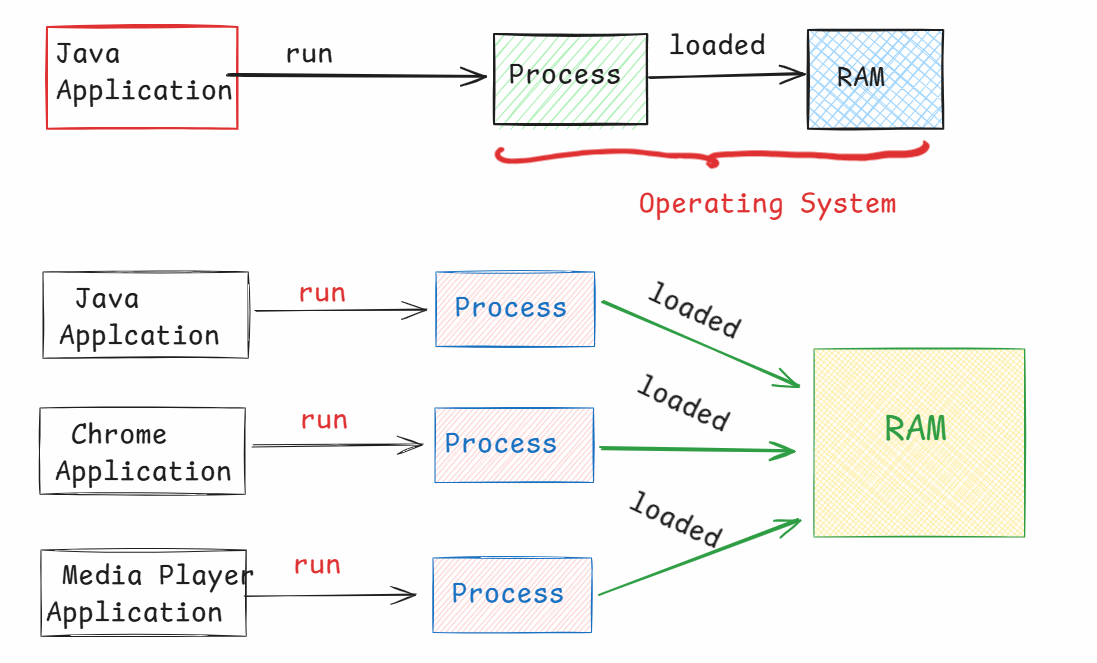
A) correct.

Multithreading

----------------

Introduction:

* When you run any application on your computer, the operating system will create a process and loads that process into RAM.
* When you run multiplications on your computer, the operating system will creates will create multiple processes and loads them into RAM.



* If the system has a single CPU Core, then all the processes will run on the Same core. So, the performance of the computer is slow.
* If the system has multiple CPU cores, like 4 cores/8 cores, then at a time, the OS can run 4 processes/8 processes concurrently(Exactly at the same time).
* Within a process/application, If you want to execute multiple tasks independently at the same time then you have to use a mechanism called “Multithreading”.

what is a Thread?

--------------

* A Thread is a group of instructions to perform a task, which are executing independently/separately from the rest of the application.
* For example:
* Suppose, you have an application, which has the tasks like upload the students details to the database and also generate the reports for the students.
* if these two tasks are executing sequentially, then tasks are delayed and application will take more time to complete.
* If you execute the two tasks parallely by creating two threads, then the tasks are finished fastly and application provides a good performance.

Q) what is the difference between process and thread?

A) 1. A process contains threads, but thread doesn’t process.

2. Switching from process to process will take more time for

CPU, but switching from thread to thread will take less

time for CPU. This switching is called context switch.

1. Each process will have a separate address space. But threads doesn’t have address space.

creating threads:

----------------

* In Java, threads can be created in 2 ways.

1. extend Thread class
2. implement Runnable interface.

* In both the ways, you have to override the run() method in the class to define the logic of a thread.

ex1:

class MyThread1 extends Thread {

@Override

public void run() {

//logic

}

}

ex2:

class MyThread2 implements Runnable {

@Override

public void run() {

//logic

}

}

Note: Runnable is a functional interface, because it has a single abstract method called run().

* After creating the thread classes, next we need to start the threads.
* If a thread is created by extending Thread class, then we can directly start the thread by calling start() method.
* If a thread is created by implementing Runnable interface, then we can’t directly start the thread.
* So, we need to wrap up the Runnable object into a new thread object, then we need to start().

example program:

--------------

* In this program, we are creating 2 threads, where one thread has a task to display 1 to 10 values and the other thread has to display 11 to 20 values.
* we have called sleep() method, to make a thread as waiting for given milliseconds.

**package com.ashokit.thread;**

**class MyThread1 extends Thread {**

**@Override**

**public void run() {**

**for(int i = 1; i <= 10; i++ ) {**

**System.*out*.println("i = " + i);**

**try {**

**Thread.*sleep*(2000); //2 sec**

**}**

**catch(InterruptedException ex) {**

**ex.printStackTrace();**

**}**

**}**

**}**

**}**

**class MyThread2 implements Runnable {**

**@Override**

**public void run() {**

**for(int j = 11; j <= 20; j++ ) {**

**System.*out*.println("j = " + j);**

**try {**

**Thread.*sleep*(2000); //2 sec**

**}**

**catch(InterruptedException ex) {**

**ex.printStackTrace();**

**}**

**}**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**MyThread1 t1 = new MyThread1();**

**//wrapping Runnable object into a Thread object**

**//we are wrapping, because Runnable object**

**//doesn't have a start() method.**

**Thread t2 = new Thread(new MyThread2());**

**t1.start();**

**t2.start();**

**}**

**}**

Q) which is the best way to create a thread in Java program. Is it extend Thread class or implement Runnable interface?

A) implement Runnable interface.

Question: why implement Runnable is the best way?

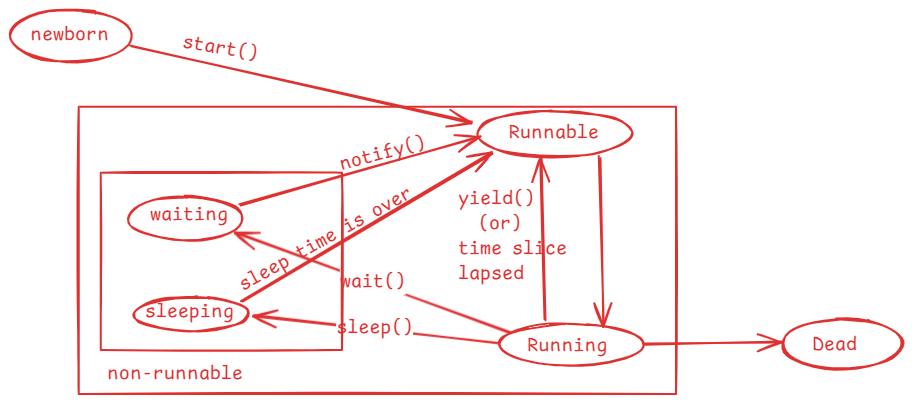
Answer: If you create a thread class by extending Thread then you can’t extend your class from another parent class. But if you create a thread class by implementing Runnable, then you class can extend another parent class.

Thread life cycle:

----------------

* Thread life cycle means, from thread creation to thread destruction, it will undergo different phases and these phases forms the life cycle.

1. Newborn
2. Runnable
3. Running
4. Non-Runnable
5. Dead



* when a thread object is created then the thread enters into newborn state.
* when start() method is called on a thread object then the thread is moved to the Runnable state.
* mulitple threads can sit in Runnable state at a time.
* Thread schedular will pick up one thread from Runnable state and moved

it into Running state.

* In Running state, run() method is called.
* When the time slice is lapsed/when yield() method is called, a thread is moved back from Running to Runnable state by the schedular.
* when sleep() is called, a thread is moved from Running state to the non-runnable state.
* once sleep time is completed, a thread is moved to Runnable state.
* when wait() is called, a thread is moved from Running state to non-runnable state.
* when notify()/notifyAll() method is called, then a waiting thread goes to Runnable state.
* If a thread’s execution is finished, then it goes to dead state.

key methods of Thread class:

------------------------------------

1. getId() : returns the identifier of a thread.
2. getName(): returns the name of a thread.
3. setName(String name): changes the name of this thread to the given name.
4. getPriority(): returns the proirity of a thread.
5. setPriority(int priority): changes the priority of this thread to the given value.
6. start(): moves this thread from newborn state to runnable state.
7. isAlive(): returns true, if a thread is alive.
8. isDaemon() : return true, if a thread daemon
9. setDaemon(boolean): changes this thread to a daemon thread.
10. getState(): returns the state of this thread.
11. yield(): It provides a hint to the schedular that the current thread is willing to come out of Running state, and want to give a chance for other high priority threads to enter into Running state. The schedular may accept it or may ignore it.
12. join(): This method will join this thread execution with another. So, the other thread will be suspended, until this joined thread is completed.
13. sleep(milliseconds): It makes a thread to enter into a sleeping state for a given milliseconds.

Thread priority variables:

-----------------------

* In java.lang.Thread class, three public static final variables are provided to represent the thread priority.

public static final int MIN\_PRIORITY = 1;

public static final int MAX\_PRIORITY = 10;

public static final int NORM\_PRIORITY = 5;

* A thred priority can be in between 1 to 10.
* The default priority is 5.
* If you change the priority of a thread to < 1 or > 10, then

IllegalArgumentException will be thrown.

**/\*\***

**\* yield() method demo application**

**\*/**

**package com.ashokit.thread;**

**class MyThread1 extends Thread {**

**@Override**

**public void run() {**

**//Thread.currentThread() returns a current thread reference**

**//getName() returns the name of the current thread.**

**System.*out*.println("Inside " + Thread.*currentThread*().getName());**

**for(int i=1; i<=5; i++) {**

**System.*out*.println("i = "+i);**

**Thread.*yield*();**

**}**

**System.*out*.println( Thread.*currentThread*().getName() + " finished");**

**}**

**}**

**class MyThread2 extends Thread {**

**@Override**

**public void run() {**

**System.*out*.println("Inside " + Thread.*currentThread*().getName());**

**for(int j=1; j<=5; j++) {**

**System.*out*.println("j = "+j);**

**Thread.*yield*();**

**}**

**System.*out*.println( Thread.*currentThread*().getName() + " finished");**

**}**

**}**

**public class MainClass {**

**public static void main(String[] args) {**

**MyThread1 t1 = new MyThread1();**

**MyThread2 t2 = new MyThread2();**

**t1.setName("First-Thread");**

**t2.setName("Second-Thread");**

**t1.start();**

**t2.start();**

**}**

**}**

**/\***

**\* join() method demo application**

**\* This example has two threads, main thread and other thread**

**\* we have joined other thread with the main thread**

**\* the main thread execution is suspended until the other thread**

**\* execution is finished.**

**\*/**

**package com.ashokit.thread;**

**class MyThread extends Thread {**

**@Override**

**public void run() {**

**System.*out*.println("Inside " +Thread.*currentThread*().getName());**

**for(int i = 1; i <=50; i++) {**

**System.*out*.println(" i = "+i);**

**try {**

**Thread.*sleep*(2000);**

**}**

**catch(InterruptedException ex) {**

**ex.printStackTrace();**

**}**

**}**

**System.*out*.println(Thread.*currentThread*().getName() +" is completed");**

**}**

**}**

**public class MainClass {**

**public static void main(String[] args) throws Exception {**

**Thread.*currentThread*().setName("Main Thread");**

**System.*out*.println("Inside : " + Thread.*currentThread*().getName());**

**MyThread t1 = new MyThread();**

**t1.setName("Other Thread");**

**t1.start();**

**for(int j = 51; j <= 70; j++ ) {**

**System.*out*.println(" j = "+j);**

**if(j==55) {**

**t1.join();**

**}**

**Thread.*sleep*(2000);**

**}**

**System.*out*.println(Thread.*currentThread*().getName()+ " is completed");**

**}**

**}**

Thread synchronization:

-----------------------

* If multiple threads are using a shared resource and trying to read/modify the data at the same time then we will get unpredictable results.
* Suppose, two people are withdrawing the money from the same bank account at the same time, then there is a chance of overdrawing the money.
* For example, balance available is 5000.0, and the two people at a time can see the balance as 5000.0 and if they withdraw 3000.0 each, then the total amount withdrawn is 6000.0. This is overdrawing the amount and leads to inconsistency.
* Threads synchronization controls access to a shared resource by multiple threads, by allowing only one thread to act on the shared resource at the same time.
* Threads synchronization can be implemented using synchronized keyword.
* with synchronized keyword(non-access modifier) we can create synchronized methods or synchronized blocks.
* synchronization works by acquiring the lock on an object by a thread.
* In Java, every object has a lock/monitor.
* A thread acquires the lock on the object, by calling synchronized method or synchronized block.
* Only one thread can acquire the lock on an object at a time.
* If two threads are calling a synchronized method at the same time, then one of the two threads acquires lock on the object.
* when the execution of a synchronized method/block is completed by a thread then the lock gets released.

example:

**/\***

**\* In this example, Course is a shared resource for two threads**

**\* The two threads are trying to register for the same course**

**\* which has only one seat available.**

**\* If synchronization is not applied, there is possibility that**

**\* the two threads registered for the same seat.**

**\* To avoid this inconsistency, we made the registerForCourse() method**

**\* as synchronized.**

**\* One thread at a time can execute the synchronized method, so that**

**\* we can avoid the inconsistency.**

**\*/**

**package com.ashokit.thread;**

**class Course {**

**private String courseName;**

**private int numOfSeatsAvailable;**

**public Course(String courseName, int numOfSeatsAvailable) {**

**super();**

**this.courseName = courseName;**

**this.numOfSeatsAvailable = numOfSeatsAvailable;**

**}**

**public synchronized void registerForCourse(int rollno) {**

**try {**

**if ( this.numOfSeatsAvailable - 1 < 0 ) {**

**throw new Exception("Sorry, seats are not available!. Your rollno : " + rollno);**

**}**

**numOfSeatsAvailable -= 1;**

**System.*out*.println("Booking successful!!! Your rollno is : " + rollno);**

**System.*out*.println("Available seats now : " + this.numOfSeatsAvailable);**

**}**

**catch(Exception ex) {**

**System.*out*.println("ERROR : " + ex.getMessage());**

**}**

**}**

**}**

**class RegisterThread extends Thread {**

**Course course;**

**int rollno;**

**RegisterThread(Course course, int rollno) {**

**this.course = course;**

**this.rollno = rollno;**

**}**

**@Override**

**public void run() {**

**course.registerForCourse(rollno);**

**}**

**}**

**public class MainClass {**

**public static void main(String[] args) {**

**Course course = new Course("CSE", 1);**

**RegisterThread t1 = new RegisterThread(course, 101);**

**RegisterThread t2 = new RegisterThread(course, 102);**

**t1.start();**

**t2.start();**

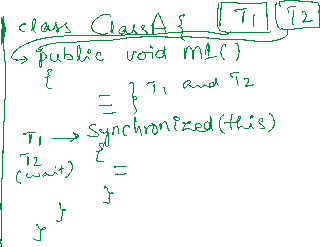
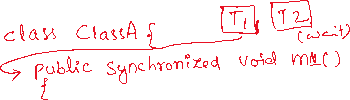
**}**

**}**

synchronized block:

-----------------

* Suppose, you have a synchronized method but only some statements of the method if they are executed simultaneously by two threads will cause inconsistency.
* In this case, when one thread is executing synchronized method, the other thread has to wait until the entire method is executed by the thread one. It means, the other thread has to wait for more time unnecessarily.
* So, to reduce the unnecessary waiting time for the threads, we can use synchronized blocks inside a method.
* If one thread is executing synchronized block, the other thread has to wait only until the thread1 has executed the block. So, waiting time is decreased.



* To the synchronized block, we should pass an object, whose lock a thread has to acquire as a parameter.
* If it is a non-static method, we can pass “this” as a parameter.
* If it is a static method, we can pass Classname.class as a parameter.

Q) How many synchronized blocks can be created in a method?

A) zero or multiple

Q) Which can improve the performance of an application, synchronized method or synchronized block?

A) synchronized block.

Deadlock:

* Suppose, we have two threads and two resources.
* thread1 holds the lock on resource1 and waiting for lock on resource2 and thread2 holds the lock on resource2 and waiting for lock on resource1.
* In this case, thread1 is blocked for resource2 and thread2 is blocked for resource1.
* So, the two threads will never complete. This is called deadlock situation.
* Deadlocks can be prevented by making the two threads to acquire the locks on the resources in the same order.
* **package pack1;**
* **class MyThread1 extends Thread**
* **{**
* **Object obj1;**
* **Object obj2;**
* **public MyThread1(Object obj1, Object obj2)**
* **{**
* **this.obj1=obj1;**
* **this.obj2=obj2;**
* **}**
* **@Override**
* **public void run() {**
* **synchronized(obj1) {**
* **System.*out*.println(Thread.*currentThread*().getName() +" has acquired lock on resource1");**
* **try {**
* **Thread.*sleep*(2000);**
* **}**
* **catch(InterruptedException ie) {**
* **System.*out*.println(ie);**
* **}**
* **System.*out*.println(Thread.*currentThread*().getName()+ " is waiting to acquire lock on resource2");**
* **synchronized(obj2) {**
* **System.*out*.println(Thread.*currentThread*().getName() +" has acquired lock on resource2");**
* **}**
* **}**
* **System.*out*.println(Thread.*currentThread*().getName()+" : is completed");**
* **}**
* **}**
* **class MyThread2 extends Thread**
* **{**
* **Object obj1;**
* **Object obj2;**
* **public MyThread2(Object obj1, Object obj2)**
* **{**
* **this.obj1=obj1;**
* **this.obj2=obj2;**
* **}**
* **/\***
* **@Override**
* **public void run() {**
* **synchronized(obj2) {**
* **System.out.println(Thread.currentThread().getName() +" has acquired lock on resource2");**
* **try {**
* **Thread.sleep(2000);**
* **}**
* **catch(InterruptedException ie) {**
* **System.out.println(ie);**
* **}**
* **System.out.println(Thread.currentThread().getName()+ " is waiting to acquire lock on resource1");**
* **synchronized(obj1) {**
* **System.out.println(Thread.currentThread().getName() +" has acquired lock on resource1");**
* **}**
* **}**
* **System.out.println(Thread.currentThread().getName()+" : is completed");**
* **}\*/**
* **@Override**
* **public void run() {**
* **synchronized(obj1) {**
* **System.*out*.println(Thread.*currentThread*().getName() +" has acquired lock on resource1");**
* **try {**
* **Thread.*sleep*(2000);**
* **}**
* **catch(InterruptedException ie) {**
* **System.*out*.println(ie);**
* **}**
* **System.*out*.println(Thread.*currentThread*().getName()+ " is waiting to acquire lock on resource2");**
* **synchronized(obj2) {**
* **System.*out*.println(Thread.*currentThread*().getName() +" has acquired lock on resource2");**
* **}**
* **}**
* **System.*out*.println(Thread.*currentThread*().getName()+" : is completed");**
* **}**
* **}**
* **public class DeadLock {**
* **public static void main(String[] args) {**
* **Object obj1 = new Object();**
* **Object obj2 = new Object();**
* **MyThread1 t1 = new MyThread1(obj1, obj2);**
* **t1.setName("Thread1");**
* **MyThread2 t2 = new MyThread2(obj1, obj2);**
* **t2.setName("Thread2");**
* **t1.start();**
* **t2.start();**
* **}**
* **}**

Inter-thread communication:

* Inter-thread communication is possible between synchronized threads.
* If we want one synchronized thread to wait and wants to give chance to another synchronized thread to execute then we need inter-thread communication.
* For inter-thread communication, we have to use the following methods of the Object class.

1. wait() : moves current thread to waiting state, until another invokes notify()/notifyAll() method.
2. wait(millis) : moves current thread to waiting state until another thread invokes notify()/notifyAll() method, or time has lapsed.
3. notify(): notifies a single waiting thread, to resume
4. notifyAll(): notifies all waiting threads, to resume

**/\***

**\* In this example, Course is a shared resource for three threads**

**\* The two threads are trying to register for the same course**

**\* which has only one seat available and the third thread**

**\* is cancelling the seat.**

**\* The one thread gets booking success, and the other thread waits to**

**\* see for the cancellation of seat.**

**\* The cancellation thread cancels the seat and notifies the waiting thread.**

**\* so, the other thread also gets booking successful.**

**\***

**\*/**

**package com.ashokit.thread;**

**class Course {**

**private String courseName;**

**private int numOfSeatsAvailable;**

**public Course(String courseName, int numOfSeatsAvailable) {**

**super();**

**this.courseName = courseName;**

**this.numOfSeatsAvailable = numOfSeatsAvailable;**

**}**

**public synchronized void registerForCourse(int rollno) {**

**try {**

**if(this.numOfSeatsAvailable - 1 < 0) {**

**System.*out*.println("seats are not available now, you please wait!!");**

**System.*out*.println("May be meanwhile another student may cancel his seat");**

**this.wait(200000);**

**}**

**if ( this.numOfSeatsAvailable - 1 < 0 ) {**

**throw new Exception("Sorry, seats are not available!. Your rollno : " + rollno);**

**}**

**System.*out*.println("Booking successful!!! Your rollno is : " + rollno);**

**numOfSeatsAvailable -= 1;**

**System.*out*.println("Available seats now : " + this.numOfSeatsAvailable);**

**System.*out*.println("===================================================");**

**}**

**catch(Exception ex) {**

**System.*out*.println("ERROR : " + ex.getMessage());**

**}**

**}**

**public synchronized void cancelSeats() {**

**try {**

**this.numOfSeatsAvailable += 1;**

**System.*out*.println("Cancellation Successful");**

**System.*out*.println("Available seats now : " + this.numOfSeatsAvailable);**

**System.*out*.println("==============================================");**

**this.notify();**

**}**

**catch(Exception ex) {**

**ex.printStackTrace();**

**}**

**}**

**}**

**class RegisterThread extends Thread {**

**Course course;**

**int rollno;**

**RegisterThread(Course course, int rollno) {**

**this.course = course;**

**this.rollno = rollno;**

**}**

**@Override**

**public void run() {**

**course.registerForCourse(rollno);**

**}**

**}**

**class CancelThread extends Thread {**

**Course course;**

**CancelThread(Course course) {**

**this.course = course;**

**}**

**@Override**

**public void run() {**

**course.cancelSeats();**

**}**

**}**

**public class MainClass {**

**public static void main(String[] args) {**

**Course course = new Course("CSE", 1);**

**RegisterThread t1 = new RegisterThread(course, 101);**

**RegisterThread t2 = new RegisterThread(course, 102);**

**t1.start();**

**t2.start();**

**try {**

**Thread.*sleep*(18000);**

**}**

**catch(Exception ex) {**

**System.*out*.println(ex);**

**}**

**CancelThread t3 = new CancelThread(course);**

**t3.start();**

**}**

**}**

Q) why wait(), notify() and notifyAll() methods are given in Object class, not in Thread class?

A) 1. When you synchronize on an object, the thread acquires lock

associated with the object.

2.In Inter-thread communication, the threads coordination happens by

acquiring and releasing the locks on the object. So, the wait(),

notify() and notifyAll() are ties to the lock of the object, not the

thread.

1. That’s why these methods are given in Object class, but not in Thread class.

What is a deamon thread?

--------------------

2 types of threads

1. user threads

2. daemon threads

\* by default, each thread is a user thread.

\* Main thread will not terminate the currently running JVM, until user threads are completed.

\* If make any thread as daemon thread, then Main thread does not wait for the completion of deamon thread.

\* If all the user threads in application are completed, then main thread will terminate the JVM. So, the deamon threads are also terminated.

\* setDeamon(true) method can make a thread as daemon thread.

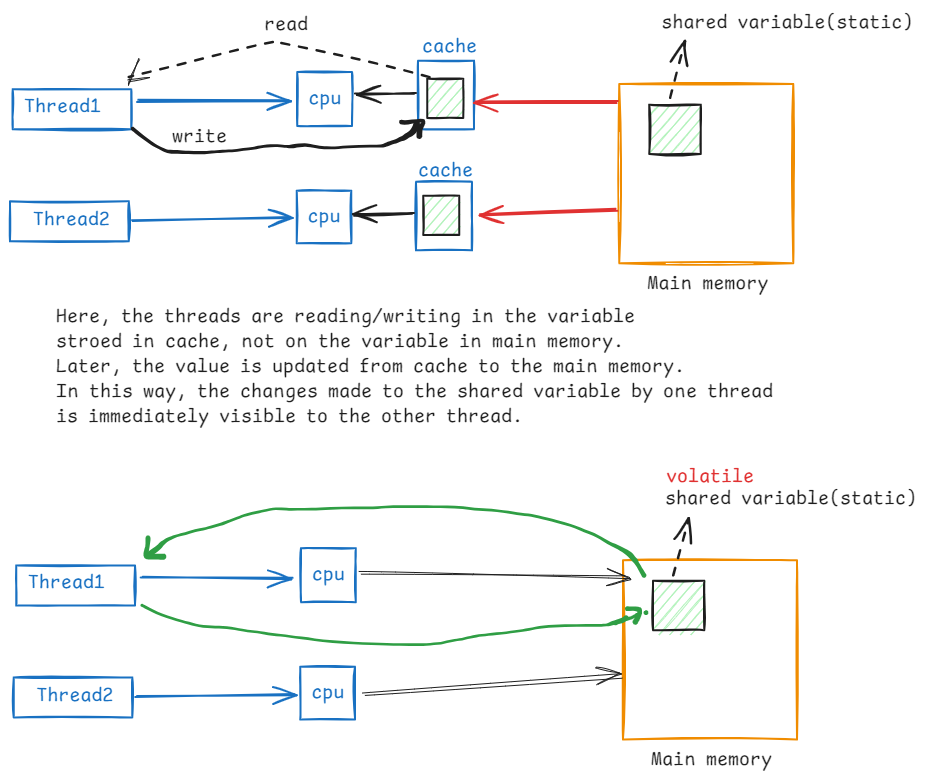
\* isDeamon() method can be used to verify that a thread is daemon thread or user thread.

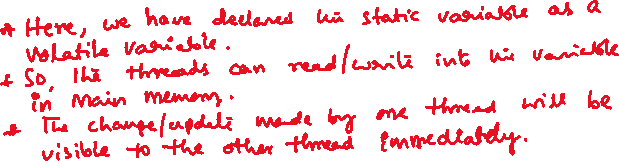
\* we create daemon threads in an application, to perform any background activities. For example, performance monitoring of an application.

\* In JVM, Garbage Collector thread is a daemon thread and it executes continously until JVM is terminated.

volatile keyword:

* In Java, when a thread is executing on a CPU core, it allows a thread to cache variable for better performance.
* If there is a shared variable(static) for two threads, then the variable gets cached and the threads are performing read/write operations on the variable in cache.
* So, the changes made to the shared variable by one thread does not immediately reflect in the main memory. Hence, the other thread can’t access the updated value.
* To avoid this issue, volatile is provided and if we declare a variable as volatile, then a thread can not store it in cache and the threads can perform read/write operations directly on the shared variable in main memory.





Q) can we call run() method directly on a thread, without calling start()?

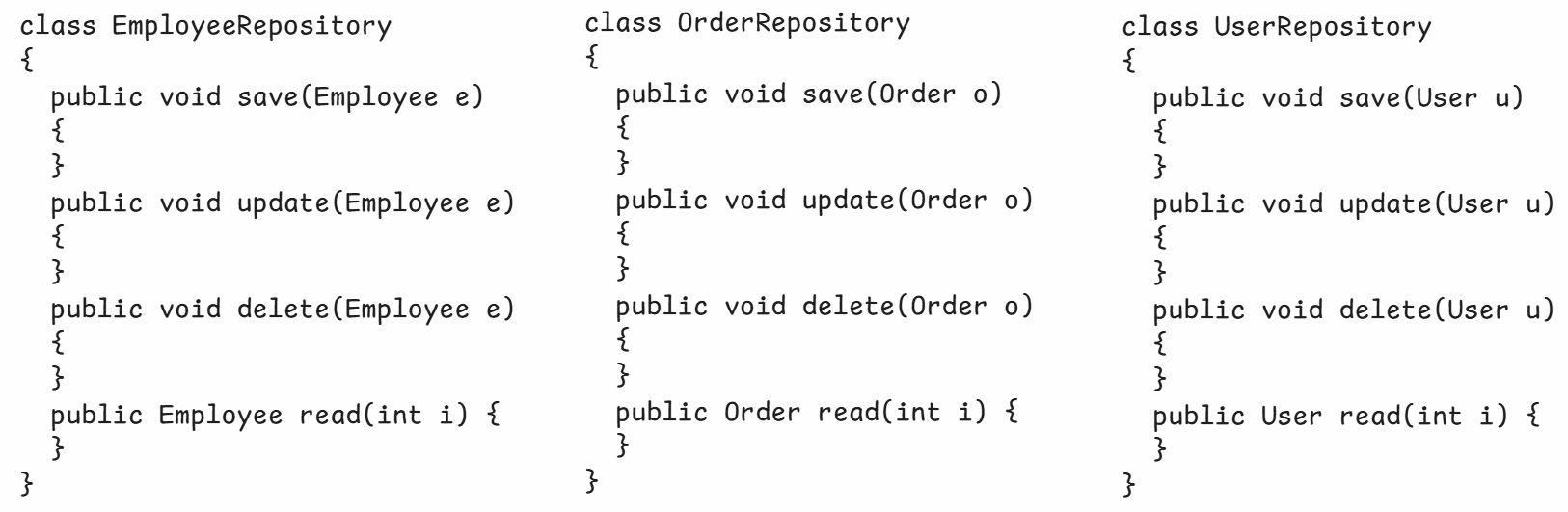
A) Yes. Now the threads will execute sequentially, but not concurrently.

Q) can we call start() method on a thread for more than once?

A) We will get IllegalThreadStateException.

Generic class?

* Suppose, you have a requirement that you need multiple classes with similar operations, but they should perform the operations on different types of objects.
* For example,



* Instead of creating multiple classes, we can create a single class and we can use it for different types of objects, using generics.
* So, with generics, we can reduce the number of classes in a project.
* A generic class can be created by adding a type parameter as a suffix for the classname.

Collections

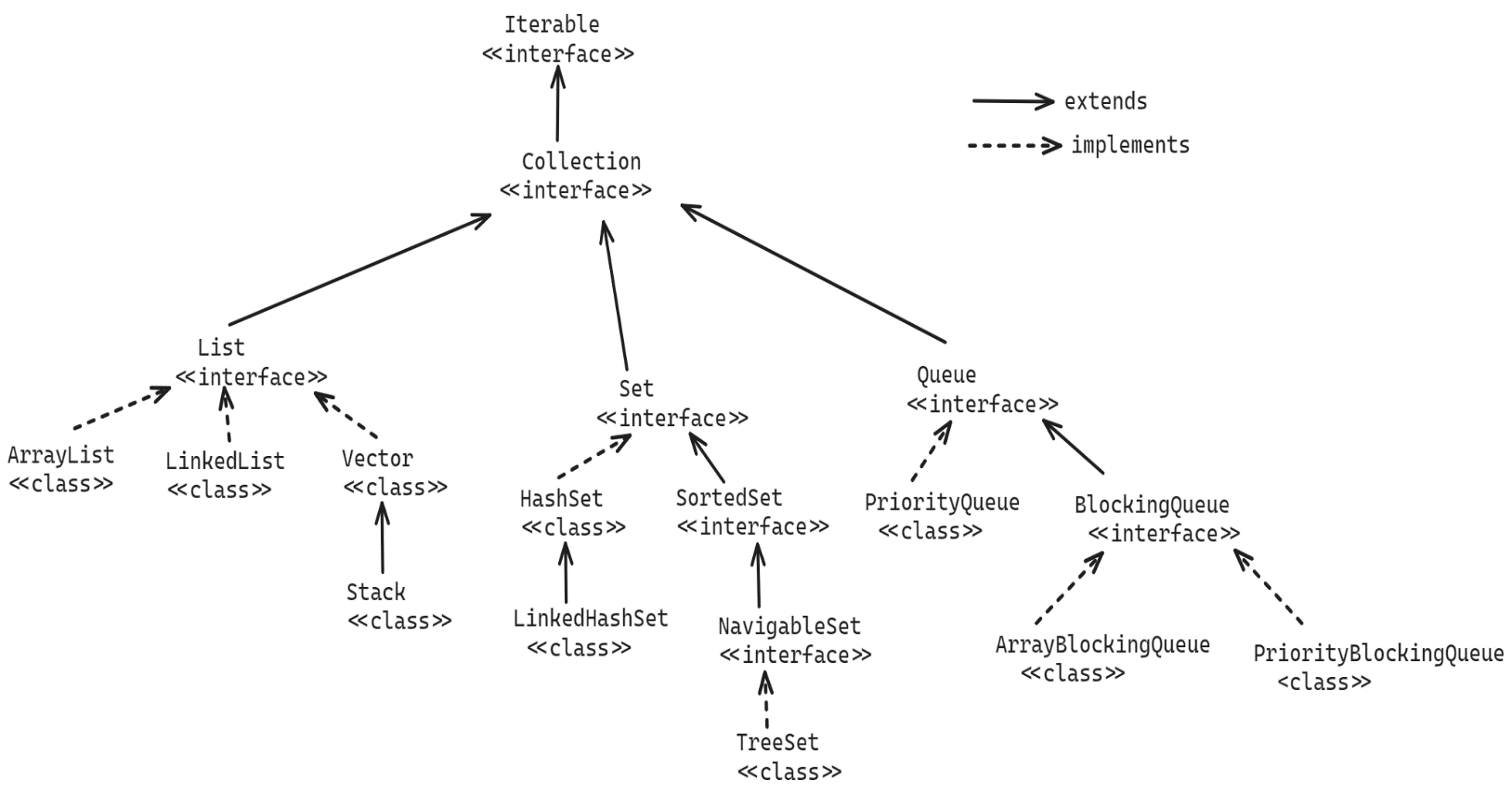
----------------------------------

* Data structures is a way to organaize, store and manage the data efficiently, so that it can be used effectively.
* To organize the data efficiently, we have different data structure principles.

* For example,

1. array : fixed set of elements, and elements can be accessed fastly.
2. linked list : It is a group of elements connected with links, where add/remove of an elements is easy and can grow or shrink automatically.
3. LIFO(stack) : Last In First Out. The elements can be added/removed from the same end.
4. FIFO(queue): First In First Out. The elements can be added from the last and removed from the first.
5. Map : The elements are stored in key-value pairs.
6. tree : The elements are arranged in the hierachical order.
7. graph : The elements are connected with multiple edges.

* Java has provided, collections API, which contains a set of interfaces and classes, which are developed by implementing the data structure principles.
* So, as a programmer we can directly use the pre-defined classes to organize our data efficiently, without implementing those principles manually.
* The main interface of Java collections is the java.util.Collection interface.
* The hieararchy of Collection interface is,



* Collection interface has 3 child interfaces, for different requirements.

1. List : can store the elements in the insertion order, can store duplicate elements and you can access eleements by their index.
2. Set : can store unique elements, do not maintain insertion order, and you can’t access elements with index.
3. Queue: can store the elements in the FIFO, can process elements based on priority, can store duplicate elements.

Methods of Collection interface:

1. add(E e) : Adds an element to the collection. If added successfully then returns true, otherwise returns false.
2. addAll(Collection c): Adds all the elements of the specified collection to this collection.
3. clear(): removes all the elements from this collection.
4. remove(E e) : removes a specified element from this collection. If removed successfully then returns true, otherwise returns false.
5. contains(Object o): checks whether a specified object exists in this collection or not. If exist, returns true. Otherwise, returns false.
6. iterator(): returns an Iterator object over this collection.
7. size(): returns the number of elements in this collection.
8. isEmpty(): returns true, if this collection is empty. Otherwise, returns false.
9. toArray(): returns an array for this collection.
10. removeAll(Collection c): removes the elements from this collection, that are also available in the specified collection.
11. retainAll(Collection c): Retains only the elements in this collection that are contained in the specified collection.

List<E> interface:

------------------

* When you want to store the elements in a sequence in a collection object then you have to use List object.
* In a List object, each element has position, which is also called index.
* Because the elements are stored in the List object in the order of insertion, we call List as a ordered collection.
* In a List object, you can store duplicate elements also.
* The implementation classes of List interface are,

ArrayList

LinkedList and

Vector

* The methods of Collection interface are inherited to List interface and also List interface has provided the below key methods.

1. add(index, element): inserts an element at the given position.
2. get(index): returns the element from the given position.
3. indexOf(element): returns the position of the element.
4. lastIndexOf(element): returns the posistion of the last occurrence of the element.
5. remove(index): removes the element from the given position.
6. listIterator() : returns a ListIterator object.
7. set(index, element): replaces an element at the given position.

Traversing a list :

* Traversing a list means, squentially accessing the elements of a list.
* To traverse a list, we have 4 options.

1. Iterator object
2. for each loop
3. for loop
4. ListIterator object.

* Iterator is an interface in java.util package and an Iterator object will access the elements of a collection object one at a time.
* In Collection interface, iterator() method is provided and different implementation classes in collections have overridden this iterator() method in their own way.
* iterator() method creates and returns a new Iterator object.
* The Iterator object is created as a wrapper object on top of the exisiting collection object. So, it doesn’t consume more memory.
* Iterator object maintains a cursor, and the cursor is by default placed at before the first element of the collection.
* Iterator interface has provided methods to access the elements.

1. hasNext() : return true, if next element existing after the cursor
2. next() : first moves the cursor to the next element and returns that element.
3. remove(): removes the element at the cursor.

a sample code:

ArrayList<Integer> arrList = new ArrayList<Integer>();

arrList.add(10);

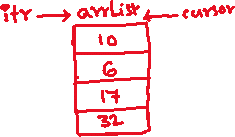
arrList.add(6);

arrList.add(17);

arrList.add(32);



Iterator<Integer> itr =



arrList.iterator();

while(itr.hasNext())

{

Interger i = itr.next();

S.o.println(i);

}

* A for each loop can be used like below.

for(Integer i : arrList)

{

S.o.println(i);

}

* A for loop can be used like below.

for(int i=0; i<arrList.size(); i++)

{

S.o.println(arrList.get(i));

}

* ListIterator interface extends Iterator interface and it has provided additional methods, to move the cursor in backward direction.
* hasPrevious()
* previous()

ArrayList class:

---------------

* ArrayList class uses a resizable array(dynamic array) internally to store the elements.
* When we create ArrayList object, internally an array **Object[] elementData** is created with capacity of 10 elements.
* When the elements are execeed the capacity, a new larger array is created with capacity increment of 50%, and elements are copied from previous array to the new array.
* You can specify the initial capacity at the time constructing ArrayList object.

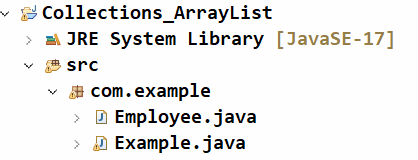
For example,

ArrayList<String> arr1 = new ArrayList<>(); //constructs a list with initial

capacity of ten.

ArrayList<String> arr2 = new ArrayList<>(8); //constructs a list with the

specified initial capacity.



Employee.java

-------------

**package** com.example;

**public** **class** Employee {

**private** **int** empno;

**private** String ename;

**private** **double** sal;

**private** **int** deptno;

**public** Employee(**int** empno, String ename, **double** sal, **int** deptno) {

**super**();

**this**.empno = empno;

**this**.ename = ename;

**this**.sal = sal;

**this**.deptno = deptno;

}

**public** **int** getEmpno() {

**return** empno;

}

**public** **void** setEmpno(**int** empno) {

**this**.empno = empno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **double** getSal() {

**return** sal;

}

**public** **void** setSal(**double** sal) {

**this**.sal = sal;

}

**public** **int** getDeptno() {

**return** deptno;

}

**public** **void** setDeptno(**int** deptno) {

**this**.deptno = deptno;

}

@Override

**public** String toString() {

**return** "Employee [empno=" + empno + ", ename=" + ename + ", sal=" + sal + ", deptno=" + deptno + "]";

}

}

Example.java

-----------

**package** com.example;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**public** **class** Example {

**public** **static** **void** main(String[] args) {

ArrayList<Employee> employees = **new** ArrayList<>();

employees.add( **new** Employee(7891, "KING", 7000, 10) );

employees.add( **new** Employee(7655, "MARK", 5000, 20) );

employees.add( **new** Employee(7123, "SMITH", 9000, 10) );

employees.add( **new** Employee(7506, "DAVID", 8000, 30) );

employees.add( **new** Employee(7112, "MARY", 3000, 10) );

employees.add( **new** Employee(7788, "SCOTT", 5000, 20) );

employees.add( **new** Employee(7264, "ALLEN", 9000, 30) );

employees.add( **new** Employee(7993, "SOPHIA", 2000, 10) );

employees.add( **new** Employee(7021, "MILLER", 4000, 20) );

/\*

ArrayList<Employee> newList = getEmployeesBySalLessThan(employees, 6000);

System.out.println("List of employees with sal < 6000");

for(Employee e : newList)

{

System.out.println(e);

}

\*/

System.***out***.println("-------------------------------------");

*removeEmployeesStartsWith*("M", employees);

}

**private** **static** ArrayList<Employee> getEmployeesBySalLessThan(ArrayList<Employee> employees, **double** sal)

{

ArrayList<Employee> newList = **new** ArrayList<>();

//for each loop

**for**(Employee e : employees) {

**if**(e.getSal() < sal )

{

newList.add(e);

}

}

**return** newList;

}

**private** **static** **void** removeEmployeesStartsWith(String letter, ArrayList<Employee> employees) {

Iterator<Employee> itr = employees.iterator();

**while**(itr.hasNext()) {

Employee e = itr.next();

**if**(e.getEname().startsWith(letter)) {

itr.remove();

}

}

//display the remaining list

**for**(Employee e : employees) {

System.***out***.println(e);

}

}

}

Note:

code-1:

List<String> lst = new ArrayList<>();

lst.add(“John”);

lst.add(“Tom”);

lst.add(“Bob”);

code-2:

List<String> lst = Arrays.asList(“John”, “Tom”, “Bob”);

* The above code-1 and code-2 are same.
* we can create an ArrayList class object with values easily by calling asList() method of Arrays class.

sorting the elements of a List:

=============================

* To sort the elements of a List, 2 ways are available.

1. Comparable<T> interface
2. Comparator<T> interface

* Both are Functional interfaces, because they have provided a single abstract method.
* The single abstract method of Comparable<T> interface is,

int compareTo(T o)

* The single abstract method of Comparator<T> interface is,

int compare(T o1, T o2)

* compareTo() method Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
* compare() method Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
* When you want to define a single sorting strategy on the elements/objects of a class, then you have to use Comparable<T> interface.
* When you want to define mulitple sorting strategies on the elements of a class, then you have to use Comparator<T> interface.
* To sort the elements using Comparable, first implement a class from Comparable interface, whose elements you want to sort.
* For example, I have a list of employees and I want to sort the employees based on their salaries, then I have to implement Comparable interface into the Employee class.
* Next, To sort the elements of the list, call Collections.sort(list) method.
* The below example code sorts the list of employees, based on the salaries in ascending order.

Employee.java

-----------

**package** com.example;

**public** **class** Employee **implements** Comparable<Employee>{

**private** **int** empno;

**private** String ename;

**private** **double** sal;

**private** **int** deptno;

**public** Employee(**int** empno, String ename, **double** sal, **int** deptno) {

**super**();

**this**.empno = empno;

**this**.ename = ename;

**this**.sal = sal;

**this**.deptno = deptno;

}

**public** **int** getEmpno() {

**return** empno;

}

**public** **void** setEmpno(**int** empno) {

**this**.empno = empno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **double** getSal() {

**return** sal;

}

**public** **void** setSal(**double** sal) {

**this**.sal = sal;

}

**public** **int** getDeptno() {

**return** deptno;

}

**public** **void** setDeptno(**int** deptno) {

**this**.deptno = deptno;

}

@Override

**public** String toString() {

**return** "Employee [empno=" + empno + ", ename=" + ename + ", sal=" + sal + ", deptno=" + deptno + "]";

}

@Override

**public** **int** compareTo(Employee o) {

**if**(**this**.getSal() < o.getSal() )

**return** -1;

**else** **if**(**this**.getSal() > o.getSal() )

**return** 1;

**else**

**return** 0;

}

}

Example.java

----------

**public** **class** Example {

**public** **static** **void** main(String[] args) {

ArrayList<Employee> employees = **new** ArrayList<>();

employees.add( **new** Employee(7891, "KING", 7000, 10) );

employees.add( **new** Employee(7655, "MARK", 5000, 20) );

employees.add( **new** Employee(7123, "SMITH", 9000, 10) );

employees.add( **new** Employee(7506, "DAVID", 8000, 30) );

employees.add( **new** Employee(7112, "MARY", 3000, 10) );

employees.add( **new** Employee(7788, "SCOTT", 5000, 20) );

employees.add( **new** Employee(7264, "ALLEN", 9000, 30) );

employees.add( **new** Employee(7993, "SOPHIA", 2000, 10) );

employees.add( **new** Employee(7021, "MILLER", 4000, 20) );

Collections.*sort*(employees);

//display the elements

**for**(Employee e : employees) {

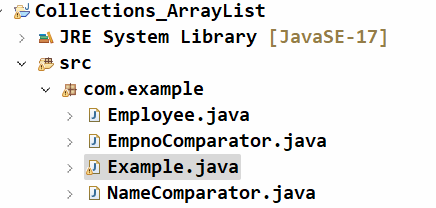
System.***out***.println(e);

}

}

}

* If you want to sort the elements of a class, stored in a list object using Comparator, then you have to create a separate comparator class for each strategy.
* For example, if you want to sort the employees in a list, in empno’s sorting order and also ename’s sorting order, then you have to create two comparator classes.
* The below example contains a list of employees and we are sorting the list on empno’s and also on ename’s by creating two comparator classes.



Employee.java

-----------

**public** **class** Employee {

**private** **int** empno;

**private** String ename;

**private** **double** sal;

**private** **int** deptno;

**public** Employee(**int** empno, String ename, **double** sal, **int** deptno) {

**super**();

**this**.empno = empno;

**this**.ename = ename;

**this**.sal = sal;

**this**.deptno = deptno;

}

**public** **int** getEmpno() {

**return** empno;

}

**public** **void** setEmpno(**int** empno) {

**this**.empno = empno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **double** getSal() {

**return** sal;

}

**public** **void** setSal(**double** sal) {

**this**.sal = sal;

}

**public** **int** getDeptno() {

**return** deptno;

}

**public** **void** setDeptno(**int** deptno) {

**this**.deptno = deptno;

}

@Override

**public** String toString() {

**return** "Employee [empno=" + empno + ", ename=" + ename + ", sal=" + sal + ", deptno=" + deptno + "]";

}

}

EmpnoComparator.java

-----------------

**public** **class** EmpnoComparator **implements** Comparator<Employee> {

**public** **int** compare(Employee o1, Employee o2) {

**if**( o1.getEmpno() < o2.getEmpno() )

**return** -1;

**else** **if**( o1.getEmpno() > o2.getEmpno() )

**return** 1;

**else**

**return** 0;

};

}

NameComparator.java

------------------

**public** **class** NameComparator **implements** Comparator<Employee> {

**public** **int** compare(Employee o1, Employee o2) {

**return** o1.getEname().compareTo(o2.getEname());

};

}

Example.java

----------

**public** **class** Example {

**public** **static** **void** main(String[] args) {

ArrayList<Employee> employees = **new** ArrayList<>();

employees.add(**new** Employee(7891, "KING", 7000, 10));

employees.add(**new** Employee(7655, "MARK", 5000, 20));

employees.add(**new** Employee(7123, "SMITH", 9000, 10));

employees.add(**new** Employee(7506, "DAVID", 8000, 30));

employees.add(**new** Employee(7112, "MARY", 3000, 10));

employees.add(**new** Employee(7788, "SCOTT", 5000, 20));

employees.add(**new** Employee(7264, "ALLEN", 9000, 30));

employees.add(**new** Employee(7993, "SOPHIA", 2000, 10));

employees.add(**new** Employee(7021, "MILLER", 4000, 20));

employees.sort(**new** EmpnoComparator());

// display the elements

System.***out***.println("Employees in empno's order");

**for** (Employee e : employees) {

System.***out***.println(e);

}

System.***out***.println("++\*\*++".repeat(40));

employees.sort(**new** NameComparator());

// display the elements

System.***out***.println("Employees in name's order");

**for** (Employee e : employees) {

System.***out***.println(e);

}

}

}

Q) What is the difference between Comparable and Comparator?

A) 1. Comparable interface is provided in java.lang package, and

Comparator interface is provided in java.util package.

2. Comparable is used to implement a single sorting strategy, and

Comparator is used to implement multiple sorting strategies.

1. To implement Comparable, we implement it into the class of the elements. To implement Comparator, we need to create a separate class.

4. Comparable has compareTo() method with a single input argument

and Comparator has compare() method with two input arguments.

Q) ArrayList object is thread-safe or not?

A) Not a thread-safe object. But we can make it as thread-safe object, with the below statement.

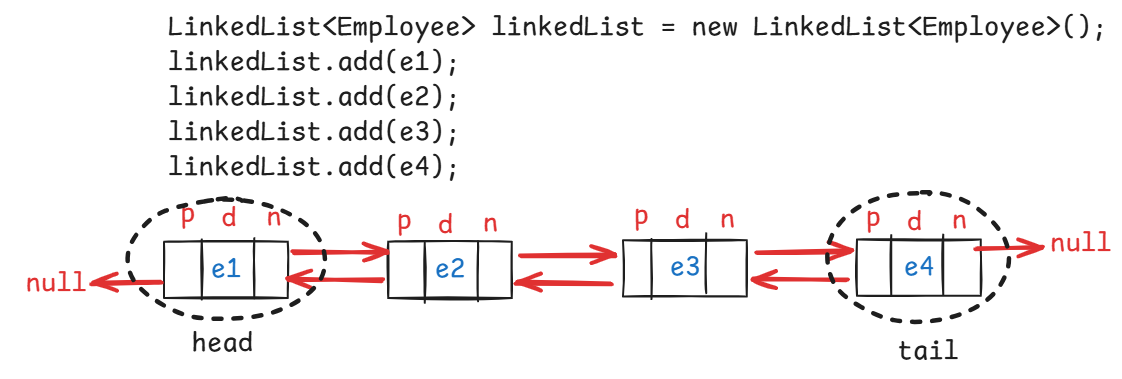
lst = Collections.synchronizedList(lst);

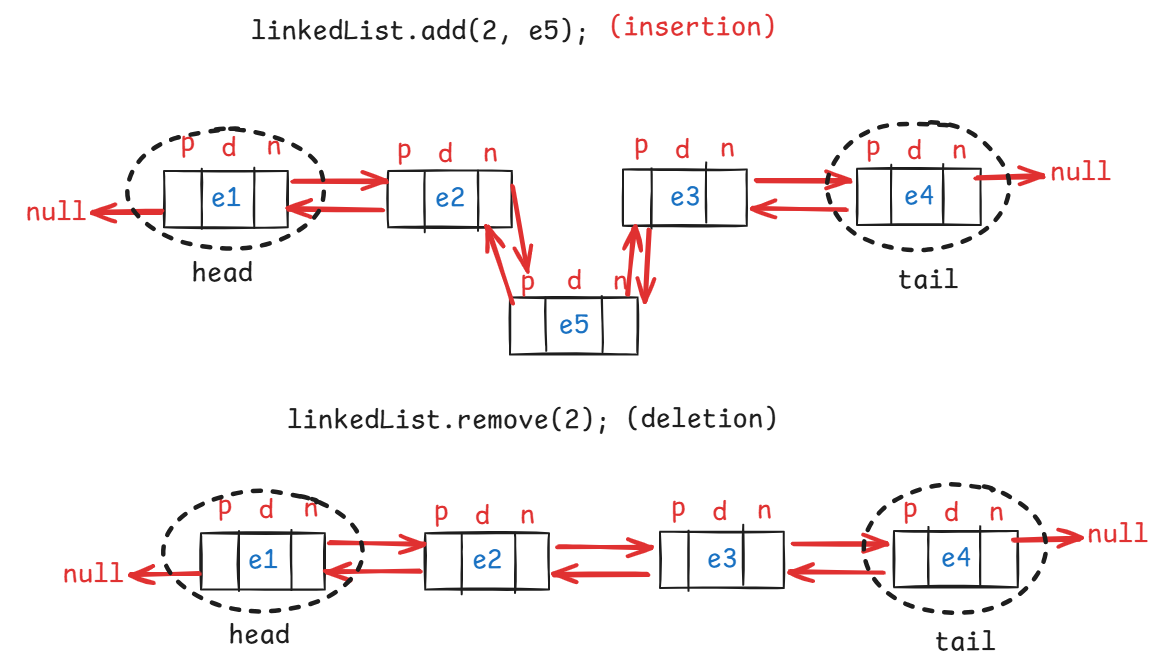
LinkedList class

* LinkedList class is also an implementation of List interface.
* In LinkedList, the elements are stored as nodes.
* So, a LinkedList is a collection of nodes.
* Each node has 3 parts.

1. previous pointer
2. data
3. next pointer

* previous pointer holds the address of the previous node, next pointer holds the address of the next node, and the data part contains the actual element.
* So, a LinkedList object is a double linked list implementation of List interface.
* The first node of the LinkedList is called head/front and the last node of the LinkedList is called tail/rear.





* To a LinkedList object, eleements can be added/removed from head or from tail also.
* The methods are,

addFirst(E e)

addLast(E e)

getFirst()

getLast()

peekFirst()

peekLast()

pollFirst()

pollLast()

removeFirst()

removeLast()

* Difference between getFirst() and peekFirst() is,

if the list is empty, getFirst() throws NoSuchElementException, but peekFirst() returns null.

* Difference between peek, poll and remove is,

. peek methods will return the element, but does not remove the element. If the linked list is empty then returns null.

. poll methods will return and removes the element. If the linked list is empty then returns null.

. remove methods will return and removes the element. If the linked list is empty then throws NoSuchElementException.

Q) When to use ArrayList and When to use LinkedList?

A) If you need retrieval more times and insertion/deletion of elements less times then use ArrayList.

If you need insertion/deletion of elements more times and retrieval less times then use LinkedList.

Q) LinkedList object is thread safe or not?

A) No, not a thread safe object. But we can make it as thread-safe.

linkedLst = Collections.synchronizedList(linkedLst);

hashCode() and equals() method:

--------------------------

* hashCode() and equals() methods are available in Object class.
* hashCode() method of Object class returns the memory address of the object.
* equals() method of Object class works like equality operator(==).
* Suppose if two different objects of a class have the same data, the hashCode() method of Object class returns the different values. Because their memory addresses are different.
* Similarly, if two differennt objects of a class have the same data, the equals() method of Object class returns false. Because the reference variables are pointing to the different objects.
* But, when we are working with the collections like Set and Map, we have a requirement that, if two different objects of a class have the same data then hashCode() method should return the same value and equals() method should return true.
* So, you have to override the hashCode() and equals() methods in your class.
* In IDE, you no need to override them manually. You can override them with source option.
* Right click in the class, source then Generate hashCode and equals methods.

java.util.Set interface

* Set is an unordered collection of elements without duplicates.
* A Set object doesn’t store the duplicate elements. If a duplicate element is added, then it will skip the element.
* It is unordered collection, because it doesn’t store the elements in the same order internally.
* The implementation classes of Set interface are,

HashSet, LinkedHashSet and TreeSet.

* The set objects internally uses map objects, where map objects uses hash table data structure.
* hash table data structure stores the elements in key-value pairs.
* When you add an element to the set object, that element will be stored as a key and with value as PRESENT.
* Here, PRESENT is a dummy object, created for Object class.

HashSet<E> class:

---------------

* When you create a HashSet class object, it internally creates HashMap class object, to store the elements.
* HashMap class uses hash table data structure and the table is created with default initial capacity as 16 and load factor 0.75.
* While creating HashSet class object, you can also specify the intial capacity and load factor.

HashSet<String> set = new HashSet<>();

. initial capacity : 16

. load factor : 0.75

HashSet<String> set = new HashSet<>(10);

. initial capacity : 10

. load factor : 0.75

HashSet<String> set = new HashSet<>(10, 0.5f);

. initial capacity : 10

. load factor : 0.5

* The capacity of hash table structure will be doubled based on capacity and load factor.
* if capacity is 16 and load factor is 0.75 then the capacity will be doubled after adding 12(16 \* 0.75) pairs to the hash table.
* if the capacity is 10 and load factor is 0.5 then the capacity will be doubled after adding 5( 10 \* 0.5) pairs to the hash table.
* When you are adding an element to a set object, first hashcode is calculated and then based on the hashcode, bucket index is calculated and then the element will be stored into that bucket as a key with value as PRESENT.
* To access the elements of a set object, you have to use Iterator object or for each loop or for loop.

//A sample code

**public** **class** Example {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Set<String> set = **new** HashSet<>();

set.add("orange");

set.add("apple");

set.add("banana");

set.add("mango");

set.add("grapes");

set.add("banana");

System.***out***.println("Size of this Set object : " + set.size());

//for each loop

System.***out***.println("The elements in this Set object : ");

**for**(String str : set) {

System.***out***.println(str);

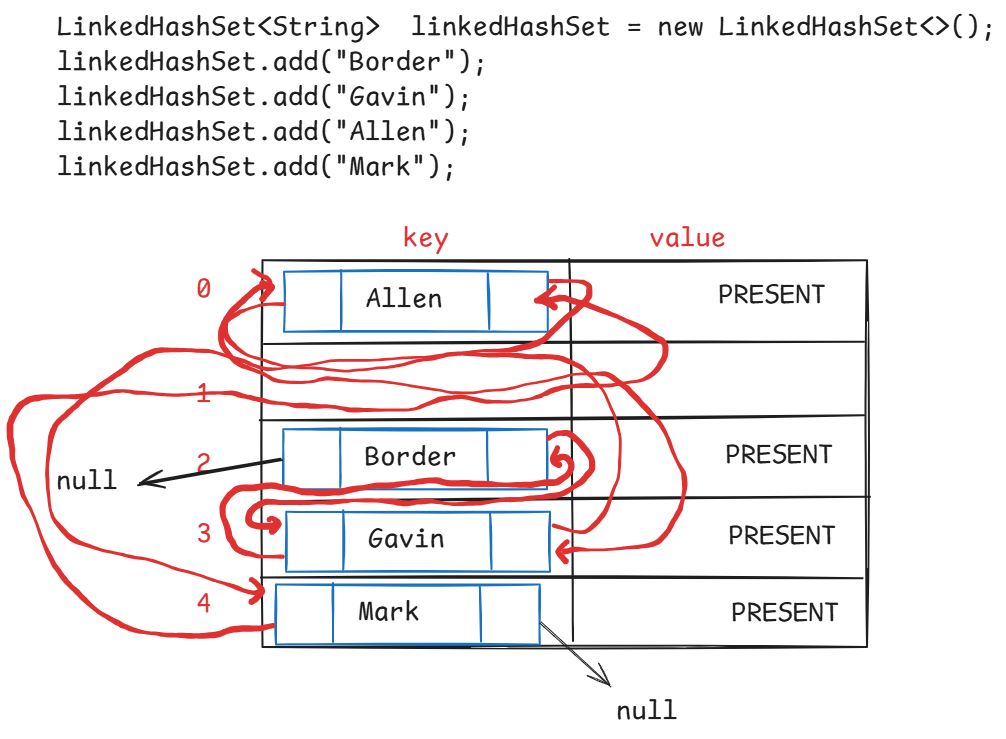
}

}

}

LinkedHashSet<E> class:

* It is Hash table and linked list implementation of the Set interface, with predictable iteration order.
* This implementation differs from HashSet in this it maintains a doubly-linked list running through all of its entries.
* This linked list defines the iteration ordering, which is the order in which elements were inserted into the set (*insertion-order*).



Q) HashSet object is thread safe or not?

A) No, not a thread-safe. But we can make it as thread safe object.

hashSet = Collections.synchronizedSet(hashSet);

Q) LinkedHashSet object is thread safe or not?

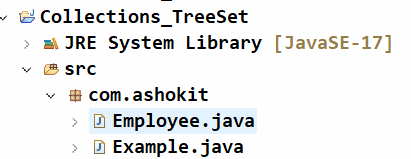
A) No, not a thread-safe. But we can make it as thread safe object.

linkedHashSet = Collections.synchronizedSet(linkedHashSet);

TreeSet<E> class:

---------------------

* When you want a collection to store the elements without duplicates and also it should maintain the elements in sorting order, then use TreeSet class.
* TreeSet class internally uses TreeMap object.
* while constructing a TreeSet object, if we do not provide Comparator object as parameter then the TreeSet will sort the elements according to the natural sorting order defined by the Comparable.
* If the elements are not Comparable type, then ClassCastException will be thrown.
* To make the objects added to the TreeSet as Comparable objects, the class of that objects should implement Comparable interface.
* Suppose, if you want to add the objects of Employee class to the TreeSet collection, that Employee class should implement Comparable interface.
* Suppose, if you are constructing a TreeSet object, by providing Comparator parameter, then that Employee class no need to implement Comparable interface.



Employee.java

**package** com.ashokit;

**import** java.util.Objects;

**public** **class** Employee **implements** Comparable<Employee> {

**private** **int** empno;

**private** String ename;

**private** **double** sal;

**private** **int** deptno;

**public** Employee(**int** empno, String ename, **double** sal, **int** deptno) {

**super**();

**this**.empno = empno;

**this**.ename = ename;

**this**.sal = sal;

**this**.deptno = deptno;

}

**public** **int** getEmpno() {

**return** empno;

}

**public** **void** setEmpno(**int** empno) {

**this**.empno = empno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **double** getSal() {

**return** sal;

}

**public** **void** setSal(**double** sal) {

**this**.sal = sal;

}

**public** **int** getDeptno() {

**return** deptno;

}

**public** **void** setDeptno(**int** deptno) {

**this**.deptno = deptno;

}

@Override

**public** String toString() {

**return** "Employee [empno=" + empno + ", ename=" + ename + ", sal=" + sal + ", deptno=" + deptno + "]";

}

@Override

**public** **int** hashCode() {

**return** Objects.*hash*(deptno, empno, ename, sal);

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Employee other = (Employee) obj;

**return** deptno == other.deptno && empno == other.empno && Objects.*equals*(ename, other.ename)

&& Double.*doubleToLongBits*(sal) == Double.*doubleToLongBits*(other.sal);

}

@Override

**public** **int** compareTo(Employee o) {

**if** (**this**.getEmpno() < o.getEmpno() )

**return** -1;

**else** **if** ( **this**.getEmpno() > o.getEmpno() )

**return** 1;

**else**

**return** 0;

}

}

Example.java

**package** com.ashokit;

**import** java.util.TreeSet;

**public** **class** Example {

**public** **static** **void** main(String[] args) {

TreeSet<Employee> ts = **new** TreeSet<>();

ts.add(**new** Employee(7891, "KING", 7000, 10));

ts.add(**new** Employee(7655, "MARK", 5000, 20));

ts.add(**new** Employee(7123, "SMITH", 9000, 10));

ts.add(**new** Employee(7506, "DAVID", 8000, 30));

ts.add(**new** Employee(7112, "MARY", 3000, 10));

ts.add(**new** Employee(7788, "SCOTT", 5000, 20));

ts.add(**new** Employee(7264, "ALLEN", 9000, 30));

ts.add(**new** Employee(7993, "SOPHIA", 2000, 10));

ts.add(**new** Employee(7021, "MILLER", 4000, 20));

System.***out***.println("Display the elements of TreeSet");

**for**(Employee e : ts) {

System.***out***.println(e);

}

}

}

Q) when to use HashSet, LinkedHashSet and TreeSet?

A) When you need a collection which should not store duplicates and it should no need to maintain insertion order of elements then use HashSet.

When you need a collection which should not store duplicates and it should need to maintain insertion order of elements then use LinkedHashSet.

When you need a collection which should not store duplicates and it should need to maintain sorting order of elements then use TreeSet.

Queue<E> interface:

-------------------

* Queue orders elements in First In First Out(FIFO) manner, but not necessarily.
* A Queue represents a collection designed for holding the elements prior to processing.
* UseCase1: In Operating Systems, processes are added to the Queue as they arrive. The CPU picks a process from the front of the queue and executes it.
* UseCase2: In a Customer Service call center, incoming calls are placed in a queue. The first call is attended first.
* UseCase3: Employees submit print jobs, they are stored in a queue. The printer processes jobs one by one from the front of the queue.
* A queue maintains two pointers called head/fornt and tail/rear.
* The elements are inserted from the tail pointer and removed from the head pointer.
* The methods for inserting the element into the Queue are,

1. add(E e) : adds an element and returns true, if successful. Otherwise, throws IllegalStateException.
2. offer(E e): adds an element and returns true, if successful. Otherwise, returns false.

* The methods for removing element from the Queue are,

1. poll() : retrieves and removes the element from the head pointer. Returns null if the queue is empty.
2. remove(): retrieves and removes the element from the head pointer. Throws NoSuchElementException, if the queue is empty.

* The methods for retrieving the element from the Queue are,

1. element() : retrieves but does not remove the element from the head pointer. Throws NoSuchElementException, if the queue is empty.
2. peek() : retrieves but does not remove the element from the head pointer. Returns null, if the queue is empty.

Deque<E> interface:

-------------------

* Deque<E> interface extends Queue<E> interface, and it is called double ended queue.
* It means, a Deque object allows addition/deletion/retrieval of elements from both head and tail pointers.
* The methods are,

1. addFirst(e)
2. addLast(e)
3. offerFirst(e)
4. offerLast(e)
5. getFirst()
6. getLast()
7. removeFirst()
8. removeLast()
9. pollFirst()
10. pollLast()
11. peekFirst()
12. peekLast()

* PriorityQueue class implements Queue interface, and LinkedList class implements Deque interface.

PriorityQueue<E> class:

* PriorityQueue class does not maintain the elements in the FIFO order. It maintains the elements in the natural sorting order or the order defined by the Comparator object.
* The default capacity of the PriorityQueue object is 11.
* The PriorityQueue object is an unbounded queue, which means, there is no size restrictions. We can add any number of elements.
* The elements must be Comparable elements, for sorting in natural sorting order.
* While constructing PriorityQueue object, if we pass Comparator object as a parameter then the elements may not be Comparable elements.
* we can not add null value to PriorityQueue object. If added, we will get NullPointerException.

=============================

BlockingQueue<E> interface:

---------------------------

* It is a queue, which additionally supports blocking operations.
* A blocking queue will have a fixed capacity.
* If a thread wants to add an element, but the queue is full, then the thread has to wait until the space becomes available.
* If a thread wants to remove an element, but the queue is empty, then the thread has to wait until the queue becomes non-empty.
* The blocking operations are put() and take().
* When put() is called, if the queue is full, put() will wait for the space.
* when take() is called, if the queue is empty, take() will wait for the element.
* The implementation classes are, ArrayBlockingQueue, PriorityBlockingQueue and LinkedBlockingQueue.
* For creating producer-consumer applications, producer adds items to the queue and consumer remvoes items from the queue.
* If the queue is full, producer waits until space becomes available. If the queue is empty, consumer waits until item becomes available.
* For example, In a Job execution application, one thread is adding the job to the queue and the other thread is removing the job from the queue and executing it.
* If the queue if full, the first thread waits to add the job, until space becomes available. If the queue is empty, the other threads waits until a job is added.

Example code on ArrayBlockingQueue<E> class:

**import** java.util.concurrent.ArrayBlockingQueue;

**class** Thread1 **extends** Thread {

ArrayBlockingQueue<Integer> abq;

**public** Thread1(ArrayBlockingQueue<Integer> abq) {

**this**.abq = abq;

}

@Override

**public** **void** run() {

System.***out***.println("Inside : " + Thread.*currentThread*().getName());

**for** (**int** i = 1; i <= 6; i++) {

**try** {

**if**(i==6) {

System.***out***.println("Element removed : " + abq.take());

}

abq.put(i);

System.***out***.println("element put is : " + i);

} **catch** (Exception ex) {

System.***out***.println(ex);

}

}

System.***out***.println("finished : " + Thread.*currentThread*().getName());

}

}

**public** **class** Solution {

**public** **static** **void** main(String[] args) {

ArrayBlockingQueue<Integer> abq = **new** ArrayBlockingQueue<Integer>(5);

Thread1 t1 = **new** Thread1(abq);

t1.start();

}

}

output:

Inside : Thread-0

element put is : 1

element put is : 2

element put is : 3

element put is : 4

element put is : 5

Element removed : 1

element put is : 6

finished : Thread-0

Q) what is the difference between array and ArrayList?

A) 1. array has a fixed size. But ArrayList has dynamic size.

2. array can hold homogeneous elements. But ArrayList can hold heterogeneous elements.

3. arrays use less memeory. But ArrayList use more memory

4. You have to use loops to perform operations on arrays. But ArrayList has methods like add(), remove(), size(), etc…

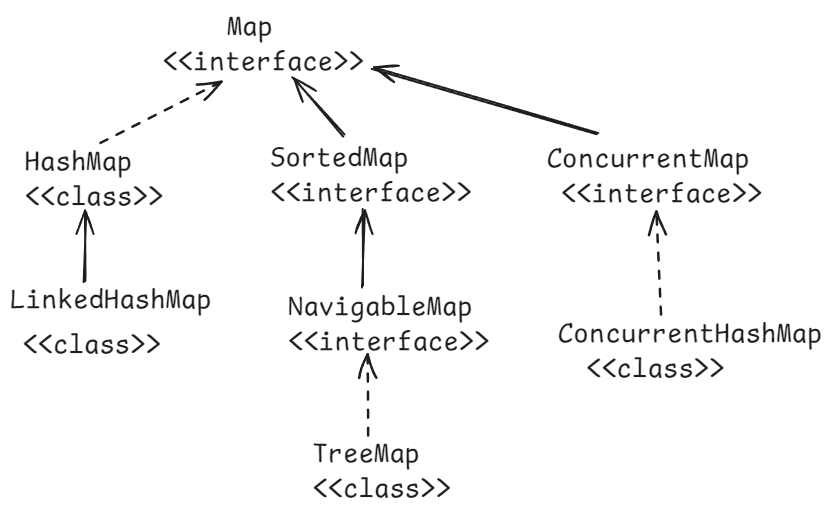
Q) what is the difference between ArrayList and Vector classes?

A) 1. ArrayList is not a thread-safe object. But Vector is a thread-safe object.

2. ArrayList performance is faster. But Vector performance is slower.

3. ArrayList increases size by 50% when capacity is exceeded. But Vector increases the size by 100% when capacity is exceeded.

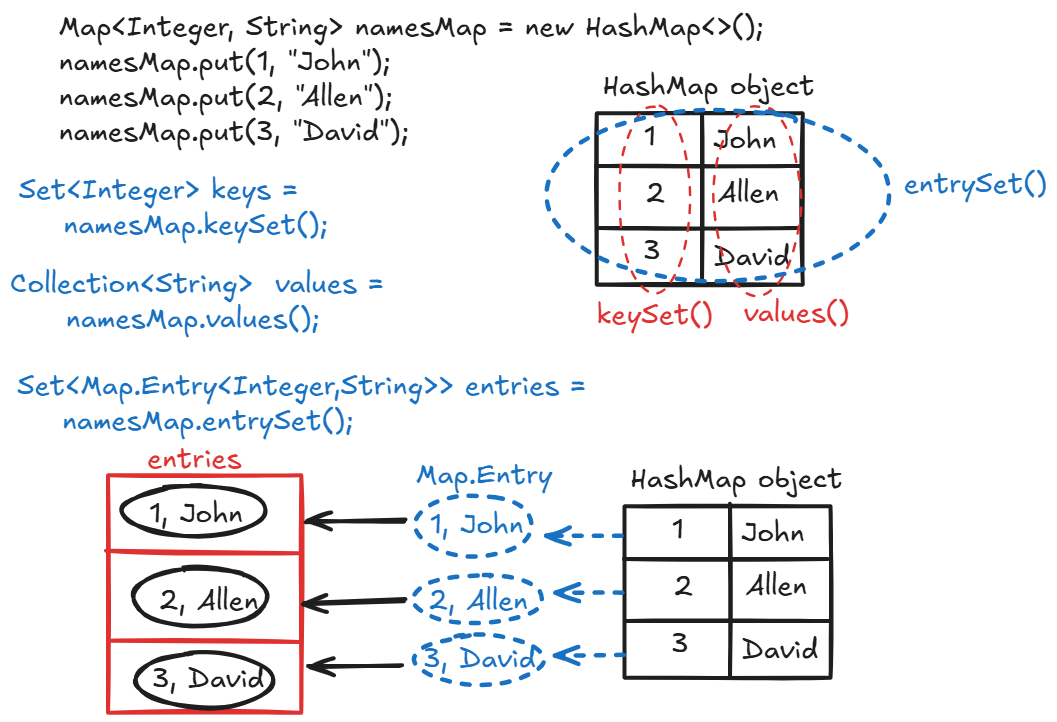
Map interface



* When you want a collection to store the elements in a key&value pairs, then you have to use Map object.
* A Map object internally uses hash table data structure, which can store the data in the key & value pairs.
* A Map object can not store duplicate keys. It means, keys are unique. But it can store duplicate values.
* A Map object can store atmost one null key, but multiple null values.
* If you put a duplicate key, then a map object will replace the previous value with the current value for that key.

methods of Map interface:

* + 1. put(key,value): inserts the key-value to the map and returns the value.
    2. get(key): retruns the value of the specified key. Returns null if the key doesn’t exist.
    3. remove(key): removes the mapping from this map object for the key, and returns the value. If the map is empty, then returns null.
    4. replace(key,value): replaces the value for a key in this map, if the key exists. Otherwise returns null.
    5. containsKey(key): returns true, if this map object contains the specified key. Otherwise, returns false.
    6. containsValue(value): returns true, if this map object contains the specified value. Otherwise, returns false.
    7. keySet() : returns keys of this map object into a Set. Returns null if this map object is empty.
    8. values(): returns values of this map object into a set. Returns null if this map object is empty.
    9. entrySet(): returns the map entries as Map.Entry objects into a set. Returns null, if the map object is empty.
    10. size() : returns the number of mappings in this map object.
    11. clear(): removes all the mappings from this map object.
    12. isEmpty(): returns true, if this map is empty. Otherwise, returns false.
    13. putIfAbsent(key,value): puts this key-value, if the key is absent in this map object. If the key already exist, then returns its current value. But does not replace the value.



Q) When to use HashMap object?

A) When you want to store the data as key and value pairs in a collection and if you do not want to maintain the data in insertion order of the keys then you can use HashMap object.

Q) When to use LinkedHashMap object?

A) When you want to store the data as key and value pairs in a collection and if you want to maintain the data in insertion order of the keys, then you can use LinkedHashMap object.

Q) When to use a TreeMap object?

A) When you want to store the data as key and value pairs in a collection and if you want to maintain the data in sorting order of the keys, they you can use TreeMap object.

//example on HashMap methods

**package** com.ashokit;

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Set;

**public** **class** Example {

**public** **static** **void** main(String[] args) {

Map<String, String> hashMap = **new** HashMap<>();

//country code is key, country name is value

hashMap.put("AUS", "Australia");

hashMap.put("NZ", "Newzeland");

hashMap.put("IND", "India");

hashMap.put("UK", "United Kingdom");

hashMap.put("US", "United States of America");

//retrieve the keys

Set<String> keys = hashMap.keySet();

System.***out***.println("The keys in this Map object");

**for**(String key : keys) {

System.***out***.println("key : " + key);

}

//retrieve the values

Collection<String> values = hashMap.values();

System.***out***.println("The values in this Map object");

**for**(String v : values) {

System.***out***.println("value : " + v);

}

//retrieve the entries

Set<Entry<String, String>> entries = hashMap.entrySet();

System.***out***.println("The entries in this Map object");

**for**(Entry<String, String> entry : entries) {

System.***out***.println("key : " + entry.getKey() + " , value : " + entry.getValue());

}

}

}

Q) HashMap object is thread safe or not?

A) No, not a thread safe object.

Q) can we make HashMap object as thread safe or not?

a) Yes. Call synchronizedMap() method of Collections class.

ex:

employeesMap = Collections.synchronizedMap(employeesMap);

Q) HashMap object is mutable object or not?

A) Yes. We can add/remove/replace the elements in HashMap object.



Q) can we make HashMap object as immutable or not?

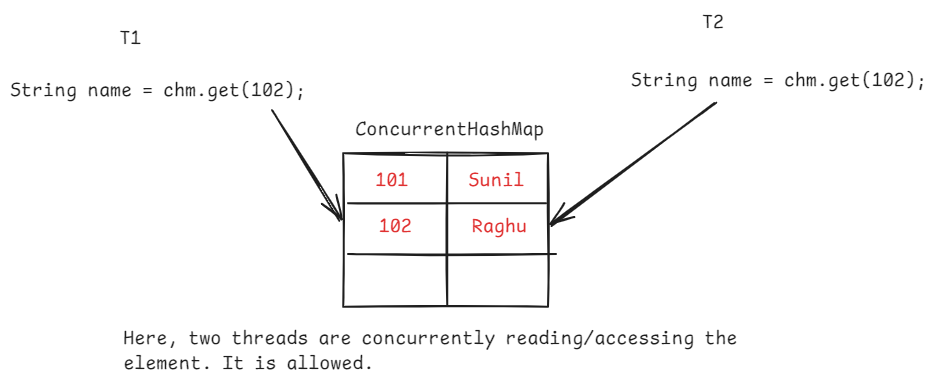
A) Yes. Call unmodifiableMap() method of Collections class.

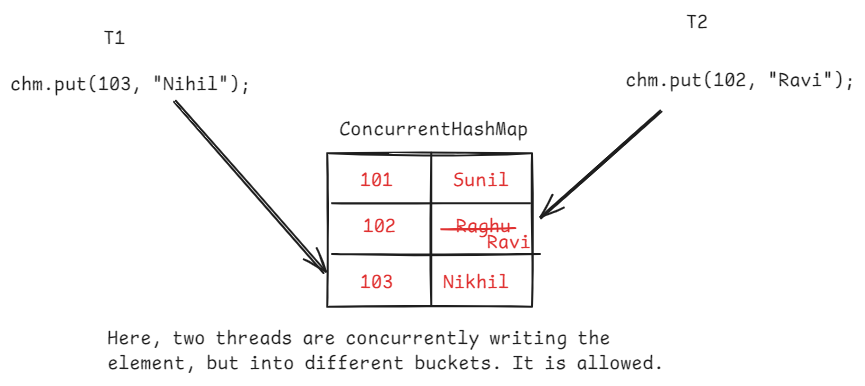
ex:

employeesMap = Collections.unmodifiableMap(employeesMap);

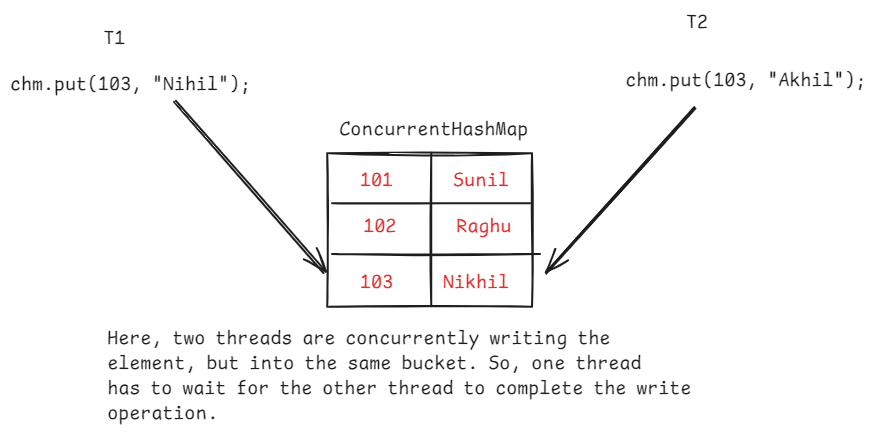
ConcurrentHashMap :

* ConcurrentHashMap class implements ConcurrentMap interface and ConcurrentMap interface extends Map interface.
* ConcurrentHashMap class is a thread-safe variant of HashMap class.
* ConcurrentHashMap allows multiple threads at a time to access its elements.
* ConcurrentHashMap provides better performance in multi-threading applications, because of bucket-level locking.
* ConcurrentHashMap also has default initial capacity 16 with load factor 0.75.









Q) what is difference between HashMap and ConcurrentHashMap objects?

A) 1. HashMap object is not a thread-safe by default. But ConcurrentHashMap object is thread-safe by default.

2. If HashMap object is converted to thead-safe, then entire HashMap object is locked. So, one thread has to wait, for the completion of another thread, even though the two threads wants to modify the different buckets.

But in ConcurrentHashMap object, the threads are no need to wait, while working with different buckets. So, ConcurrentHashMap object provides better performance.

3.HashMap object allows to store null key and null value. But ConcurrentHashMap object can not store null key or null value.

4. Iterator on HashMap object is fail-fast and Iterator on ConcurrentHashMap object is fail-safe.

Q) what is the difference between Hashtable and HashMap objects?

A) 1. Hashtable is a legacy class introduced in Java 1.0

HashMap is a collection framework class introducted in Java 1.2

2. Hashtable is a synchronized class(by default, thread-safe).

HashMap is not a synchronized class(by default, not thread safe).

3. Hashtable does not allow null key or null value.

HashMap allows one null key, multiple null values.

Fail-fast and Fail-safe Iterator:

--------------------------------

* Fail-fast and Fail-safe are the behaviours of an iterator.
* Fail-fast and Fail-safe indicates how an iterator behaves when any structural changes are made on the collection object, while iterator is iterating that object.
* Fail-fast means, while iterating a collection object, if any changes are made to that object by adding/removing an element then the iterator throws ConcurrentModificationException.
* Fail-safe means, while iterating a collection object, if any changes are made to that object by adding/removing an element then the iterator doesn’t throw any exception.

Fail-fast code:

ArrayList<Integer> arrList = new ArrayList<>();

arrList.add(12);

arrList.add(32);

arrList.add(29);

arrList.add(90);

arrList.add(57);

Iterator<Integer> itr = arrList.iterator();

while(itr.hasNext())

{

int k = itr.next();

System.out.println(k);

arrList.remove(3);

}

Fail-safe code:

ConcurrentHashMap<Integer, Integer> hm = **new** ConcurrentHashMap<>();

hm.put(1, 100);

hm.put(2, 200);

hm.put(3, 300);

Set<Entry<Integer,Integer>> entries = hm.entrySet();

Iterator<Entry<Integer, Integer>> itr = entries.iterator();

**while**(itr.hasNext()) {

Entry<Integer,Integer> entry = itr.next();

System.***out***.println(entry.getKey() + " -----> " +

entry.getValue());

hm.put(4, 400);

}

Q) What is WeakHashMap and IdentityHashMap?

A) [write your answer here]

Java 8 Features

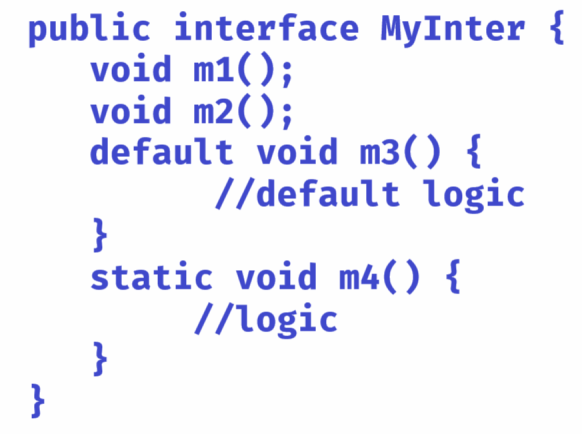
============================================

Functional interface:

------------------

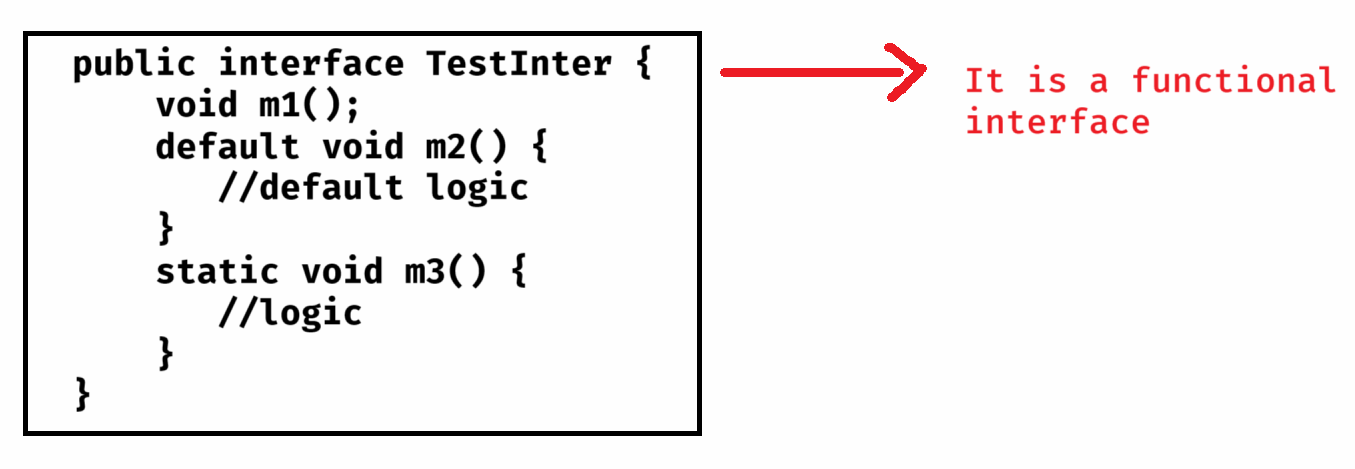
* Before Java8 version, we can only create abstract methods in an interface.
* From Java8 version, we can create abstract methods, default methods and static methods also in the interface.
* a default method can be created in the interface using default keyword and static methods can be created by using static keyword.

ex:

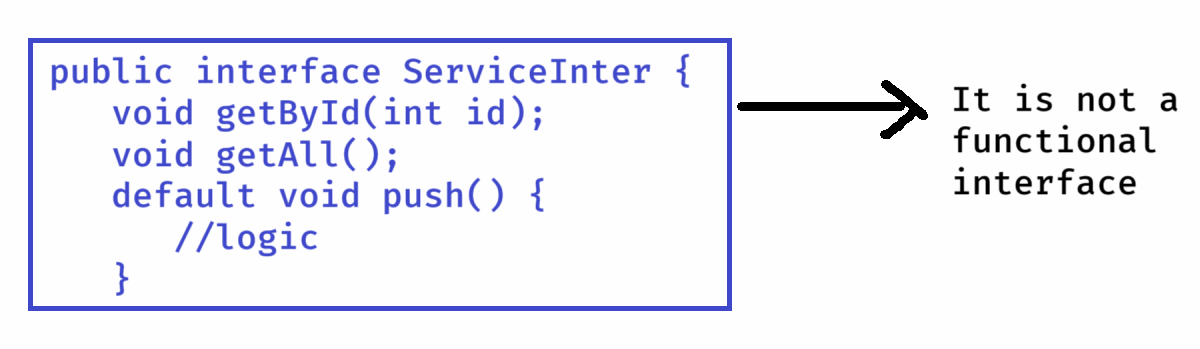


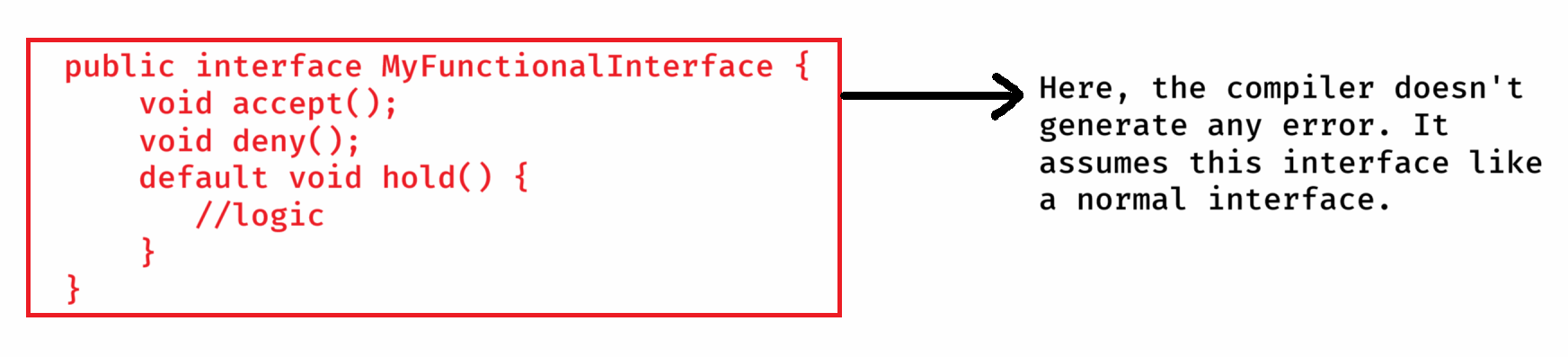
* A functional interface can contain only one abstract method. But it can contain multiple default methods and static methods also.

ex:

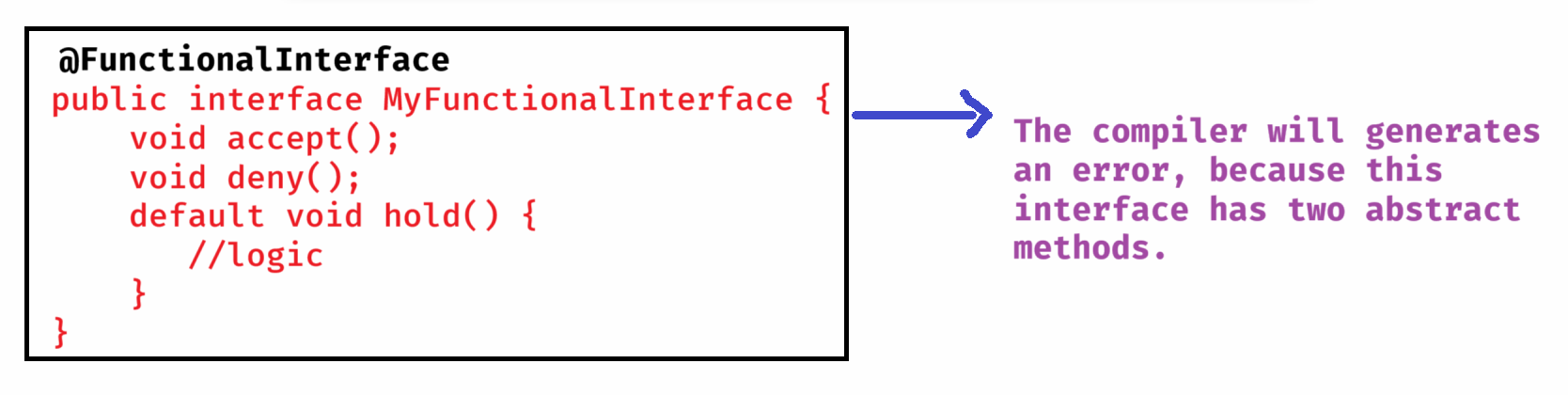


ex:

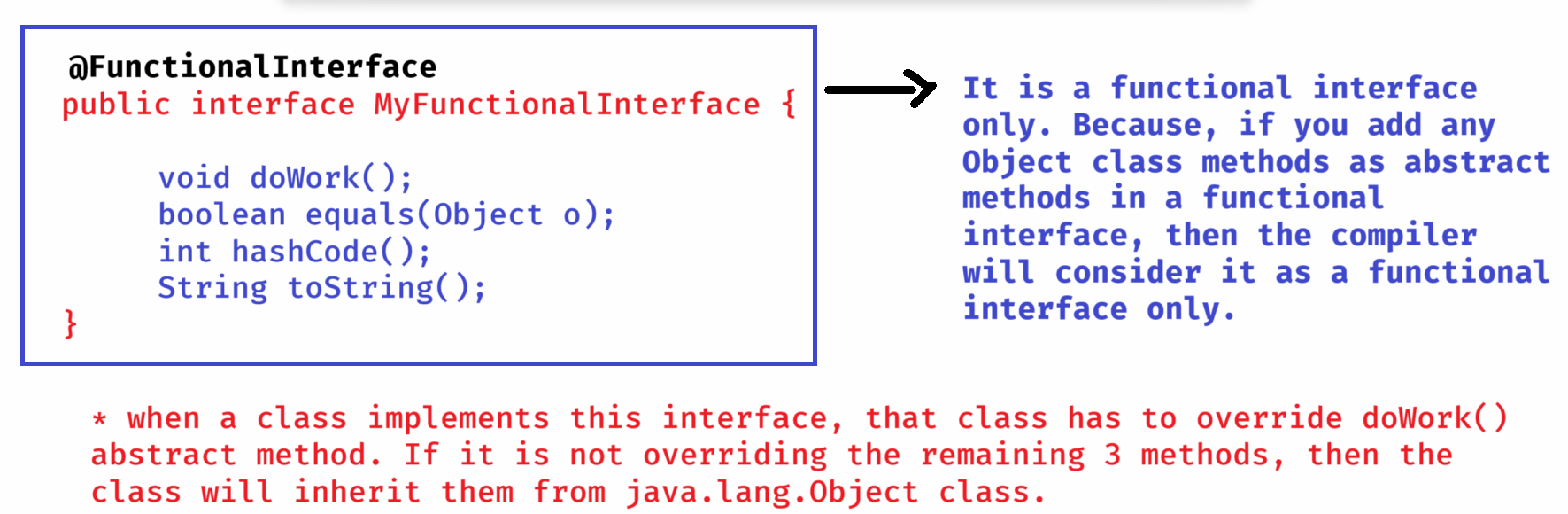




* @FunctinalInterface annotation is provided to tell the compiler that check whether an interface is a functional interface or not.
* The compiler will generate an error, if the interface contains more than one abstract method.



ex:



List of pre-defined functional interfaces:

----------------------------------------

interface name method name

---------------- -------------

1. Runnable interface void run();

2. Comparable<T> interface int compareTo(T t);

3. Comparator<T> interface int compare(T o1, T o2);

4. Predicate<T> interface boolean test(T t);

5. Supplier<T> interface T get();

6. Function<T> interface R apply(T t);

7. Consumer<T> interface void accept(T t);

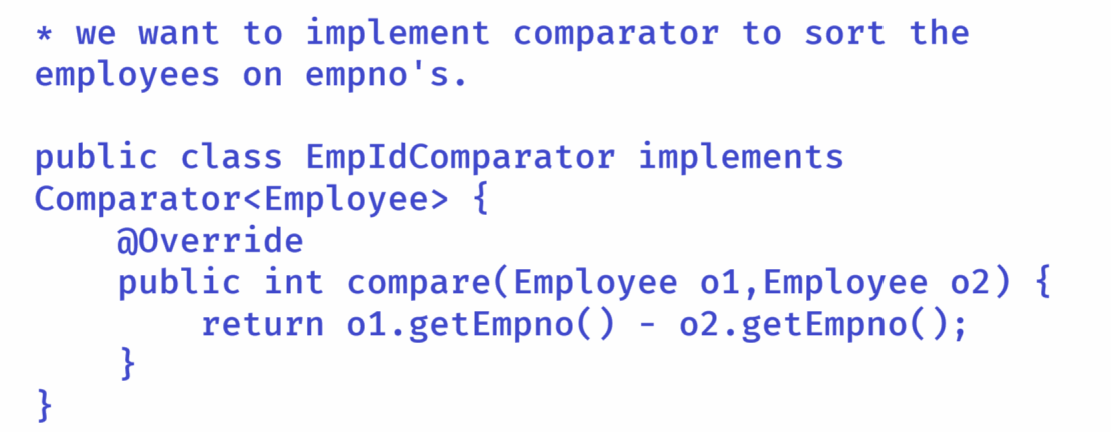
**Lambda expressions:**

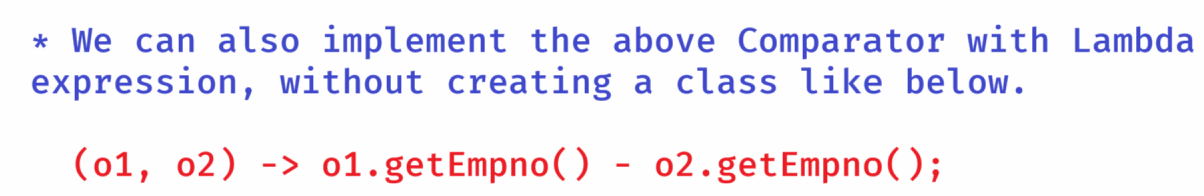
**-----------------**

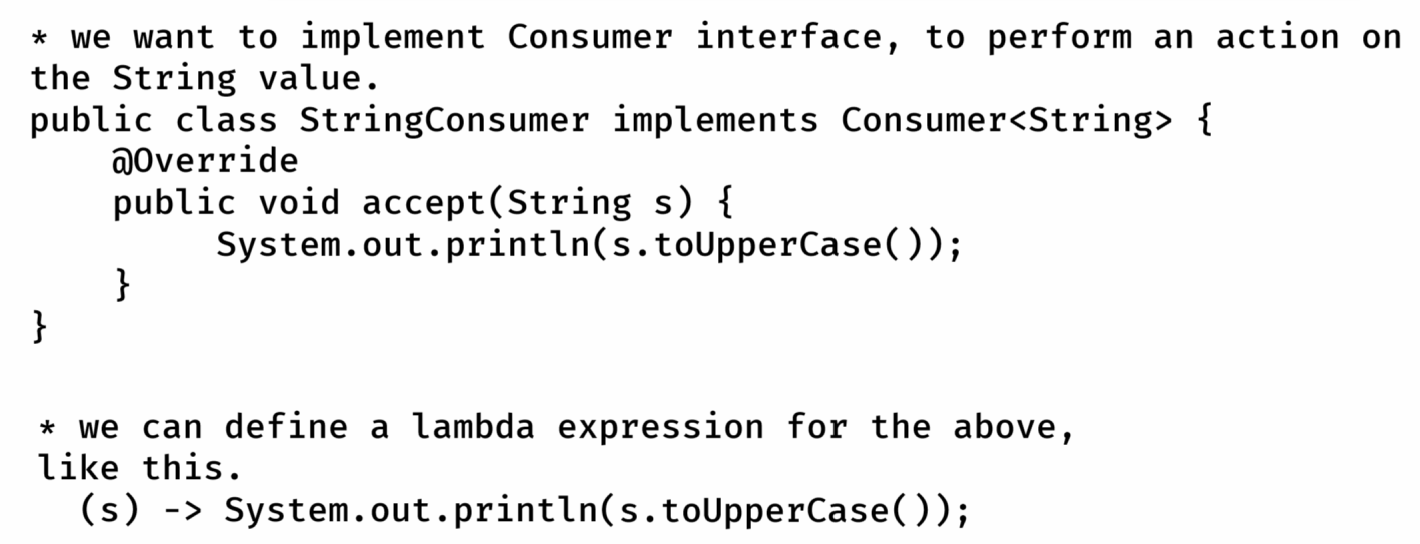
* **Lambda expressions are the shortcut way to provide an implementation for the functional interfaces, by without creating an implementation class.**
* **syntax:**

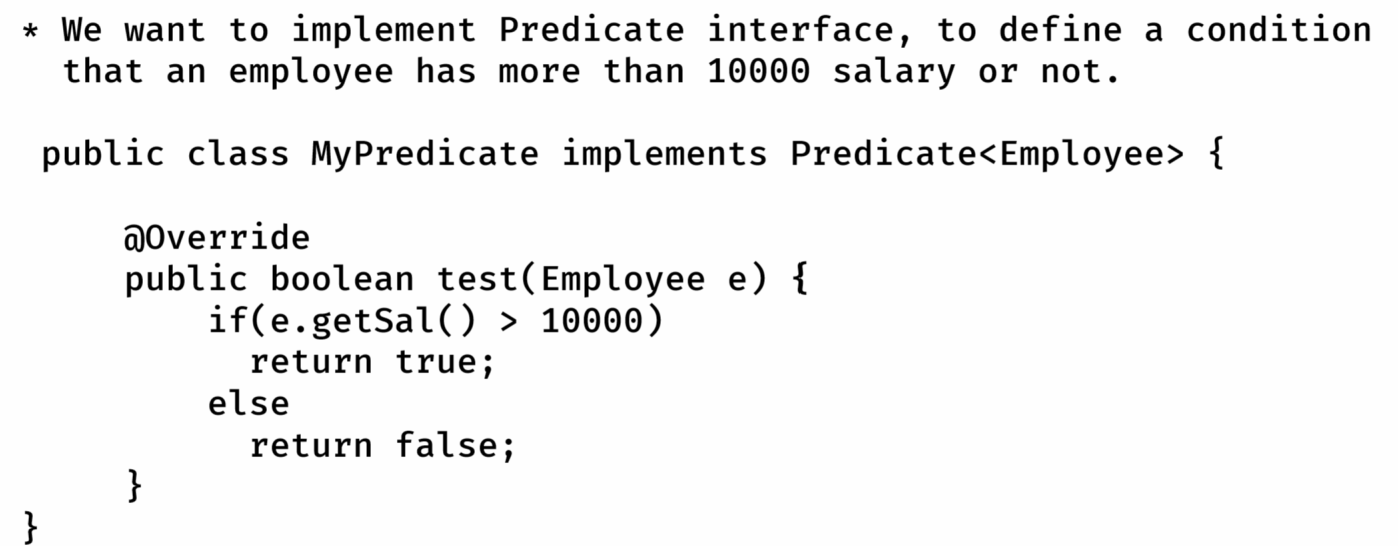
**(arguments) -> body;**

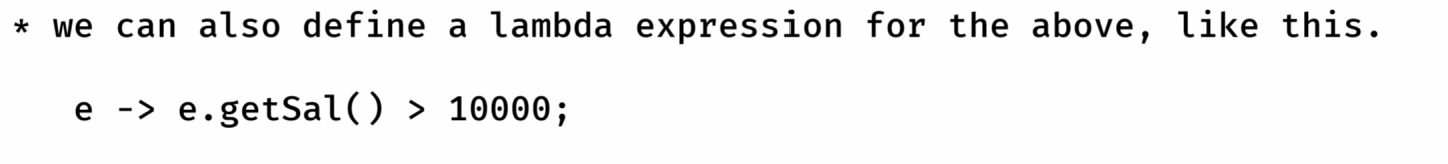
* **The arguments should match with the input arguments of the abstract method.**
* **The arguments datatypes are optional.**
* **If zero arguments or multiple arguments then paranthesis is mandatory. For single argument paranthesis is optional.**
* **If body contains a single statement then curly braces are optional.**
* **The single statement can return the value, without return keyword.**

****









Stream api

---------

* collections are used to store the group of elements and also we can perform some processing operations by iterating the collection object.
* If collection object has more elements, then will take more time to process them and decreases the appln performance.
* So, In Java8, to perform processing operations on collection objects, Stream interface is provided with a set of operations(methods). This is called Stream api.
* we can create a Stream object like below.

1. Stream stream = arrayList.stream();

2. Stream stream = Arrays.stream(array);

3. Stream stream = Stream.of(10, 30, 12, 56, 49);

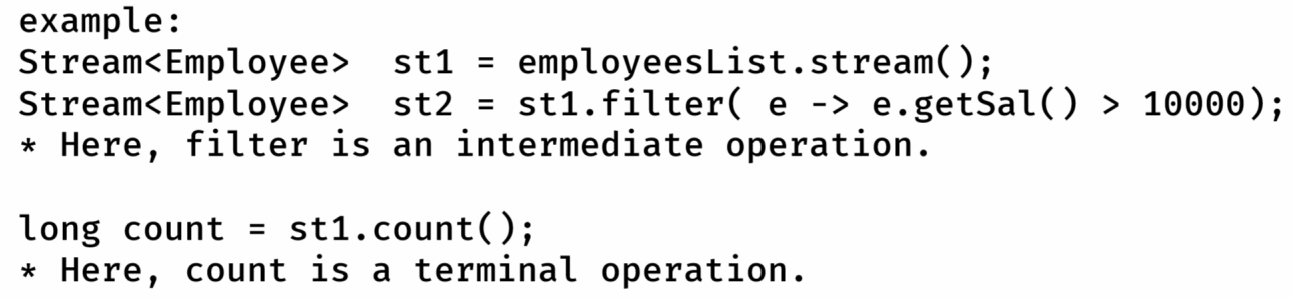
\* stream operations are divided into 2 categories.

1. intermediate operations.

2. terminal operations.

\* intermediate operations will transform one stream to another stream.

\* terminal operations doesn’t create another stream. They provide result.



ex1:

I have a list of employees and I want to print the employees with salary > 10000.

Ans:

Stream<Employee> s1 = employees.stream();

Stream<Employee> s2 = s1.filter(e -> e.getSal() > 10000);

s2.forEach( e -> System.out.println(e));

(or)

employees.stream()

.filter(e -> e.getSal() > 10000)

.forEach(e -> System.out.println(e));

Note: filter() method has Predicate as input argument

and forEach() method has Consumer as input argument.

ex2:

I want to count the number of employees with salary > 50000

Ans:

long count = employees.stream()

.filter(e -> e.getSal() > 10000)

.count();

ex3:

I want to find the employee with highest salary

Ans:

Optional<Employee> opt = employees.stream()

.sorted( (e1,e2) -> {

if(e1.getSal() < e2.getSal())

return 1;

else if(e1.getSal() > e2.getSal())

return -1;

else

return 0;

}

)

.findFirst();

opt.ifPresent( e -> System.out.println(e));

Note:

sorted() method has Comparator as input argument and

ifPresent() method has Consumer as input argument

ex4:

I want to display only the names of the employees with

salary > 5000

Ans:

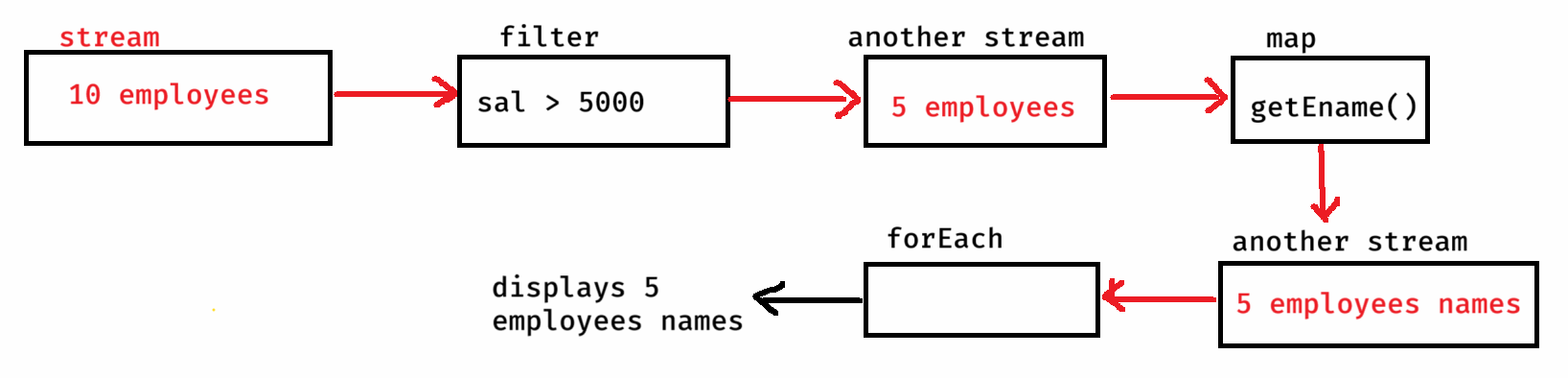
employees.stream()

.filter( e -> e.getSal() > 5000)

.map( e -> e.getEname() )

.forEach( name -> System.out.println(name));

Note: map() method has Function as an input argument.



ex5: I want to display first 5 highest salaried employees.

Ans:

employees.stream()

.sorted( (e1,e2) -> {

if(e1.getSal()<e2.getSal())

return 1;

else if(e1.getSal()>e2.getSal())

return -1;

else

return 0;

}

)

.limit(5)

.forEach(e -> System.out.println(e));

ex6: I have a list of integers, find sum of the integers.

List<Integer> lst = Arrays.asList(2, 5, 3, 8, 0);

Ans:

int sum = lst.stream()

.collect(Collectors.summingInt( i -> i.intValue());

Note:

collect() methods has Collector as input argument

Collector is an interface, and to create an object of type

Collector, we have Collectors class with static methods.

ex7:I want to group the employees by their department numbers.

Ans:

Map<Integer, List<Employee>> m = employees.stream()

.collect(Collectors.groupingBy(e->e.getDeptno());

parallelStream():



* If the source of the data (may be a collection object or an array) has huge elements, like 10 million elements, then a stream can process them, but it takes more time.
* So, we can use parallel stream for processing faster.
* parallel stream will divide the elements into data chunks and executes the operations on each chunk at different cpu core.
* After processing, it will join the results from each cpu core and returns the final output.
* So, parallel stream can utilize the full potential of the cpu.
* Just, we need to call parallelStream() method in place of stream() method.

ex:

List<Employee> lst2 = lst1.parallelStream()

.filter(e -> e.getSal() > 50000)

.collect(Collectors.toList());

Optional<T> class:

Optional<T> class:

----------------

* If an object has null value, and if you are calling method of that object then NullPointerException will be thrown.
* In a project, we have more objects, so the most frequently occrurred exception is NullPointerException.
* To avoid the NullPointerException, you have add the null checks.

ex:

public class Service {

public Employee getEmpById(int empid){

//reads an employee from the database

//if does not exist, then returns null

}

}

public class Controller {

public void show() {

Service s = new Service();

Employee e = s.getEmpById(101);

S.o.p(e.getEname());

S.o.p(e.getSal());

}

}

* In the above example, if 101 employee does not exist then

getEmpById() method returns null.

* In show() method, the variable e stores null and when you call e.getEname(), NullPointerException will be thrown.
* So, we have to modify the Controller class code like below.

public class Controller {

public void show() {

Service s = new Service();

Employee e = s.getEmpById(101);

if(e != null) { // null check

S.o.p(e.getEname());

S.o.p(e.getSal());

}

}

}

* If more null checks are added in the code, then the code becomes complicated.
* To reduce the null checks and also to avoid NullPointerException, Java 8 version has provided Optional<T> class.
* If a method has to return an object, then first has to store that object into Optional class object and then the method should return the Optional class object.
* Optional class object is a container object, it can store another object inside it.

creating Optional<T> class object:

1. Optional<Employee> opt = Optional.empty();

. creates empty Optional object.

2. Optional<Employee> opt = Optional.of(e1);

. If e1 is not null, then Optional object is created and

inside it e1 object will be stored.

. If e1 is null then NullPointerException will be thrown.

3. Optional<Employee> opt = Optional.ofNullable(e1);

. If e1 is not null, then Optional object is created and

inside it e1 object will be stored.

. If e1 is null then empty Optional object is created.

Methods of Optional<T> class:

--------------------------

1. isPresent() : If value is present in the Optional object then returns true. Otherwise, returns false.

2. get() : Returns the value present in the Optional object.

ex:

Optional<String> opt = Optional.ofNullable(“Hello World”);

if(opt.isPresent()) {

String str = opt.get();

System.out.println(str);

}

3. ifPresent(Consumer<T> consumer) : Executes the given action, if the value is present in the Optional object. Otherwise, do’s nothing.

. This method accepts Consumer object as parameter.

. Consumer is a functional interface, so we can pass lambda expression.

ex:

Optional<String> opt = Optional.ofNullable(“Hello World”);

opt.ifPresent( str -> System.out.println(str) );

4. orElse(T another): if the Optional contains value, it returns that value. If it is empty then returns the another value provided as parameter.

ex1:

Optional<String> opt = Optional.ofNullable(“Hello world”);

String result = opt.orElse(“Bye world”);

System.out.println(result); //output: Hello world

ex2:

Optional<String> opt = Optional.empty();

String result = opt.orElse(“Bye world”);

System.out.println(result); //output: Bye world

5. orElseThrow(): If the Optional contains a value, then returns that value. If it is empty, then throws NoSuchElementException.

ex1:

Optional<String> opt = Optional.ofNullable(“Hello world”);

String result = opt.orElseThrow();

System.out.println(result); //output: Hello world

ex2:

Optional<String> opt = Optional.empty();

String result = opt.orElseThrow(); // throws NoSuchElementException

Date/Time API

* Before Java8 version, we have java.util.Date and java.util.Calendar classes to work with data and time values.
* The problem with these classes is, they are mutable objects. So, in mulithreaded applications, if one thread changes the value, then it effects on other thread. This is a bug.
* The other issue is, the year starts from 1900, not from 0.
* For example, if I create,
* Date d = new Date(120, 0, 21);
* then, it represents the year 2020, not 120.
* So, Java 8 has provided, new classes to work with Date and Time values in java.time package.
* LocalDate class
* LocalTime class
* LocalDateTime class
* Period class

For example,

LocalDate today = LocalDate.now(); //The LocalDate object is

created with current system date.

LocalDate expDate = LocalDate.of(2025, 11, 30);

* LocalDate, LocalTime, and LocalDateTime classes objects are immutable objects. So, if you make any changes then new objects are created with results.

ex:

LocalDate date1 = LocalDate.of(2025, 1, 31);

LocalDate date2 = date1.plusDays(3);

S.o.p(date1); //output: 2025-01-31

S.o.p(date2); //output: 2025-02-3

Note: you have other methods like,

plusMonths(), plusYears(), plusWeeks(), minusDays(), minusMonths(), minusYears(), minusWeeks().

LocalTime currentTime = LocalTime.now(); //system time

LocalTime endTime = LocalTime.of(18, 30, 45);

* Suppose, if you want to find the difference between two dates in terms of days then,
* LocalDate d1 = LocalDate.*of*(2024, 8, 15);
* LocalDate d2 = LocalDate.*of*(2025, 2, 18);
* **long** days = ChronoUnit.***DAYS***.between(d1, d2);
* System.***out***.println("difference in days : " + days);
* suppose, if you want to find the difference between two times, interm of minutes then,

LocalTime t1 = LocalTime.*of*(9, 35, 40);

LocalTime t2 = LocalTime.*of*(19, 15, 39);

**long** minutes = ChronoUnit.***MINUTES***.between(t1, t2);

System.***out***.println("difference in minutes : " + minutes);

* Period class compares the given two dates based on the values of dates, months and years. It means, it will not provide the as the difference in total no of days or total months or total years.

LocalDate d1 = LocalDate.of(2024,02, 20);

LocalDate d2 = LocalDate.now();

Period p = Period.between(d1, d2);

S.o.println(p.getDays()); //output: 0

S.o.println(p.getMonths()); //output: 0

S.o.println(p.getYears()); //output: 1