Spring Core Module

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

* Today’s enterprise applications contains four layers.

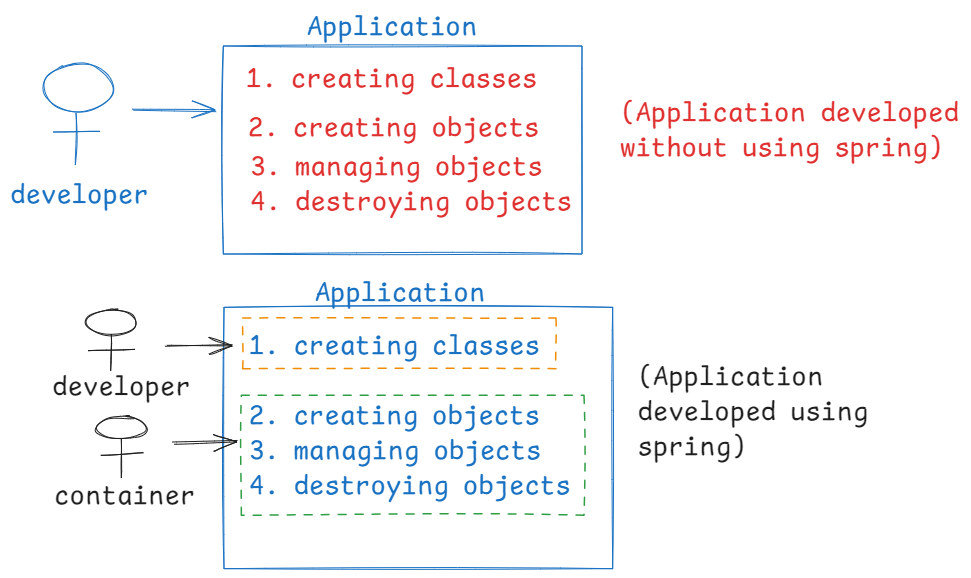
1. Presentation layer
2. API Layer
3. Business layer
4. Data Access layer

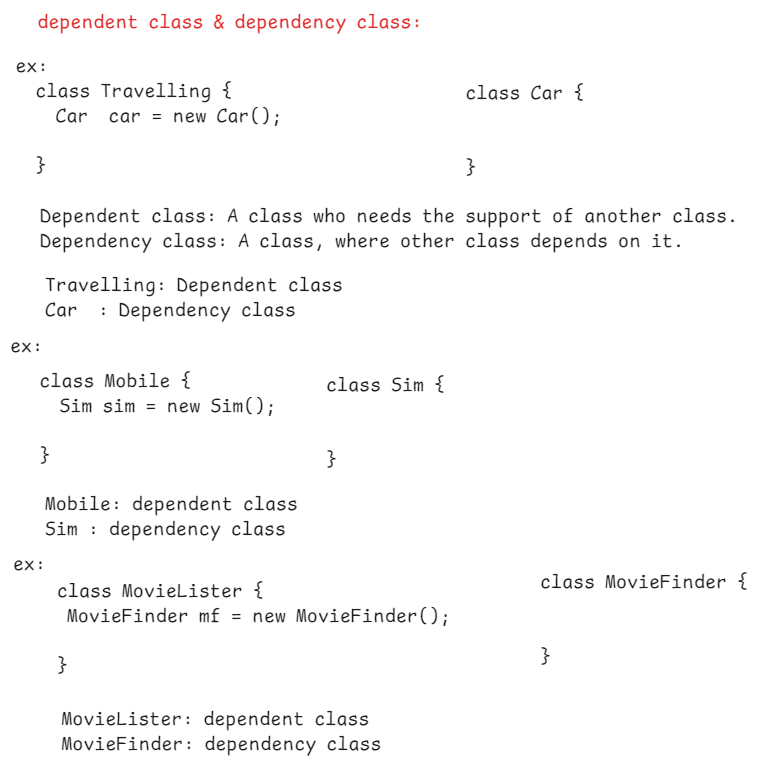
* Struts Framework – used for developing Presentation layer
* Web services – used for developing API layer
* Spring Framework – used for developing Business layer
* Hibernate Framework – used for developing Data Access layer.
* Spring framework is intially used for developing Business layer.
* Later, spring framework is expanded for creating all the layers like presentation layer, api layer, business layer and the Data access layer.
* Nowadays, Angular/React JS are used for developing presentation layer.(Front-end development).
* Spring framework is used for developing api layer, business layer and the data access layer. (Back-end development).

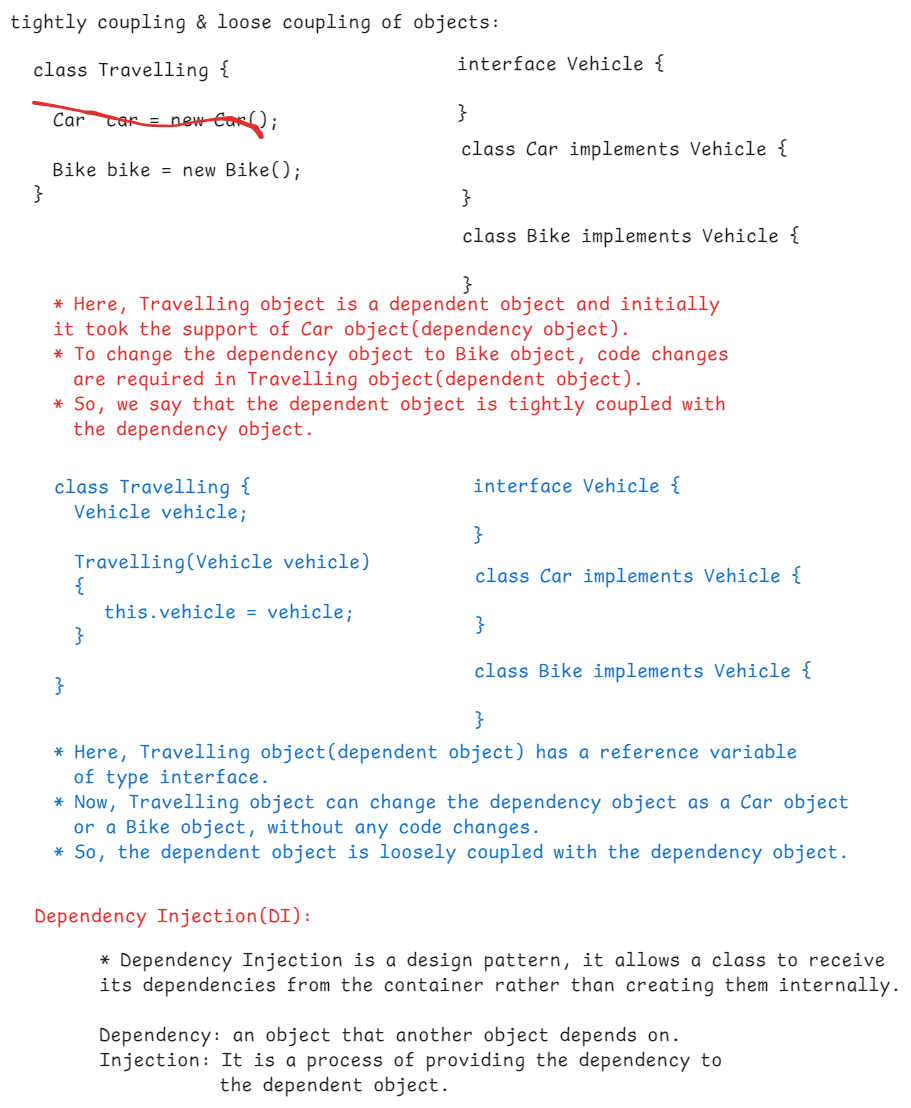
Inversion of Control(IoC):

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

* When a developer is developing an application without using spring framework, the developer is resposible for creating classes, creating objects, managing them and destroying objects.
* So, a developer has more burden.
* When a developer is developing an application with spring framework, the developer is only responsible for creating classes. The spring container will take care of creating objects, managing them and destroying the objects.
* This concept is called called Inversion of Control.
* Inversion of Control definition is, inverting some resposibilities of a developer to the another person like container.







ex:

class Client {

Service service = new Service();

}

\*Here, Client is the dependent class, and Service is the dependency class.

\* The dependent class is internally creating dependency class object. So, it is not called

dependency injection.

ex:

class Client {

Service service;

Client() {

service = new Service();

}

}

\*Here, the dependent class is internally creating dependency class object, in its constructor. So, it is not a dependency injection.

ex:

class Client {

Service service;

Client(Service service) {

this.service = service;

}

}

\*Here, the dependent class is not internally creating the dependency class object. It is receiving the dependency object through constructor from the container. This is called constructor injection.

ex:

class Client {

Service service;

public void setService(Service service)

{

this.service = service;

}

}

\*Here, the dependent class is not internally creating dependency class object. But it is receiving the dependency object through setter method from the container. This is called setter injection.

Types of dependency injection:

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

1. constructor injection: Dependencies are provided through constructor by the container.
2. setter injection: Dependencies are provided through setter methods by the container.
3. interface injection: Depenedencies are provided through an interface method by the container.

configuration in spring:

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

* In spring framework, the spring container creates objects, injects the dependencies, performs life cycle management.
* To do all these tasks, the spring container needs some information about the spring bean classes. This information, we call as metadata.
* The configuration is a way of providing this metadata to the spring container.
* The configuratin can be done in 3 ways.

1. xml file
2. annotations
3. Java configuration class

* A sample xml file looks like below.

beans.xml

---------

<beans>

<bean id=”c” class=”pack1.Client”>

…………………………

</bean>

<bean id=”s” class=”pack1.Service”>

………………………

</bean>

</beans>

* we don’t use xml based configuration in applications now. We use annotations and Java configuration classes.

spring container:

* A spring container is a predefined class.
* spring has provided 2 types of containers.

1. BeanFactory container
2. ApplicationContext container

* BeanFactory and ApplicationContext are interfaces in spring.
* In an application, you need to construct either BeanFactory or ApplicationContext object.
* BeanFactory is the basic spring container, ApplicationContext is the advanced spring container.
* The implementation class of BeanFactory is XmlBeanFactory class.
* The implementation class of ApplicationContext are, ClassPathXmlApplicationContext class and AnnotationConfigApplicationContext class.

Maven

-----

what is build process?

1. adding the jar(java archive) files to

the classpath.

1. compiling source code.
2. compiling the test cases
3. executing the test cases
4. bundle the compiled classes into distributable format(jar/war).
5. perform code quality checks
6. store the artifact(jar file) into a repository.

* Before build tools, the above steps are done manually by the developers.
* If any code changes are happended, then again the developers has to repeat the above steps again.
* To automate the build process of java projects, the tools like maven and gradle are introduced.

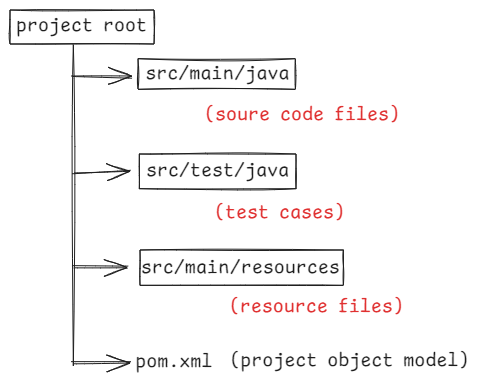
repositories:

1. central repository
2. local repository

* Maven is a tool from Apache.
* Maven maintains all the java projects related jar files(like spring jar files or hibernate jar files or JPA jar files,etc..) are maintained at central repository.
* when you are developing a Java project, the maven downloads the required jar files from the central repository and stores them in the local repository.
* local repository is created in every developer’s machine.

Maven project structure:

* Maven enforces a standarized directory structure for projects, ensures the consistency across different projects.



archetype: A template for creating a maven project structure.

groupId: a unique id to identify the group of projects developed by an org.

artifactId : name of a project

version: 0.0.1-SNAPSHOT

adding dependencies(jars):

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.7</version>

</dependency>

maven commands for build process:

mvn compile : to compile the source code

mvn test : to compile and execute unit tests

mvn package : to bundle our application into a JAR file or WAR file.

mvn install : to install the package(jar/war) into local repository.

mvn clean : to remove the files generated in the previous build.

Maven setup:

* IDE software’s like Eclipse/STS comes with Maven as in built. So we don’t require any setup separately.
* But if you want to execute a maven project’s build process from command line then you have to do the maven setup in your machine.
* 1. visit maven.apache.org
* 2. download apache-maven-3.9.8-bin.zip
* 3. extact the zip file
* 4. add the maven’n bin folder location to the path variable.

(C:\Users\WINDOWS\Downloads\apache-maven-3.9.8-bin\apache-maven-3.9.8\bin)

5.open a command prompt, execute the command

> mvn -version

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

Note: The maven local repository is created at C:\Users\WINDOWS\.m2\repository

Spring Tool Suite (STS) :

1. visit spring.io/tools
2. download for windows
3. an executable jar file spring-tool-suite-xxxxxxx.jar

is downloaded

1. open command prompt, move to downloads folder
2. execute the below command to install STS

* java -jar spring-tool-suite-xxxxxxxxxxxx.jar

1. A folder sts-4.4.20 is created.
2. You can start SpringToolSuite4 application.

Basic annotations to start the first spring application:

@Component: It declares that a class as a spring component.

@Configuration: It declares that a class as a Java configuration class.

@ComponentScan: Component scan is a facility in spring, where spring will identify the bean components and registers them in the spring container.

if you specify the base package then spring will automatically scan the base package and also its sub packages, to identify the components.

* To start with spring application, at least we need to create 3 files.

1. spring bean class
2. configuraiton class
3. main class

steps to create the project:

1. start STS
2. File -> New -> Maven Project
3. Next -> type maven-archetype-quickstart in filter -> choose maven-archetype-quickstart(last one) -> next
4. enter the parameters,

Group id : in.ashokit

Artifact id : 1-Application

version: 0.0.1-SNAPSHOT

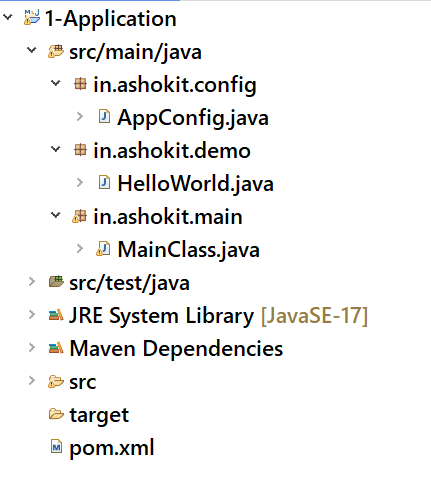
package: in.ashokit.demo

finish

1. Type Y in the console
2. project structure is created.

* open pom.xml file and add the following dependency tag, under <dependecies> tag.
* <dependency>
* <groupId>org.springframework</groupId>
* <artifactId>spring-context</artifactId>
* <version>5.3.37</version>
* </dependency>

Note: delete App.java file created.



HelloWorld.java

@Component

**public** **class** HelloWorld {

**public** String sayHello() {

**return** "Hello World!";

}

}

AppConfig.java

@Configuration

@ComponentScan( basePackages = "in.ashokit.demo")

**public** **class** AppConfig {

}

MainClass.java

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

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

//start the container

ApplicationContext context = **new** AnnotationConfigApplicationContext(AppConfig.**class**);

//get the bean from the container

HelloWorld helloWorld = context.getBean(HelloWorld.**class**);

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

}

}

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

@Value & @Autowired:

* A spring component, can have value type dependencies or object type dependencies.
* @Value injects a value.
* @Autowired injects an object.

@Component

public class ClassA {

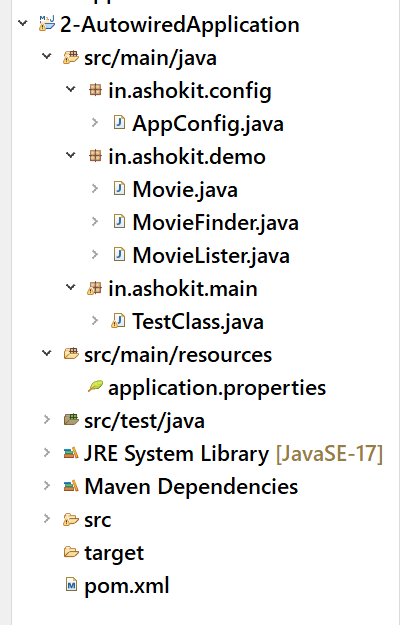
@Value(“10”)

private int x;

@Autowired

private ClassB classB;

}



Movie.java

**package** in.ashokit.demo;

//POJO class

**public** **class** Movie {

**private** **final** **int** releasedYear;

**private** **final** String movieName;

**public** Movie(**int** releasedYear, String movieName) {

**this**.releasedYear = releasedYear;

**this**.movieName = movieName;

}

**public** **int** getReleasedYear() {

**return** releasedYear;

}

**public** String getMovieName() {

**return** movieName;

}

}

MovieFinder.java

package in.ashokit.demo;

import java.util.Arrays;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.springframework.stereotype.Component;

@Component

public class MovieFinder { //dependency class

private Map<Integer, List<Movie>> moviesMap;

public MovieFinder() {

moviesMap = new HashMap<>();

moviesMap.put(2022, Arrays.asList(new Movie(2022, "RRR"), new Movie(2022, "Drishyam2"), new Movie(2022, "Pushpa")));

moviesMap.put(2023, Arrays.asList(new Movie(2023, "Pathaan"), new Movie(2023, "Jawan"), new Movie(2023, "Jailer")));

moviesMap.put(2024, Arrays.asList(new Movie(2024, "Kalki"), new Movie(2024, "Premikulu")));

}

public List<Movie> findMoviesByYear(int releasedYear) {

return moviesMap.get(releasedYear);

}

}

MovieLister.java

**package** in.ashokit.demo;

**import** java.util.List;

**import** java.util.function.Consumer;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** MovieLister { //dependent class

@Value("${movie.releasedYear}")

**private** **int** releasedYear;

@Autowired

**private** MovieFinder movieFinder;

**public** **void** printMovies() {

List<Movie> moviesList = movieFinder.findMoviesByYear(releasedYear);

Consumer<Movie> consumer = movie -> System.***out***.println(movie.getReleasedYear() + " - " + movie.getMovieName());

moviesList.forEach( consumer );

}

}

AppConfig.java

**package** in.ashokit.config;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.PropertySource;

@Configuration

@ComponentScan( basePackages = "in.ashokit")

@PropertySource("application.properties")

**public** **class** AppConfig {

}

TestClass.java

**package** in.ashokit.main;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** in.ashokit.config.AppConfig;

**import** in.ashokit.demo.MovieLister;

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

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

ApplicationContext context = **new** AnnotationConfigApplicationContext(AppConfig.**class**);

MovieLister movieLister = context.getBean(MovieLister.**class**);

movieLister.printMovies();

}

}

application.properties

movie.releasedYear=2023