Spring Framework

Introduction

What is Spring Framework?

- Open Source framework for Java enterprise edition.
- For developing reliable and high-quality applications.
- Designed by Rod Johnson back in 2003.
- Become an alternative technology in Java for the EJB model.
- Create different kinds of applications using the spring framework.
- Spring is a lightweight framework which can be thought of as a framework of frameworks because it also offers support for various frameworks such as hibernate, struts, tapestry, and JSF.

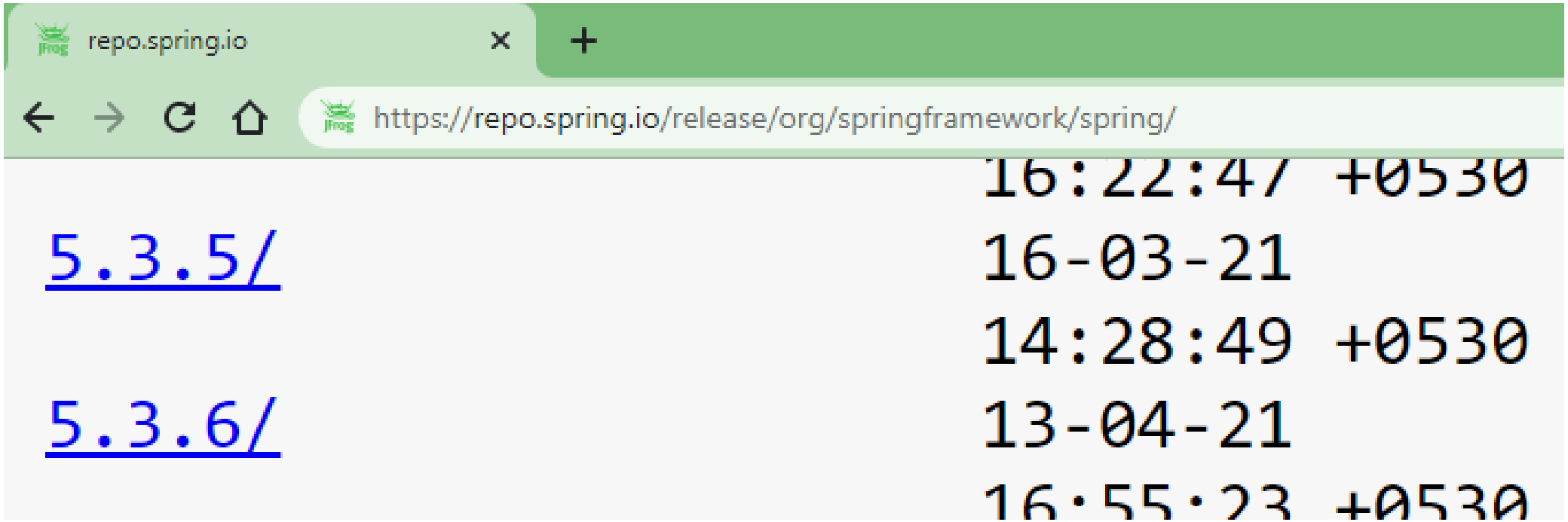
Framework of frameworks

Create a first spring project

- Create a new java project in Eclipse
- Add Spring jars

Lets add Spring Framework to project

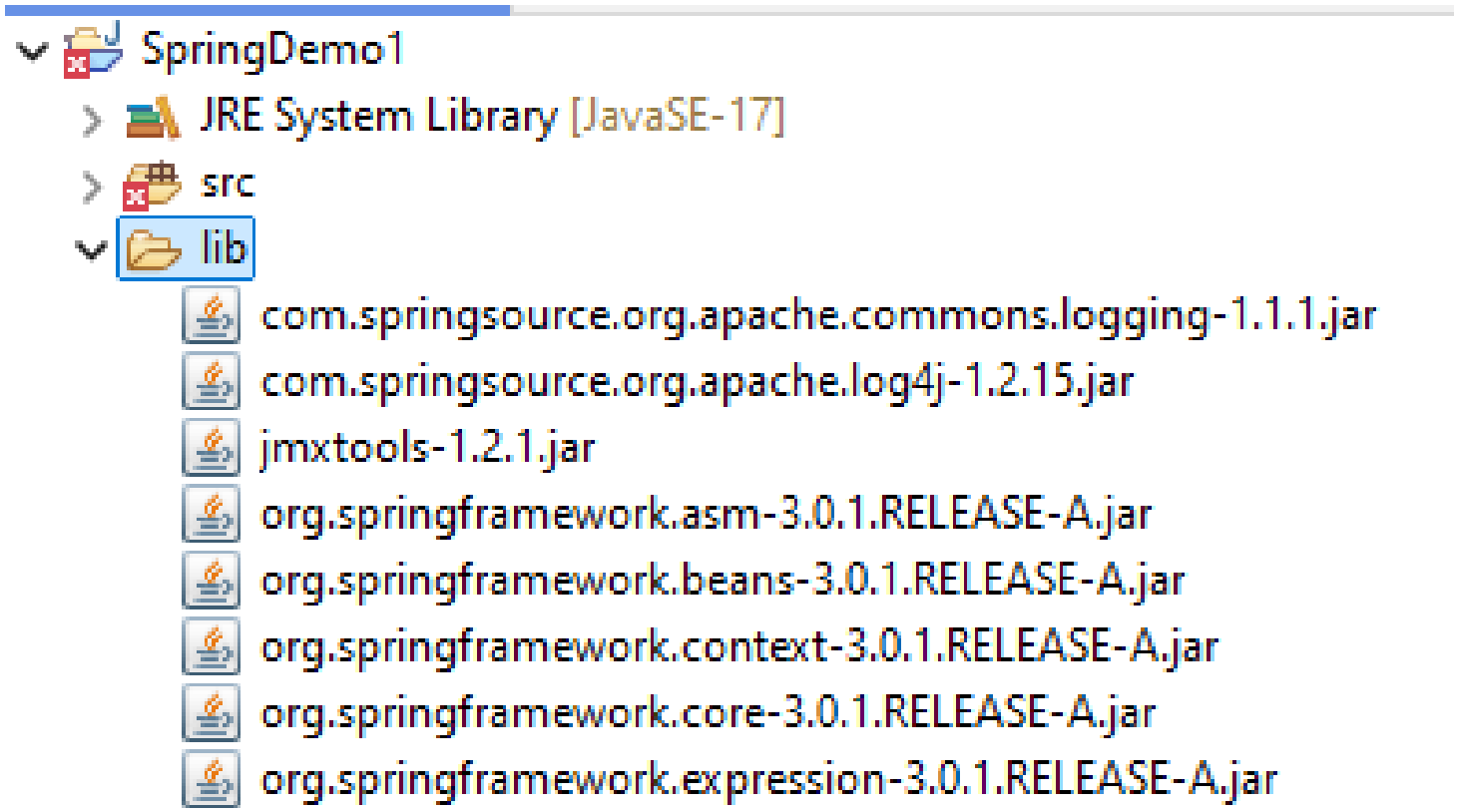
1. Download spring jars from https://repo.spring.io/release/org/springframework/spring/



Lets add Spring Framework to project

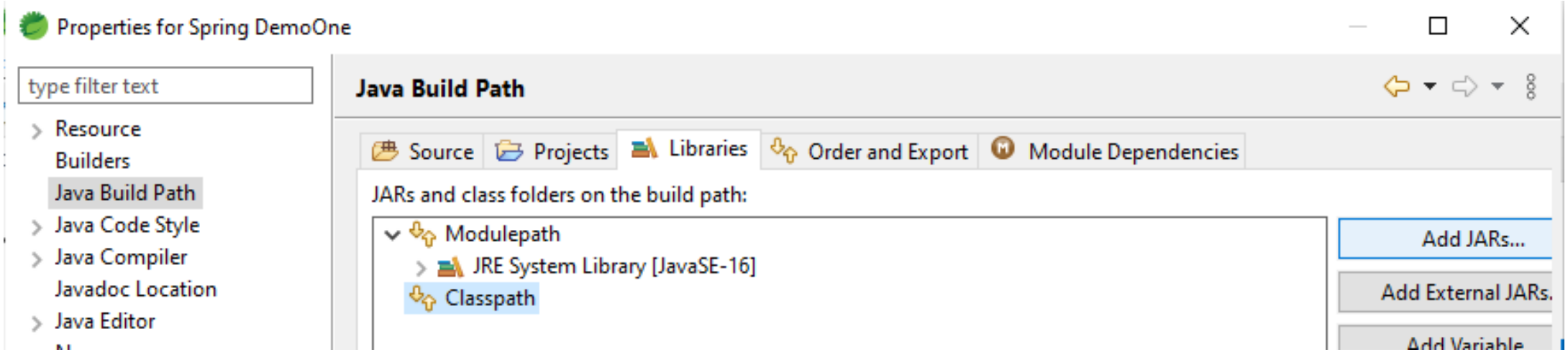
- Download & **extract** the dist.zip version

Create a folder “lib” in the project
Copy & paste all jar files from the extracted folder.

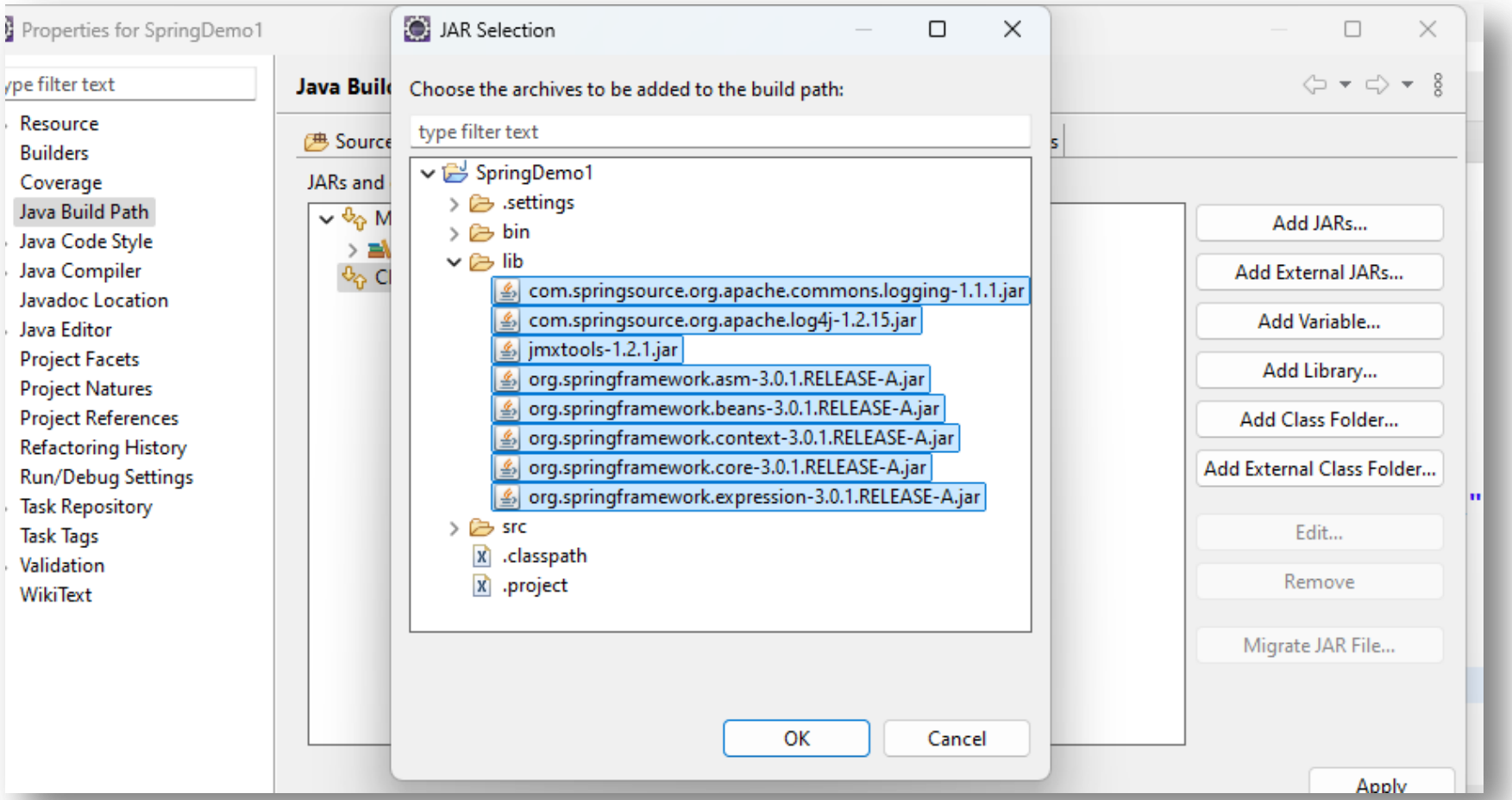


Lets add Spring Framework to project

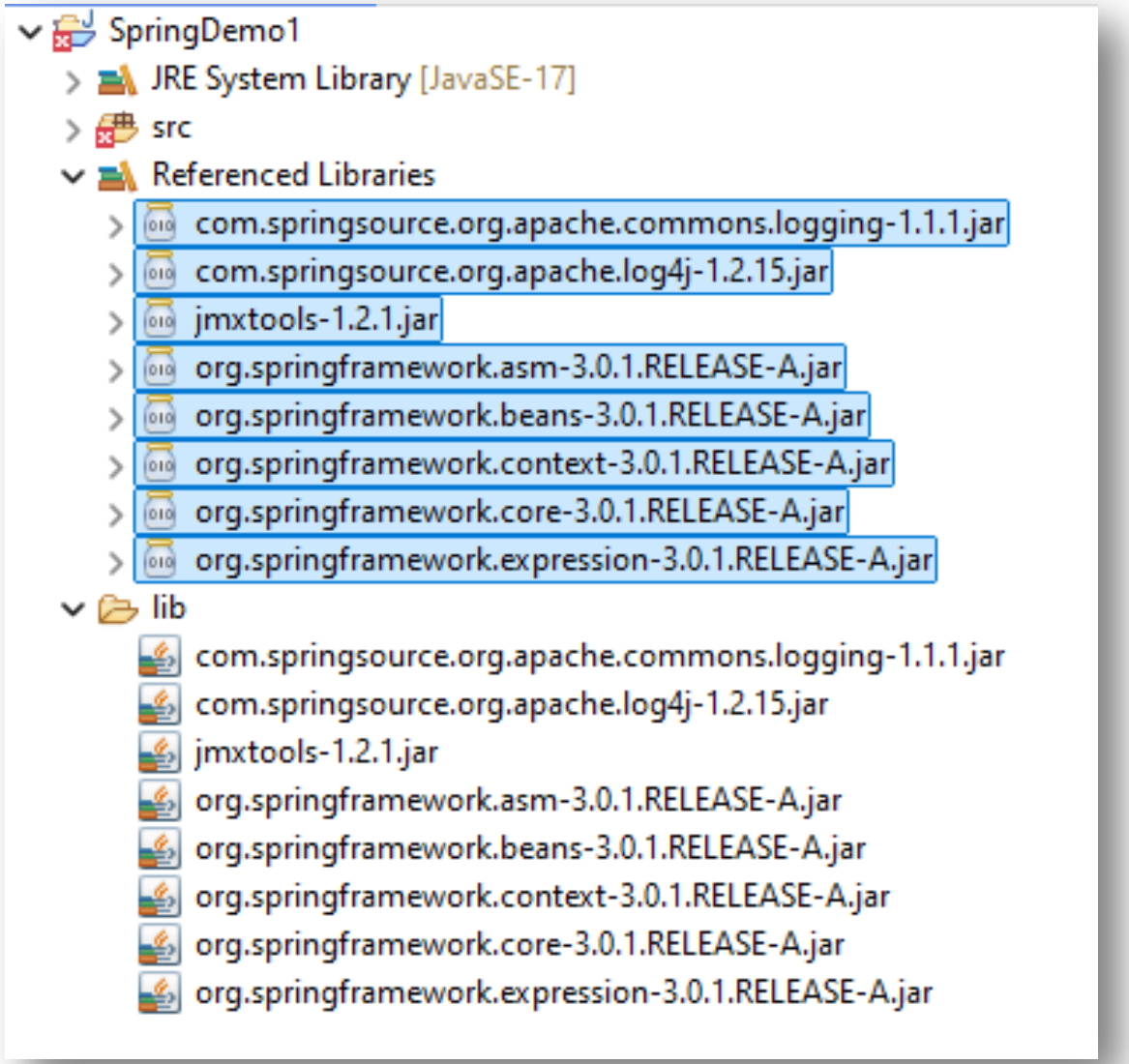
- RightClick (Project) →Properties → Java Build Path
- Select libraries → classpath → Add Jars.



Select all jars in the lib folder
Apply and Close



Spring Project is ready



Tight Coupling

Bike.java | MainDemo.java

```java
package com.rit.p1;

public class MainDemo {
    public static void main(String[] args) {
        Bike bike = new Bike();
        System.out.println(bike.getBike());
    }
}
```

```
<terminated> MainDemo [Java App
Hi Im using Bike
```

- SpringDemo1
  - JRE System Library [JavaSE-17]
  - src
    - com.rit.p1
      - Bike.java
      - Car.java
      - MainDemo.java

Bike.java

```java
package com.rit.p1;

public class Bike {
    String getBike() {
        return "Hi Im using Bike";
    }
}
```

If we upgrade from bike to car

Car.java | MainDemo.java

```java
package com.rit.p1;

public class Car {
    String getCar() {
        return "Hi Im using Car";
    }
}
```

Car.java | *MainDemo.java

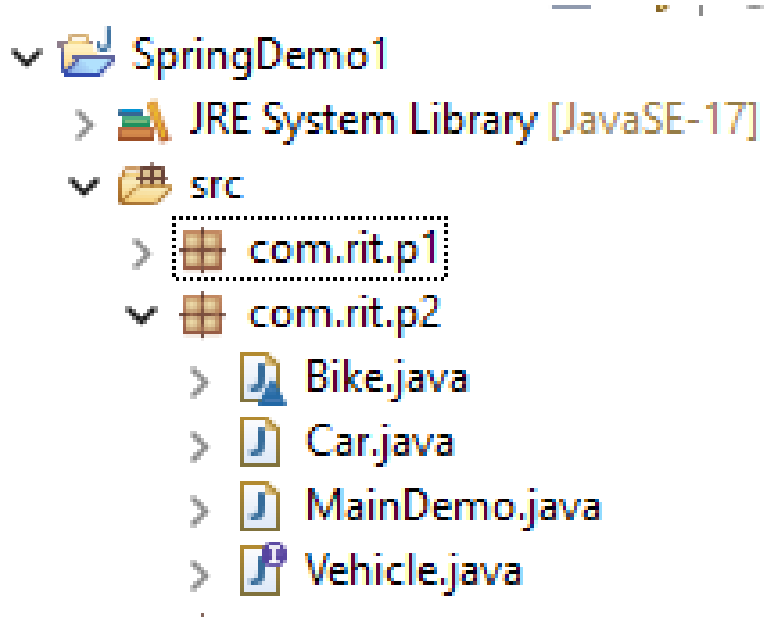
```java
package com.rit.p1;

public class MainDemo {
    public static void main(String[] args) {
//      Bike bike = new Bike();
//      System.out.println(bike.getBike());

        Car car = new Car();
        System.out.println(car.getCar());
    }
}
```

Problems | @ Javadoc

<terminated> MainDemo [Java A

```
Hi Im using Car
```

Loose Coupling



Vehicle.java

```java
package com.rit.p2;

public interface Vehicle {
    public String getVehicle();
}
```

Vehicle.java | Bike.java

```java
package com.rit.p2;

class Bike implements Vehicle {
    public String getVehicle() {
        return "Hi Im using Bike";
    }
}
```

Main Class

Vehicle.java Bike.java MainDemo.java

```java
package com.rit.p2;

public class MainDemo {
    public static void main(String[] args) {

        Vehicle vehicle = new Bike();
        System.out.println(vehicle.getVehicle());
    }
}
```

To upgrade with Car

Vehicle.java | Car.java | MainDemo.java

```java
package com.rit.p2;

public class Car implements Vehicle{
    public String getVehicle() {
        return "Hi Im using Car";
    }
}
```

Vehicle.java | Car.java | MainDemo.java

```java
package com.rit.p2;

public class MainDemo {
    public static void main(String[] args) {
//      Vehicle vehicle = new Bike();
        Vehicle vehicle = new Car();
        System.out.println(vehicle.getVehicle());
    }
}
```

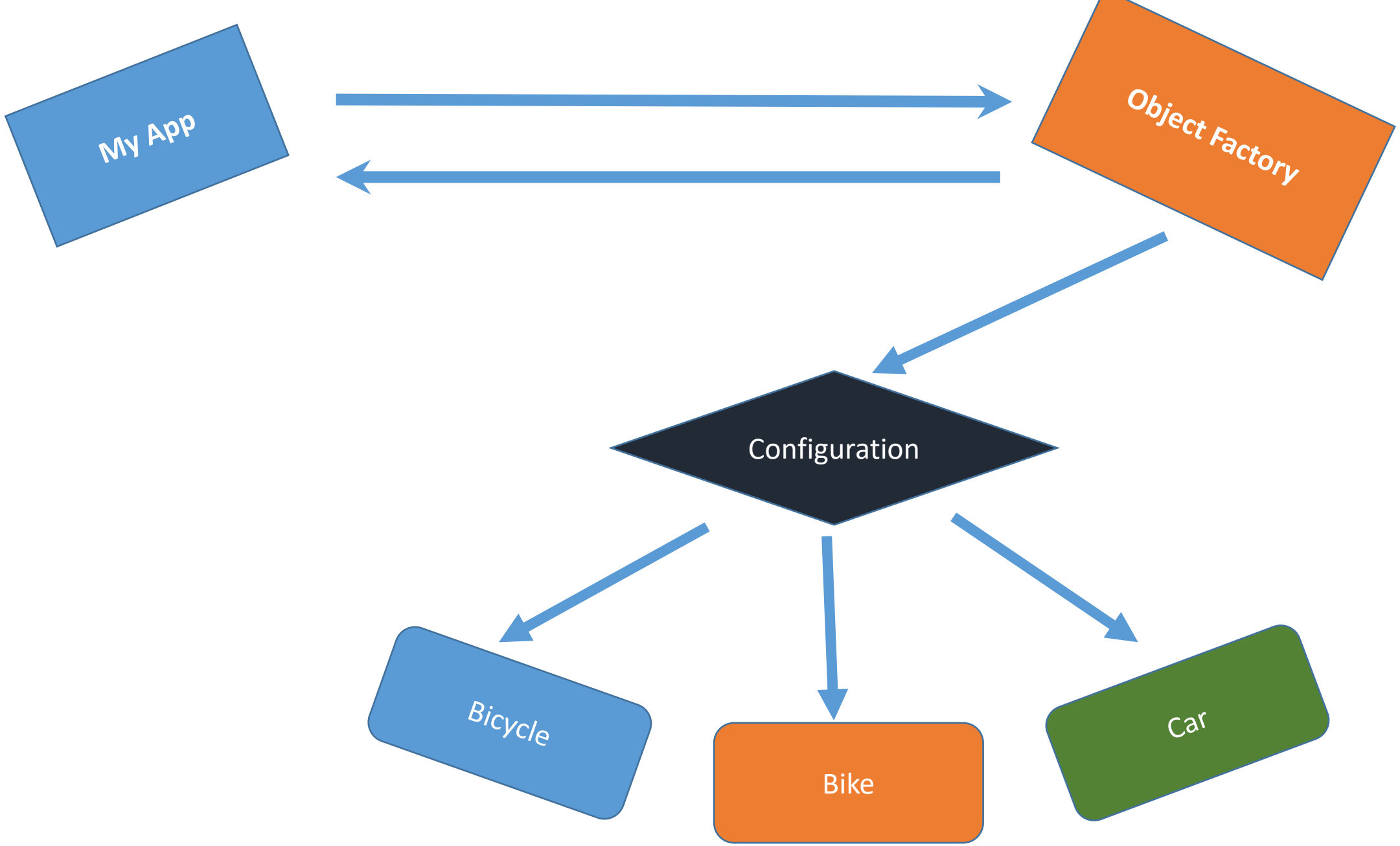
Inversion Of Control (IOC)

Inversion of Control

- One of the major principles of Software Engineering
- Inversion of Control (IoC) is a design principle that allows classes to be loosely coupled and, therefore, easier to test and maintain.
- IoC refers to transferring the control of objects and their dependencies from the main program to a container or framework.

Spring IoC Container

- 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.


My App
Object Factory
Configuration
Bicycle
Bike
Car

Types of Ioc Containers

There are 2 types of IoC containers:

- BeanFactory : is the most basic version of IoC containers

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

BeanFactory factory=new XmlBeanFactory(resource);

- ApplicationContext : extends the features of BeanFactory.

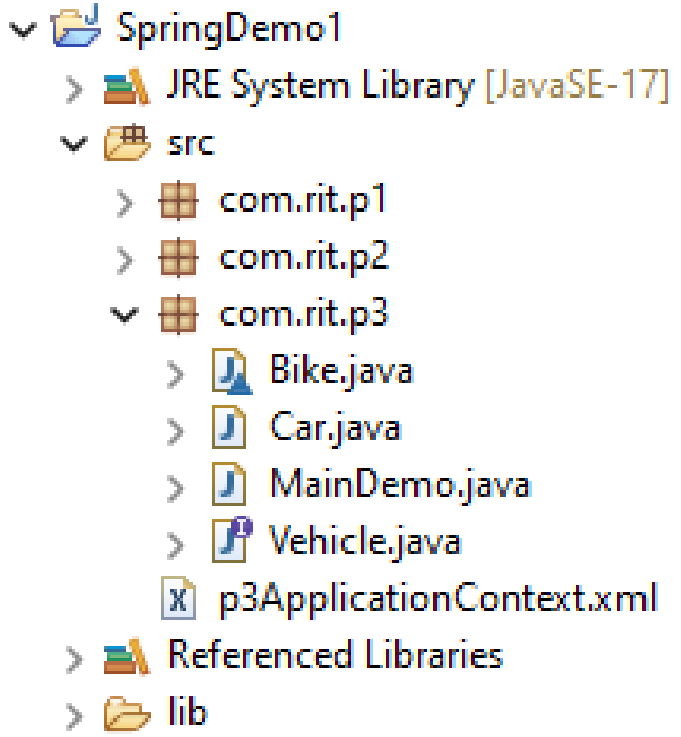
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

Implementing IOC in our project

- No change in Bike, Car & Vehicle
- We need to add a spring bean config file

  p3ApplicationContext.xml in src
- We have get the object through bean factory

  in main method



p3ApplicationContext.xml

```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="vehiclebean" class="com.rit.p3.Bike">
    </bean>
</beans>
```

p3ApplicationContext.xml MainDemo.java

```xml
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd (xsi:schemaLocati
<?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="vehiclebean" class="com.rit.p3.Bike">
    </bean>

</beans>
```

```java
public class MainDemo {
    public static void main(String[] args) {

        Resource resource=new ClassPathResource("p3ApplicationContext.xml");
        BeanFactory factory=new XmlBeanFactory(resource);

        Vehicle vehicle = (Vehicle) factory.getBean("vehiclebean");
        System.out.println(vehicle.getVehicle());

    }
}
```

p3ApplicationContext.xml | MainDemo.java

SpringDemo1/src/p3ApplicationContext.xml

```java
2
3⊕ import org.springframework.beans.factory.BeanFactory;
7
8  public class MainDemo {
9⊖     public static void main(String[] args) {
10         Resource resource=new ClassPathResource("p3ApplicationContext.xml");
11         BeanFactory factory=new XmlBeanFactory(resource);
12
13         Vehicle vehicle = (Vehicle) factory.getBean("vehiclebean");
14         System.out.println(vehicle.getVehicle());
15     }
16 }
```

Upgrade to Car

- Just change the class in p3ApplicationContext.xml

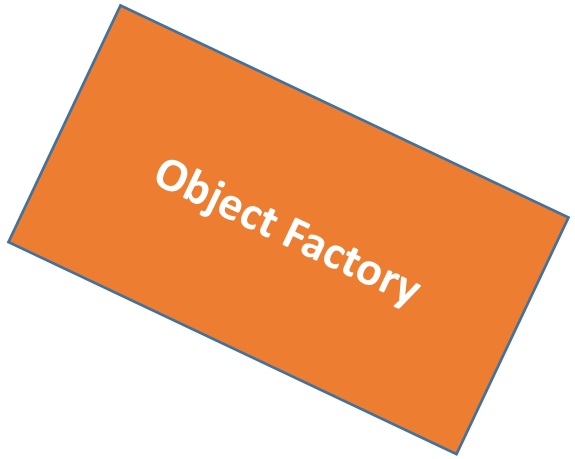
*p3ApplicationContext.xml MainDemo.java

```
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd (xsi:schemaLocati
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:p="http://www.springframework.org/schema/p"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
6                 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7
8      <bean id="vehiclebean" class="com.rit.p3.Car">
9      </bean>
10
11 </beans>
```

Main features of Spring IoC

- Creating Object for us,
- Managing our objects,
- Helping our application to be configurable,
- Managing dependencies

Spring Container



- Primary functions
  - Create and manage objects *(Inversion of Control)*
  - Inject object's dependencies *(Dependency Injection)*

- Configuring Spring Container
  - XML configuration file (legacy)
  - Java Annotation (Modern)
  - Java Source Code (Modern)

Spring Development Process

1. Configuring Spring Beans
2. Create a Spring Container
3. Retrieve Beans from Spring Container

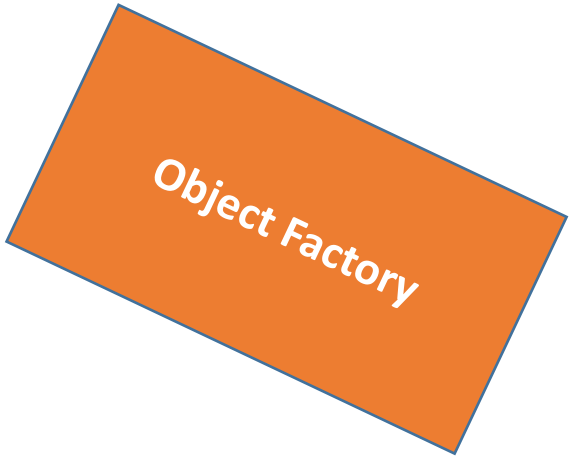
Step 1: Configuring Spring Beans

applicationContext.xml

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

  <bean id="myVehicle" class="com.rit.Bicycle">
  </bean>

</beans>
```

Step 2: Create a Spring Container

- Spring container is generally known as ApplicationContext
- Specialized implementations
  - ClassPathXmlApplicationContext
  - AnnotationConfigApplicationContext
  - GenericWebApplicationContext
  - Others…




```
ClasspathXmlApplicationContext context =
    new ClasspathXmlApplicationContext("applicationContext.xml")
```

Step 3: Retrieve Bean from Container

```
Vehicle theVehicle =
    content.getBean("myVehicle", Vehicle.class)
```

Using ClassPathXmlApplicationContext

```
package com.rit.p3;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainDemo2 {
    public static void main(String[] args) {
        ApplicationContext context = new
        ClassPathXmlApplicationContext("p3ApplicationContext.xml");

        Vehicle vehicle = context.getBean("vehiclebean", Vehicle.class);
        System.out.println(vehicle.getVehicle());
    }
}
```

Another Main Class

MainDemo2.java

```java
package com.rit.p3;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainDemo2 {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("p3ApplicationContext.xml");

        Vehicle vehicle = context.getBean("vehiclebean", Vehicle.class);
        System.out.println(vehicle.getVehicle());
    }
}
```

Thank you