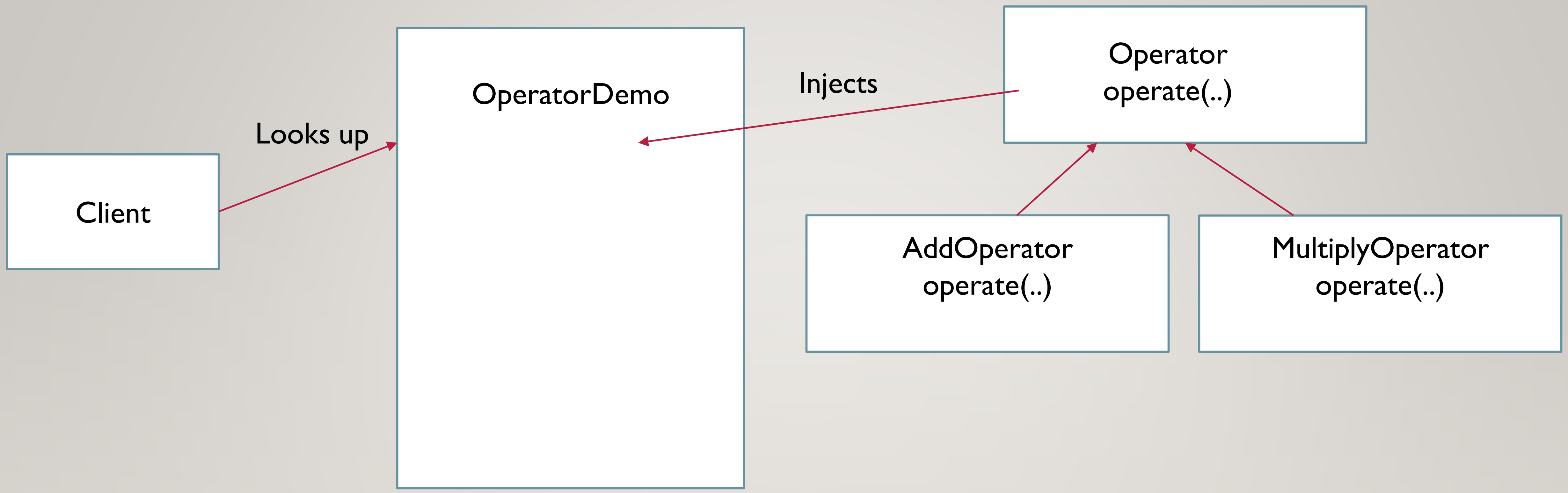
JAVA SPRING FRAMEWORK

WHAT IS SPRING?

- Spring is an IOC Container.
    - IOC stands for Inversion of Control
- The Spring framework is a powerful and flexible framework focused on building Java applications like standalone, web and web services.
- One of the core benefits of Spring is that it takes care of most of the low-level aspects of building the application to allow us to actually **focus on features and business logic**.

WHAT IS THE PRINCIPLE BEHIND SPRING?

- Dependency Injection is the principle behind Spring Framework.
- It is a Pattern used in IOC.
- Inversion of control is delegation of creating and managing the lifecycle of java components or beans to the Framework.
- Dependency Injection is injecting an object into another piece of code at runtime to create loosely coupled components.
- Basic benefit is configuration is separated from the business logic


OperatorDemo
Looks up
Client
Injects
Operator
operate(..)
AddOperator
operate(..)
MultiplyOperator
operate(..)

DEPENDENCY INJECTION TYPES

- If Operator is injected to OperatorDemo via Constructor it is Constructor Injection.
- If it is injected via setter it is Setter Injection.

```
public class OperatorDemo {
    private Operator operator;

    public OperatorDemo() {
  operator = new AddOperator();
    }
}
```

Without DI

Tightly Coupled

```
public class OperatorDemo {
 private Operator operator;
   public OperatorDemo(Operator operator) {
       this. operator = operator;
   }
public setOperator(Operator operator)
{
this. operator = operator;
}
```

Constructor Injection

Setter Injection

Loosely Coupled

```
public class OperatorDemo {

private Operator operator;
public Operator getOperator() { return
operator; }

  public void setOperator(Operator
operator) { this.operator = operator; }

public int getResult(int x,int y)
{
return operator.operate(x, y);
}

}
```

```
public interface Operator {

public int operate(int x,int y);

}
```

```
public class MultiplyOperator implements
Operator {

@Override
public int operate(int x, int y) {

return x * y;
}

}
```

Prepared by Radha V Krishna

Configuration through xml file (SpringBeans.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="operator" class="com.training.bean.MultiplyOperator">
</bean>

<bean id="demo" class="com.training.bean.OperatorDemo">

<constructor-arg name="Operator" ref=" operator "> </constructor-arg>

<property name="operator" ref=" operator "> </property>

</bean>

</beans>
```

Constructor Injection

Setter Injection

```
<!-- A bean definition with singleton scope -->
<bean id = "..." class = "..." scope = "singleton">
</bean
```

This scopes the bean definition to a single instance per Spring IoC container

prototype

This scopes a single bean definition to have any number of object instances

request

This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

session

This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

SPRING HELLO WORLD

- Steps
  - Create the Bean Class
  - Create a xml file to configure the Bean class
  - Write a Client

```
//add this dependency
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.0.7.RELEASE</version>
</dependency>
```

TEST CLIENT

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

        OperatorDemo obj = (OperatorDemo) context.getBean(" demo ");
        System.out.print(“Result:”+obj.getResult(12,20));
    }
}
```

AUTOWIRING

- There are different ways through which we can autowire a spring bean.
- autowire byName – For this type of autowiring, setter method is used for dependency injection. Also the variable name should be same in the class where we will inject the dependency and in the spring bean configuration file.
- autowire byType – For this type of autowiring, class type is used. So there should be only one bean configured for this type in the spring bean configuration file.
- autowire by constructor – This is almost similar to autowire byType, the only difference is that constructor is used to inject the dependency.

BEAN LIFECYCLE

```
<bean id="operationDemo"


class="com.classes.OperationDemo" autowire="byType" init-method="init"
destroy-method="destroy">

 <!-- <property name="oInterface" ref="operationId"></property>  -->

</bean>
```

Before the business logic executes init-method is called as soon as the bean is loaded into memory.

Before the bean is deallocated destroy method is called.

COLLECTION INJECTION

```
<property name="addressList">
    <list>
      <value>INDIA</value>
      <value>Pakistan</value>
      <value>USA</value>
      <value>USA</value>
    </list>
   </property>
```

```
<property name="addressSet">
    <set>
      <value>INDIA</value>
      <value>Pakistan</value>
      <value>USA</value>
      <value>USA</value>
    </set>
   </property>
```

```
<property name="addressMap">
    <map>
      <entry key="1" value="NDIA"/>
      <entry key="2" value="Pakistan"/>
      <entry key="3" value="USA"/>
      <entry key="4" value="USA"/>
    </map>
   </property>
```

```
<property name="addressProp">
    <props>
      <prop key="one">INDIA</prop>
      <prop key="two">Pakistan</prop>
      <prop key="three">USA</prop>
      <prop key="four">USA</prop>
    </props>
   </property>
```

Configuration via Java Class

```
public interface MyServiceI {
        public String sayHello();
}
```

```
@Service
public class MyService implements MyServiceI {
        public String sayHello()
        {
                return "This is my New Hello Service";
        }}
```

```
@Configuration
public class MyConfiguration {
        @Bean
        public MyService getService()
        {
                return new MyService();
        }
```

```
//Client.java
public class Client {

	public static void main(String[] args) {

		AnnotationConfigApplicationContext context= new
		AnnotationConfigApplicationContext(MyConfiguration.class);
MyServiceI  myService=context.getBean(MyServiceI.class);
		System.out.println(myService.sayHello());

	}
```

Configure bean in MyConfiguration.java

```
@Bean
public Operator getOperator()
{
return new MultiplyOperator();
}


@Bean
public OperatorDemo getOperatorDemo()
{
return new OperatorDemo();
}
```

```
public class OperatorDemo {

@Autowired
private Operator operator;

public int getResult(int x,int y)
{
return operator.operate(x, y);
}
```

Autowires the configured bean

```
public class App3
{
    public static void main( String[] args )
    {

AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(MyConfiguration.class);


OperatorDemo op = (OperatorDemo)context.getBean(OperatorDemo.class);
System.out.println(op.getResult(12, 34));
    }


}
```