Question 5:

@Inject Demo:

1. **A simple POJO class as Employee having Address POJO reference variable injected :**

```
package com.chandan;
import javax.inject.Inject;

//Create a POJO class Employee which has a
//Address Object reference as instance variable
publicclass Employee {

private String name;
privateintage;

@Inject
private Address address;

public Employee() {

}

public String getName() {
returnname;
}

publicvoid setName(String name) {
this.name = name;
}

publicint getAge() {
returnage;
}

publicvoid setAge(intage) {
this.age = age;
}

public Address getAddress() {
returnaddress;
}

publicvoid setAddress(Address address) {
this.address = address;
}

      }
```

2. **Address POJO as Injected into Employee**

```
package com.chandan;
publicclass Address {
```

```
	private String street;
	private String city;
	private String state;

	public Address() {

	 }


	public String getStreet() {
	returnstreet;
	 }


	publicvoid setStreet(String street) {
	this.street = street;
	 }


	public String getCity() {
	returncity;
	 }


	publicvoid setCity(String city) {
	this.city = city;
	 }


	public String getState() {
	returnstate;
	 }


	publicvoid setState(String state) {
	this.state = state;
	 }

		}
```

3. **Test class for testing the application :**

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

publicclass Test {

publicstaticvoid main(String[] args) {

// ApplicationContext is a Spring interface which  provides with the configuration
for an application.
// It provides us with all the methods that BeanFactory provides. It loads the file
resources in a older
```

```
      //and generic manner.
// We are using concrete implementation of ApplicationContext
// here called as ClassPathXmlApplicationContext because this
// bean factory reads the xml file placed in the classpath of
// our application. We provide ClassPathXmlApplicationContext
// with a configuration file called as inject.xml placed
// at classpath of our application.
  ApplicationContext context =  new ClassPathXmlApplicationContext(("inject.xml"));

// In order to get a object instantiated for a particular bean
// we call getBean() method of ClassPathXmlApplicationContext
// passing it the id for which the object is to be needed.
// Here getBean() returns an Object. We need to cast it back
// to the Employee object. Without implementing new keyword we
// have injected object of Employee just by reading an xml
// configuration file.
  Employee employee = (Employee)context.getBean("employee");

if(employee.getAddress()==null){
   System.out.println("The Employee Name : " + employee.getName());
   System.out.println("The Employee Age : " + employee.getAge());
   System.out.println("The Employee Address : " + "is not provided");
  }
else{
   System.out.println("The Employee Name : " + employee.getName());
   System.out.println("The Employee Age : " + employee.getAge());
   System.out.println("The Employee Address : " +
employee.getAddress().getStreet() + " " +
employee.getAddress().getCity() + " " +
employee.getAddress().getState());
  }
       }
```

4. **Spring configuration file for the @Inject annotation :**

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.1.xsd">


<beanid="employee"class="com.chandan.Employee">

<propertyname="age"value="22"/>
<propertyname="name"value="Chandan"/>


</bean>

<beanid="address"class="com.chandan.Address">
```

```
<propertyname="street"value="Puneeth Street"/>
<propertyname="city"value="Bang"/>
<propertyname="state"value="Kar"></property>
</bean>

	</beans>
```

Output of the program :

@Required Demo:

1. **Create a HelloWorld class**

```
package com.helloworld;

importorg.springframework.beans.factory.annotation.Required;

publicclass HelloWorld {
Stringtype;

	publicString getType() {
		returntype;
	}
	@Required
	publicvoid setType(Stringtype) {
		this.type = type;
	}

	publicvoid method() {
		System.out.println("Hello World!! of type: "+getType());
```

```
	}

}
```

2. **Create a Main class**

```
package com.helloworld;

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

import com.demo.HelloWorld;

publicclass Main {

	publicstaticvoid main(String[] args) {
		ApplicationContext context=new
ClassPathXmlApplicationContext("Beans.xml");
		HelloWorld obj=(HelloWorld) context.getBean("helloWorld");
		obj.method();

	}

}
```

3. **Create Beans.xml file**

```
<?xmlversion="1.0"encoding="UTF-8"?>
<beansxmlns="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">

<beanid="helloWorld"class="com.helloworld.HelloWorld">


</bean>
	<beanclass="org.springframework.beans.factory.annotation.RequiredAnnotationBea
nPostProcessor"/>
      </beans>
```

Output:it will raise *BeanInitializationException* exception and print the following

error along with other log messages- Property 'type' is required for bean

'helloWorld'

@RESOURCE

1.Create a Maven Project First

2. Create a Pacakage

3.Create Class

Maven Dependencies

pom.xml

```
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.spring</groupId>
  <artifactId>SpringResourceAnnotationExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>5.0.8.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-context --
>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.8.RELEASE</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api
-->
<dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>1.3.2</version>
</dependency>
   <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.12</version>
</dependency>
```

```
    </dependencies>
    <build>
        <finalName>${project.artifactId}</finalName>
    </build>
</project>
```

Java Class Creation

Implementation of Company Model

This `POJO` class contains two properties to perform the `byName` autowiring. Add the following code to it:

Company.java

```
package com.spring.pojo;

public class Company {
       private String name;
    private String location;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getLocation() {
        return location;
    }
    public void setLocation(String location) {
        this.location = location;
    }

    @Override
    public String toString() {
        return "Company [name=" + name + ", location=" + location + "]";

}
}
```

Implementation of Employee Model

This POJO class contains three setter methods for demonstrating the use of `@Resource` annotation. Add the following code to it:

Employee.java

```
package com.spring.pojo;

import javax.annotation.Resource;
       public class Employee {
```

```
            private String id;
            private String name;

            @Resource(name="mycompany")
            private Company company;

            public String getId() {
                return id;
            }
            public void setId(String id) {
                this.id = id;
            }
            public String getName() {
                return name;
            }
            public void setName(String name) {
                this.name = name;
            }
            public Company getCompany() {
                return company;
            }
            public void setCompany(Company company) {
                this.company = company;
            }

            @Override
            public String toString() {
                return "Employee [id=" + id + ", name=" + name + ", company=" +
company.toString() + "]";

}
       }
```

Implementation of Utility Class

The implementation class will get the bean definition from the context file and demonstrate the use of `@Resource` annotation in the spring framework. Add the following code to it:
AppMain.java

```
package com.spring.util;

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

import com.spring.pojo.Employee;

public class AppMain {
       @SuppressWarnings("resource")
       public static void main(String[] args) {
              ApplicationContext ac = new ClassPathXmlApplicationContext("resource-
annotation.xml");
```

```
            Employee emp = ac.getBean("myemployee", Employee.class);
        System.out.println(emp.toString());
       }

}
```

Configuration Files

Let us write all the configuration files involved in this tutorial.

Resource

A typical bean configuration file for understanding the `@Resource` annotation will look like this:

resource-annotation.xml

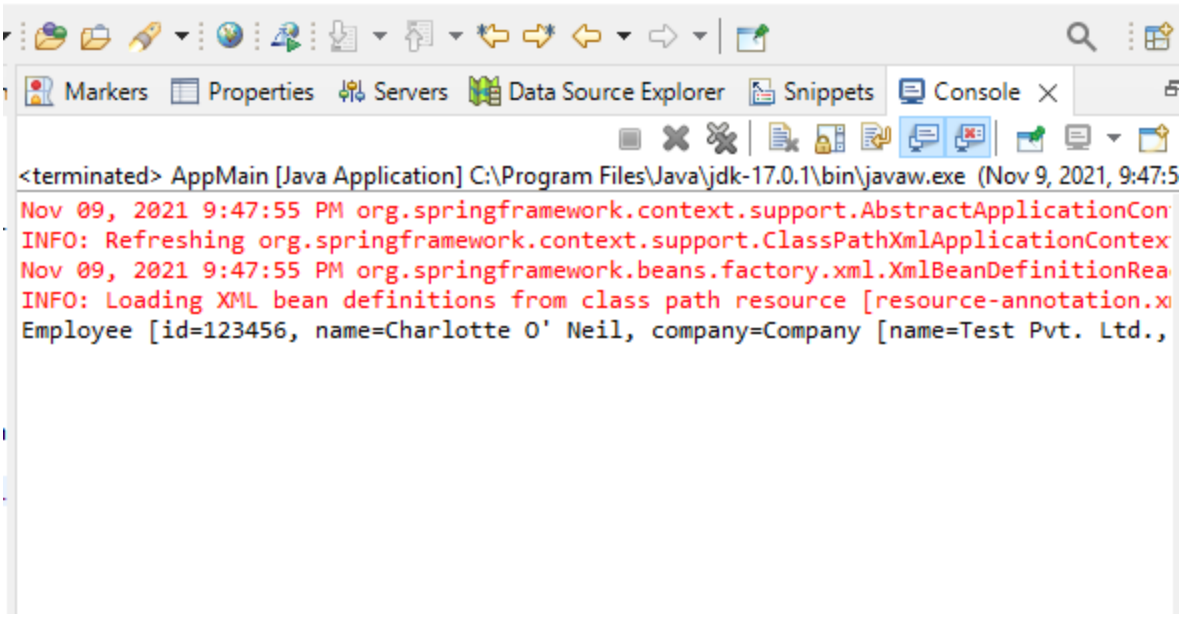
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- To activate the '@Resource' annotation in the spring framework -->
    <context:annotation-config />

    <bean id="mycompany" class="com.spring.pojo.Company">
        <property name="name" value="Test Pvt. Ltd." />
        <property name="location" value="India" />
    </bean>

    <bean id="myemployee" class="com.spring.pojo.Employee">
        <property name="id" value="123456" />
        <property name="name" value="Charlotte O' Neil" />
    </bean>
</beans>
```

OUTPUT:-


Markers
Properties
Servers
Data Source Explorer
Snippets
Console
<terminated> AppMain [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (Nov 9, 2021, 9:47:5
Nov 09, 2021 9:47:55 PM org.springframework.context.support.AbstractApplicationCon
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContex
Nov 09, 2021 9:47:55 PM org.springframework.beans.factory.xml.XmlBeanDefinitionRea
INFO: Loading XML bean definitions from class path resource [resource-annotation.x
Employee [id=123456, name=Charlotte O' Neil, company=Company [name=Test Pvt. Ltd.,