**Spring**

* Webapps
* Microservices
* Serverless
* Cloud
* Event Driven

Spring Home - https://spring.io/

Why Spring - <https://spring.io/why-spring>

Spring Projects - <https://spring.io/projects>

**Graphical user interface, text, application

Description automatically generated**

**Dependency Inversion:**

**Graphical user interface, text, application, email

Description automatically generated**

* If we want to change the sorting to quick sort, we would have to change code inside the class ComplexAlgorithmImpl. This is called tight coupling.
* **Tight coupling** means classes and objects are dependent on one another. In general, tight coupling is usually not good because it reduces the flexibility and re-usability of the code while **Loose coupling** means reducing the dependencies of a class that uses the different class directly.
* We can make this to loose coupling using Interface
* A screenshot of a computer program

  Description automatically generated
* In complex algorithm class we were initially specifying which algorithm to use, but now we have changed it.
* Loose Coupling: Coupling refers to the dependency of one object type on another, if two objects are completely independent of each other and the changes done in one doesn’t affect the other both are said to be loosely coupled. We can achieve loose coupling in Java using interfaces.
* Loose coupling is a design goal to reduce the inter-dependencies between components of a system with the goal of reducing the risk that changes in one component will require changes in any other component.

**Inversion Of Control**

* Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.
* In contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.
* The advantages of this architecture are:
  + decoupling the execution of a task from its implementation
  + making it easier to switch between different implementations
  + greater modularity of a program
  + greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts
* We can achieve Inversion of Control through various mechanisms such as: Strategy design pattern, Service Locator pattern, Factory pattern, and **Dependency Injection (DI).**

**Dependency Injection:**

* Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object’s dependencies.
* Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.
* Here’s how we would create an object dependency in traditional programming:

public class Store {

private Item item;

public Store() {

item = new ItemImpl1();

}

}

* In the example above, we need to instantiate an implementation of the Item interface within the Store class itself.
* By using DI, we can rewrite the example without specifying the implementation of the Item that we want:

public class Store {

private Item item;

public Store(Item item) {

this.item = item;

}

}

A screenshot of a computer

Description automatically generated

* The user of the complex algorithm can decide which algorithm to use. Here we can see inversion of responsibilities. The Complex algorithm class is no longer responsible for choosing the algorithm and we will leave it to the user.
* We can give it any algorithm we want and it will work with it.
* The complex algorithm impl is decoupled from which sorting algorithm it needs to work on.
* This is called dependency inversion.
* Concept in which all the classes explicitly state their dependencies and create abstract classes or interfaces around them so that they can work with any implementation of that interfaces/abstract classes – Loose coupling
* **Advantage:** Spring makes use of this concept
* Here we can add annotation to the classes and let spring know we want to create these objects
* Spring will create instance of quick sort algorithm and instance of complex algorithm using the instance of quick sort algot
* The process in which spring identifies:
  + The beans
  + The dependencies
  + Populates dependencies into beans
  + Is called dependency injection

**Creating a simple spring project**

1. **Download some Jars**
2. **Create student class**  
   **package** com.javatrainings;

**public** **class** Student {

**private** String name;

**private** String id;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** **void** displayInfo(){

System.***out***.println("Hello: "+name + " your id is :"+id);

}

}

1. **Create Test class**

package com.javatrainings;

import org.springframework.beans.factory.BeanFactory;

import org.springframework.beans.factory.xml.XmlBeanFactory;

import org.springframework.core.io.ClassPathResource;

import org.springframework.core.io.Resource;

public class Test {

public static void main(String[] args) {

Resource resource=new ClassPathResource("applicationContext.xml");

BeanFactory factory=new XmlBeanFactory(resource);

Student student=(Student) factory.getBean("bean2");

student.displayInfo();

Student student1=(Student) factory.getBean("bean1");

student1.displayInfo();

}

}

1. **Create applicationContext.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans

xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>

<bean id=*"bean1"* class=*"com.javatrainings.Student"*>

<property name=*"name"* value=*"Abhishek Birla"*></property>

<property name=*"id"* value=*"123"*></property>

</bean>

<bean id=*"bean2"* class=*"com.javatrainings.Student"*>

<property name=*"name"* value=*"Rohit "*></property>

<property name=*"id"* value=*"234"*></property>

</bean>

</beans>