Name:-Tosif Mansuri

Week_6_Assignment

//--------Creating BillCalculator Class

```
package com.mycompany.billCalculate;

public class BillCalculator {
    private int units;
    private double tax;
    private double unitPrice;

   public BillCalculator() {
    this(0,0.0d,0.0d);
    }

    public BillCalculator(int units,  double tax, double unitPrice) {
    this.units=units;
    this.tax=tax;
    this.unitPrice=unitPrice;
    }

      public double calculateBill() {
        return (unitPrice+(unitPrice*tax))*units;
    }
}
```

//------------Creating Spring.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="billCalculator"
class="com.mycompany.billCalculate.BillCalculator">
    <constructor-arg value="10">
    </constructor-arg>
    <constructor-arg value="0.18">
    </constructor-arg>
    <constructor-arg value="100">
    </constructor-arg>
</bean>
</beans>
```

```
//------Creating Ui App.java

package com.mycompany.UI;
```

```
import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.mycompany.billCalculate.BillCalculator;

public class App {

public static void main(String[] args) {

ApplicationContext context = new
ClassPathXmlApplicationContext("Spring.xml");

BillCalculator billCalculator = (BillCalculator)
context.getBean("billCalculator");

System.out.println("Total Bill to pay is
$"+billCalculator.calculateBill());

      }

}


//------------Testing

//---------Creating TestBillCalculator


package com.mycompany.Week6Assignment;

import junit.framework.TestCase;

public class TestBillCalculator extends TestCase {

      public void testCalculateBill() {
            assertEquals(1180.0,"1180.0");
      }

}


//---POM File


<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany</groupId>
  <artifactId>Week6Assignment</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Week6Assignment</name>
  <url>http://maven.apache.org</url>

   <dependencies>
<dependency>
<groupId>org.springframework</groupId>
```

```
<artifactId>spring-context</artifactId>
<version>5.3.18</version>
</dependency>
</dependencies>

<properties>
<maven.compiler.target>1.8</maven.compiler.target>
<maven.compiler.source>1.8</maven.compiler.source>
</properties>
</project>

//----------------------------------------------------------------Q1 End-----------------------------------------------


//-----------------------------------------Q.2-----------------------------------------------


//-------Creating billCalculate Class


package com.mycompany.billCalculate;

public class billCalculate {
      public double getcalculateBill() {
        int unitPrice = 100;
            double tax = 0.18;
            int units = 10;
            return (unitPrice+(unitPrice*tax))*units;
    }
}

//-------Creating BillCalculator Class

package com.mycompany.billCalculate;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Component;

public class BillCalculator {
    private int units;
    private double tax;
    private double unitPrice;

    public BillCalculator() {
    this(0,0.0d,0.0d);
    }

    public BillCalculator(int units, double tax, double unitPrice) {
    this.units=units;
    this.tax=tax;
    this.unitPrice=unitPrice;
    }
}

//--------Spring config

package com.mycompany.config;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

import com.mycompany.billCalculate.BillCalculator;
import com.mycompany.billCalculate.billCalculate;

@Configuration
@ComponentScan(basePackages = "com.mycompany.billCalculate")
public class SpringConfig {
	@Bean
	public BillCalculator getBillCalculator()
	{
		BillCalculator billCalculator=new BillCalculator();
		return billCalculator;
	}
	@Bean

	public billCalculate getbillCalculate()
	{
		billCalculate billcalculate=new billCalculate();
		return billcalculate;
	}
}

//----Creating UI App.java

package com.mycompany.UI;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.mycompany.billCalculate.BillCalculator;
import com.mycompany.billCalculate.billCalculate;
import com.mycompany.config.SpringConfig;

public class App {
	public static void main(String[] args) {
		ApplicationContext applicationContext = new
AnnotationConfigApplicationContext(SpringConfig.class);
	  billCalculate billcalculate = (billCalculate)
applicationContext.getBean(billCalculate.class);
		System.out.println("Total Bill to pay is
$"+billcalculate.getcalculateBill());

	}
}


//---Spring.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="billCalculator"
class="com.mycompany.billCalculate.BillCalculator">
```

```
    <constructor-arg value="10">
    </constructor-arg>
    <constructor-arg value="0.18">
    </constructor-arg>
    <constructor-arg value="100">
    </constructor-arg>
</bean>
</beans>



//-----Testing


package com.mycompany.Week6Assignment1;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestBillCalculator {

      @Test

      public void testCalculateBill() {

            assertEquals(1180,"calculateBill()");

      }

}


//---Pom File


<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany</groupId>
  <artifactId>Week6Assignment1</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Week6Assignment1</name>
  <url>http://maven.apache.org</url>

  <dependencies>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>5.3.18</version>
</dependency>
</dependencies>


<properties>
<maven.compiler.target>1.8</maven.compiler.target>
```

```
<maven.compiler.source>1.8</maven.compiler.source>
</properties>
</project>
```

Q2 End

Q3 Start

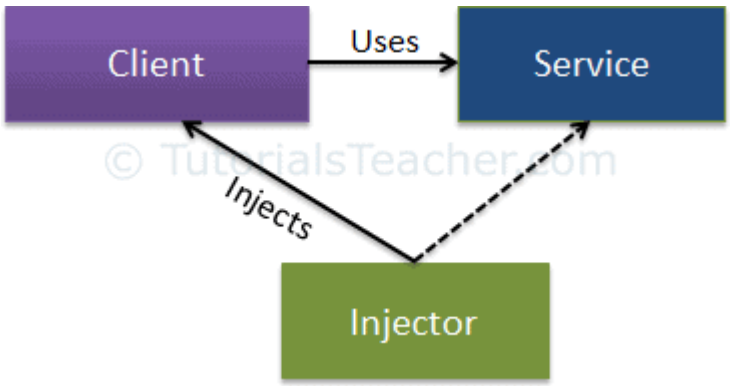
Dependency Injection:

Dependency Injection (DI) is a design pattern used to implement IoC. It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways. Using DI, we move the creation and binding of the dependent objects outside of the class that depends on them.

The Dependency Injection pattern involves 3 types of classes.

1. **Client Class:** The client class (dependent class) is a class which depends on the service class
2. **Service Class:** The service class (dependency) is a class that provides service to the client class.
3. **Injector Class:** The injector class injects the service class object into the client class

The following figure illustrates the relationship between these classes:



Dependency Injection

Types of Dependency Injection:

As you have seen above, the injector class injects the service (dependency) to the client (dependent). The injector class injects dependencies broadly in three ways: through a constructor, through a property, or through a method.

1.Constructor Injection

2.Property Injection

3.Method Injection

1. **Constructor Injection:** In the constructor injection, the injector supplies the service (dependency) through the client class constructor.

2. **Property Injection:** In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.

3. **Method Injection:** In this type of injection, the client class implements an interface which declares the method(s) to supply the dependency and theinjector uses this interface to supply the dependency to the client class.

Example of Constructor Injection:

```
<bean id="project" class="app.myspringapp.Project">
        <constructor-arg value="MIS"></constructor-arg>
        <constructor-arg value="San Diago"></constructor-arg>
</bean>
```

Inversion of Control:

Spring Ioc (Inversion of Control) Container is the core of Spring Framework. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection(DI) to manage the components that make up the application. It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion of Control.

There are 2 types of Ioc containers:

1.Bean Factory

2.Application Context

That means if you want to use an IoC container in spring whether we need to use a BeanFactory or ApplicationContext. The BeanFactory is the most basic version of IoC containers, and the ApplicationContext extends the features of BeanFactory. The followings are some of the main features of Spring IoC,

- Creating Object for us.
- Managing our objects.

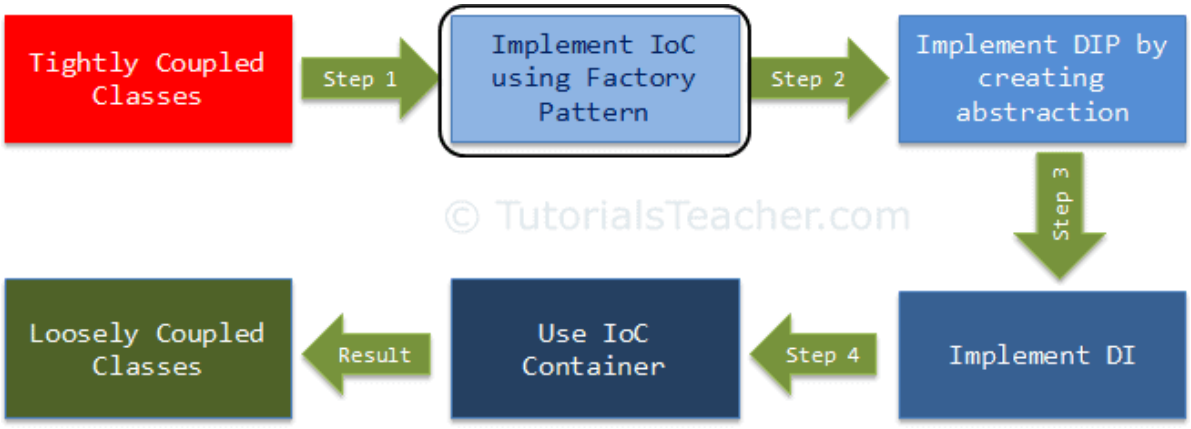
- Helping our application to be configurable.
- Managing dependencies.

Example of Ioc Containers:

```
public class Mobile {
    public static void main(String[] args) {
        // Using ApplicationContext tom implement Spring IoC

ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("beans.xml");

// Get the bean
Sim sim = applicationContext.getBean("sim", Sim.class);

// Calling the methods
sim.calling();
sim.data();
    }
}
```

And now if you want to use the Airtel sim so you have to change only inside the **beans.xml** file. The main method is going to be the same.

```
<bean id="sim" class="Airtel"></bean>
```



Steps of Ioc Containers