**Spring Core Assignments**

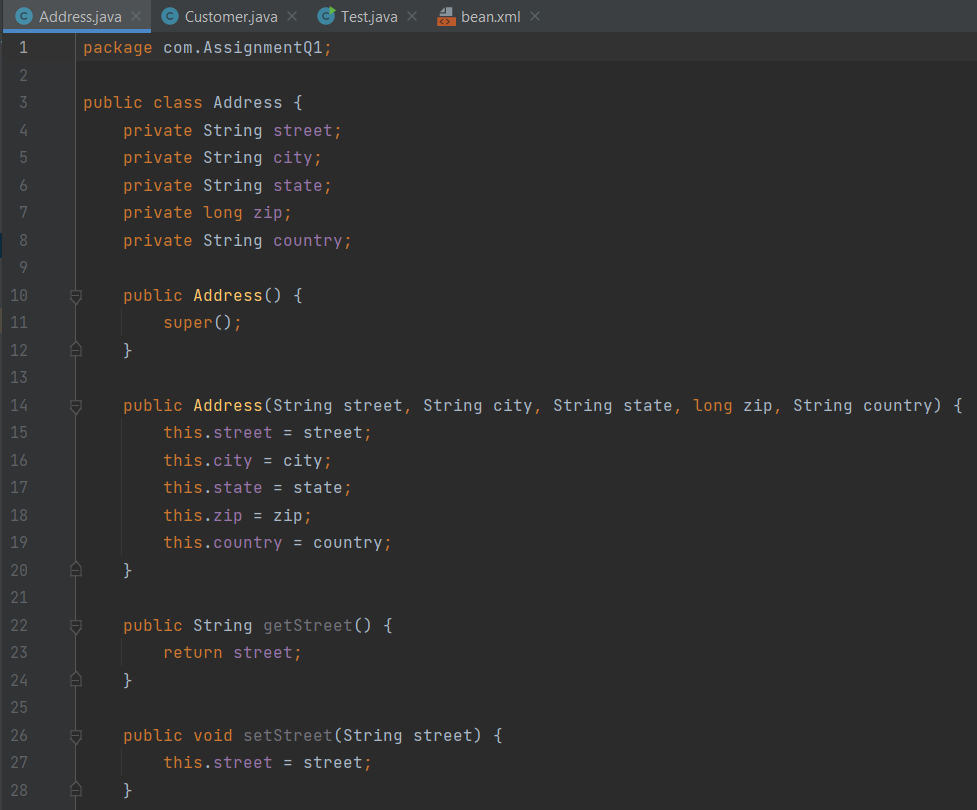
**PRATIK GUPTA**

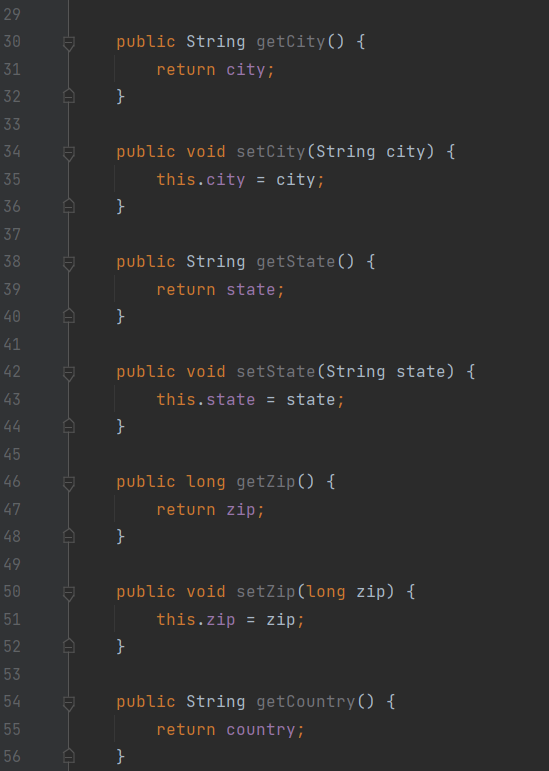
1) Create an Address class with the following attributes:- street, city, state, zip, country Create an Customer class with the following attributes:- customerId, customerName, customerContact, customerAddress. Inject the Address bean into Customer bean using setter injection Create a Test class with main() method, get Customer bean from ApplicationContext object and print details of Customer. Also write the JUnit Test cases for above program.

- Modify the above application and inject the bean using constructor injection

- Use XML based Configuration.

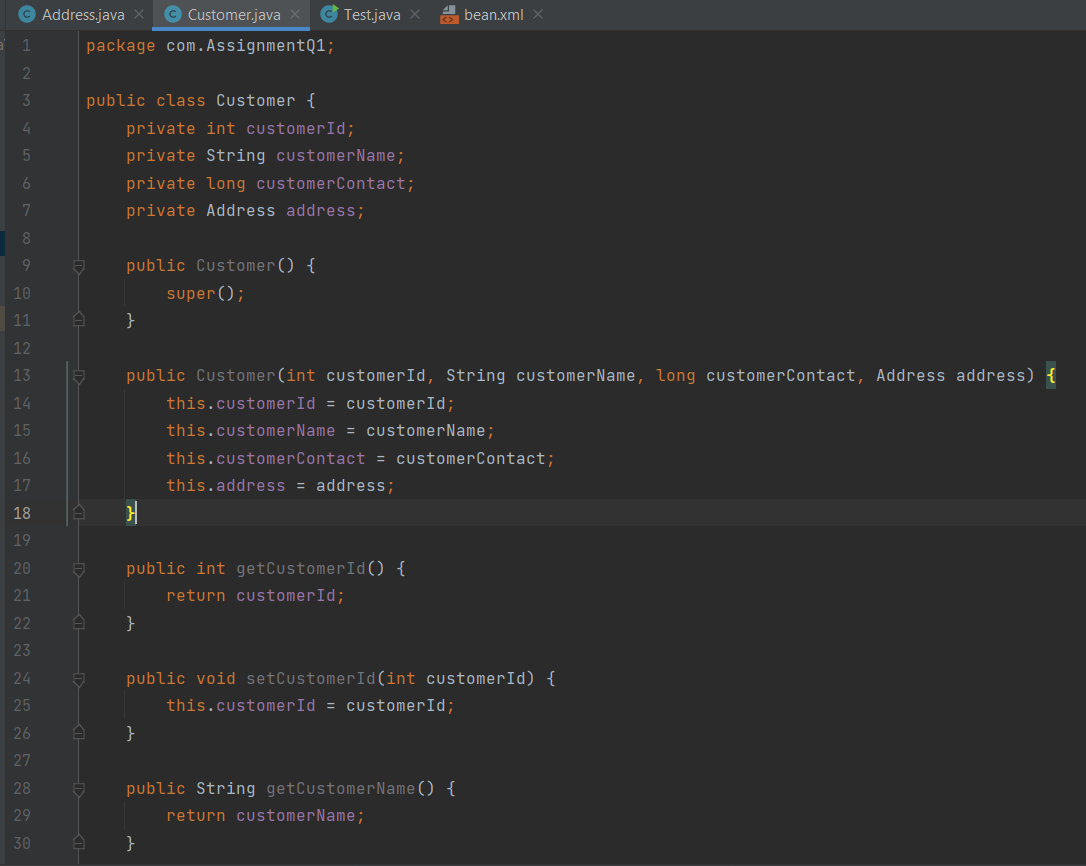
1] Address class





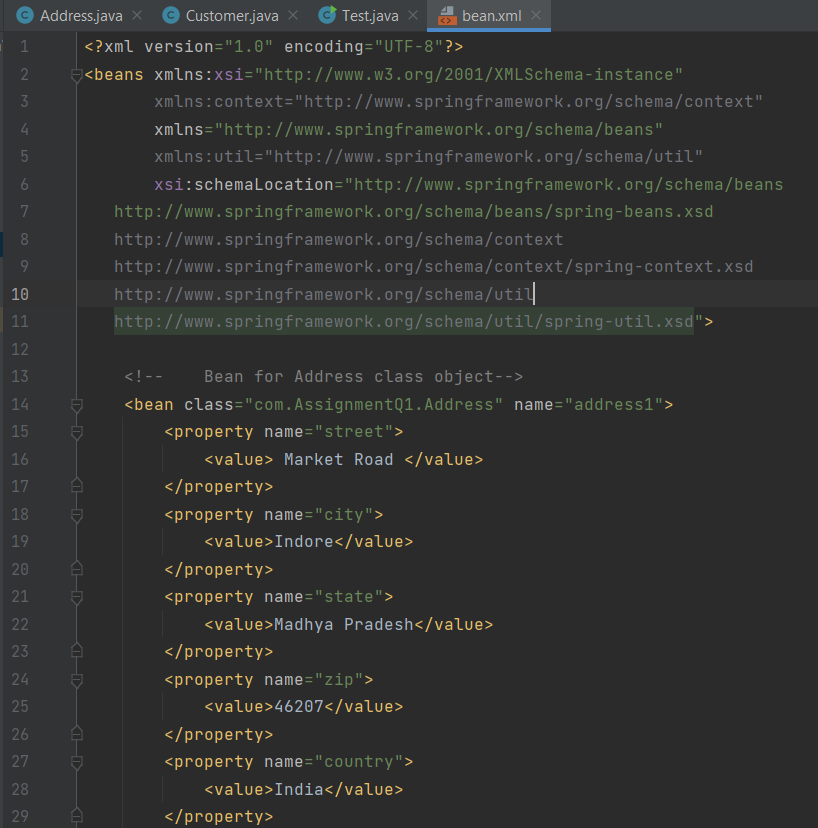


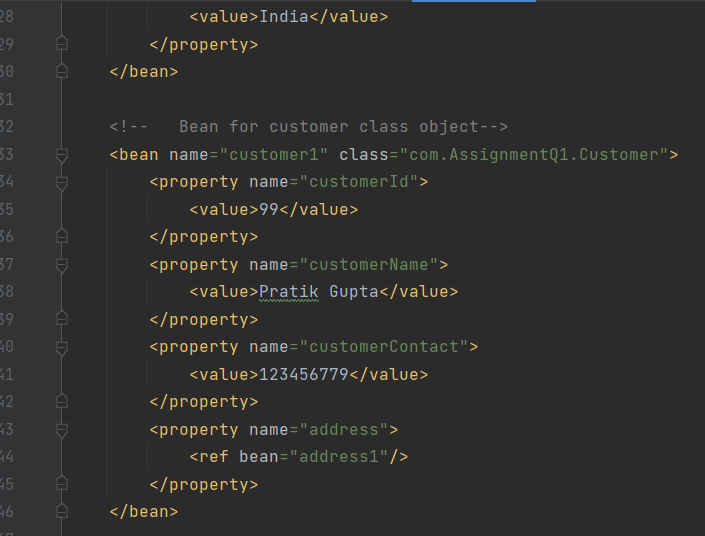
2] Customer class

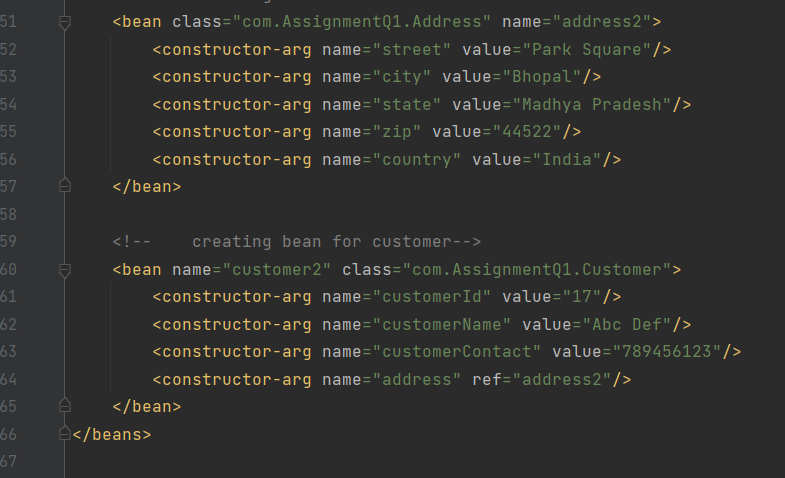




3] XML File







3] Test



OUTPUT :-

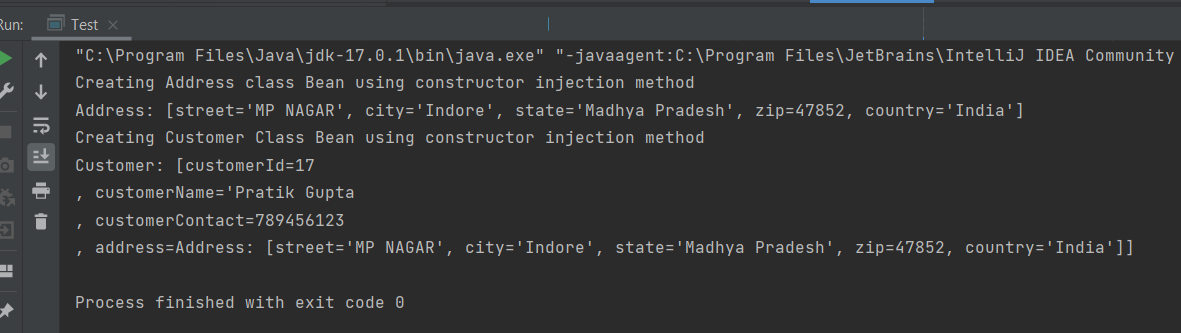


**Constructor Injection Method:**

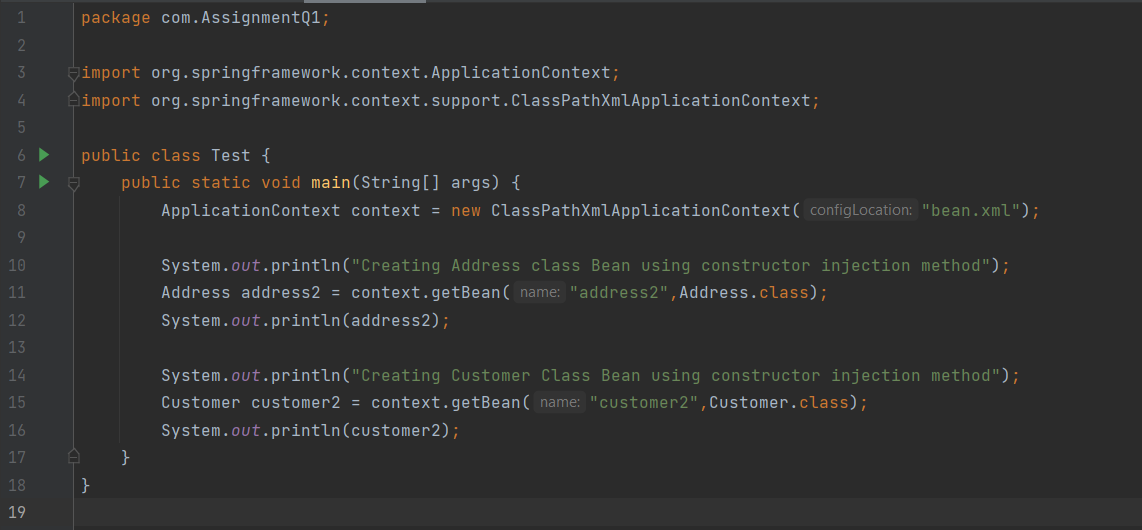
**XML Code:**



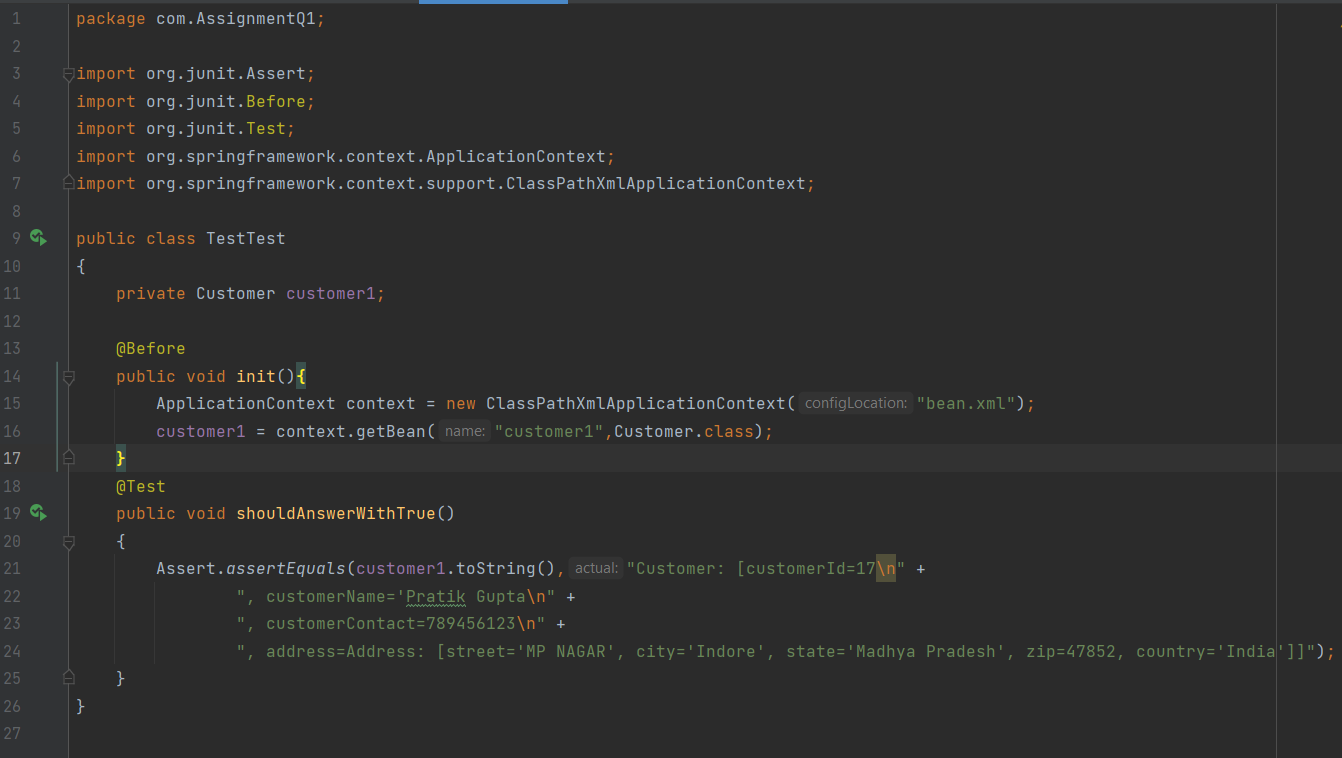
OUTPUT :-



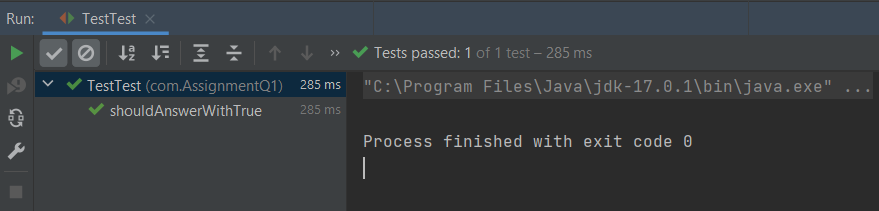
Test.java



Junit Testing code:

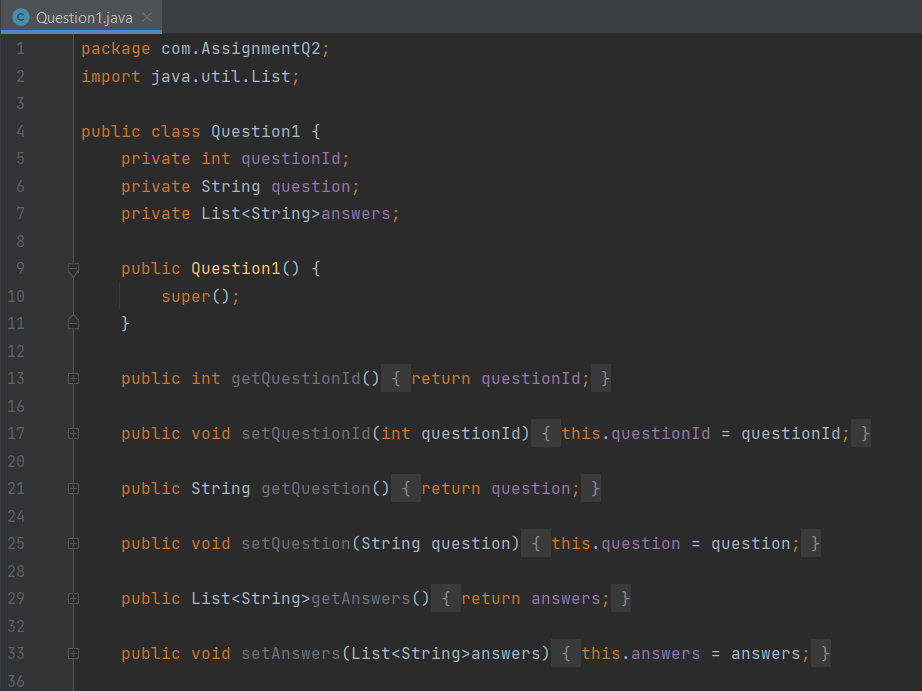


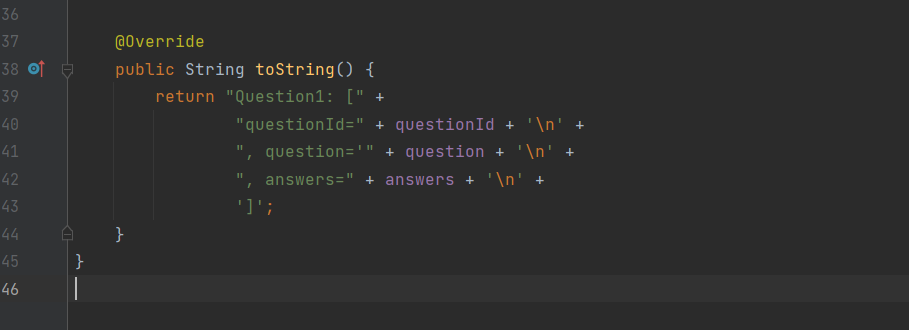
Junit Testing Output :-



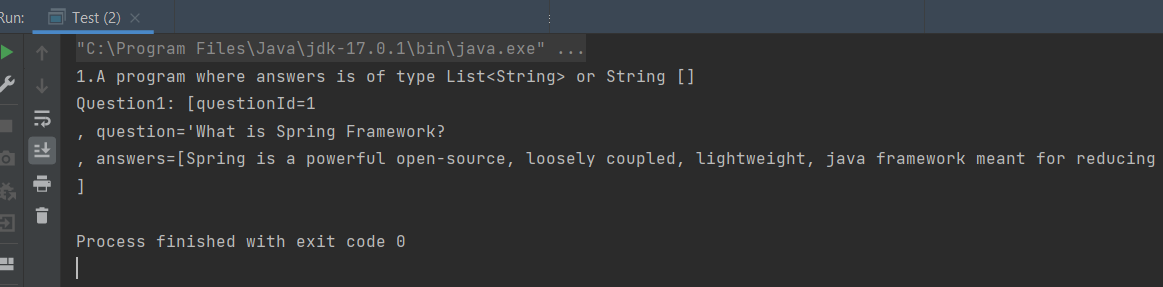
2) Example of Injecting collections (List, Set and Map) Create a class Question with following attributes: questionId, question, answers. There are 3 cases for above program.

a. Write a program where answers is of type List or String []



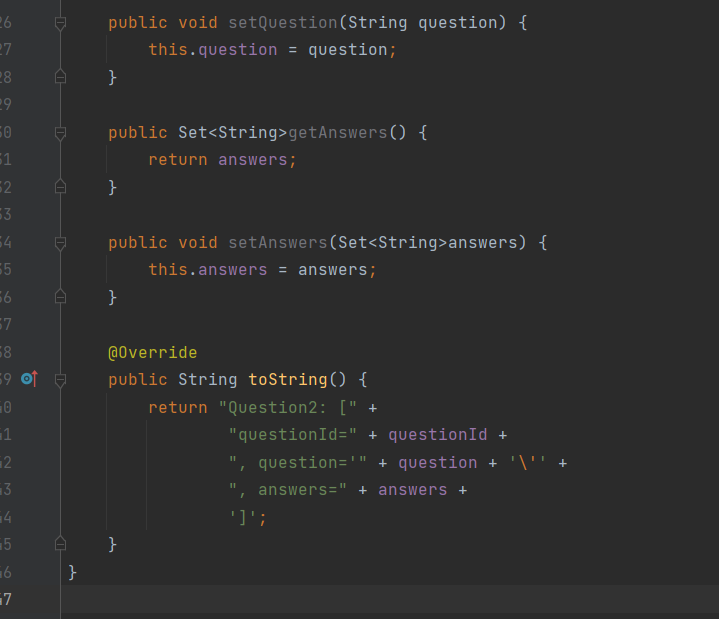


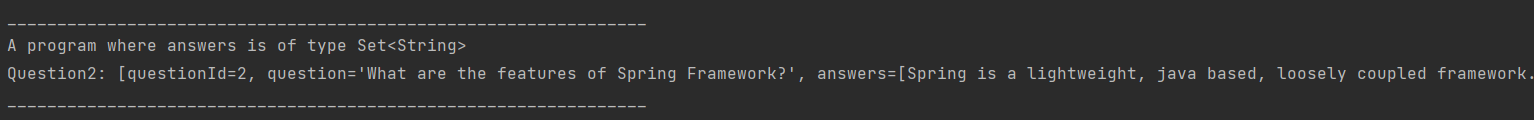
OUTPUT



b. Write a program where answers is of type Set

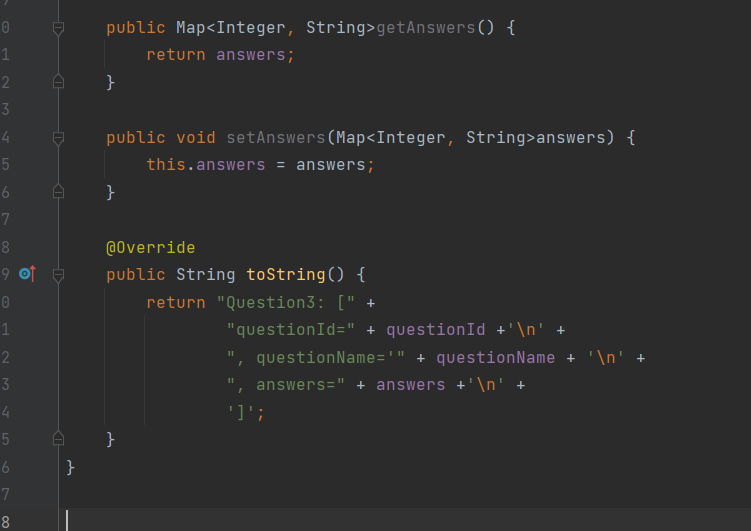


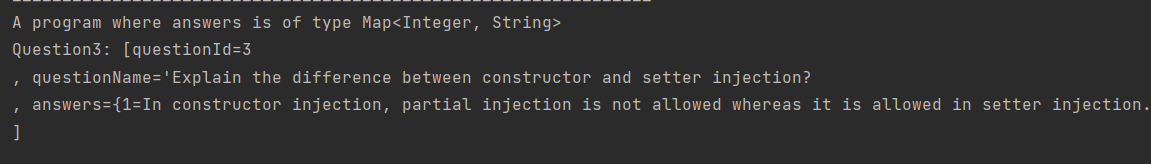




c. Write a program where answers is of type Map In case of Map, Integer value represents answer’s sequence number.

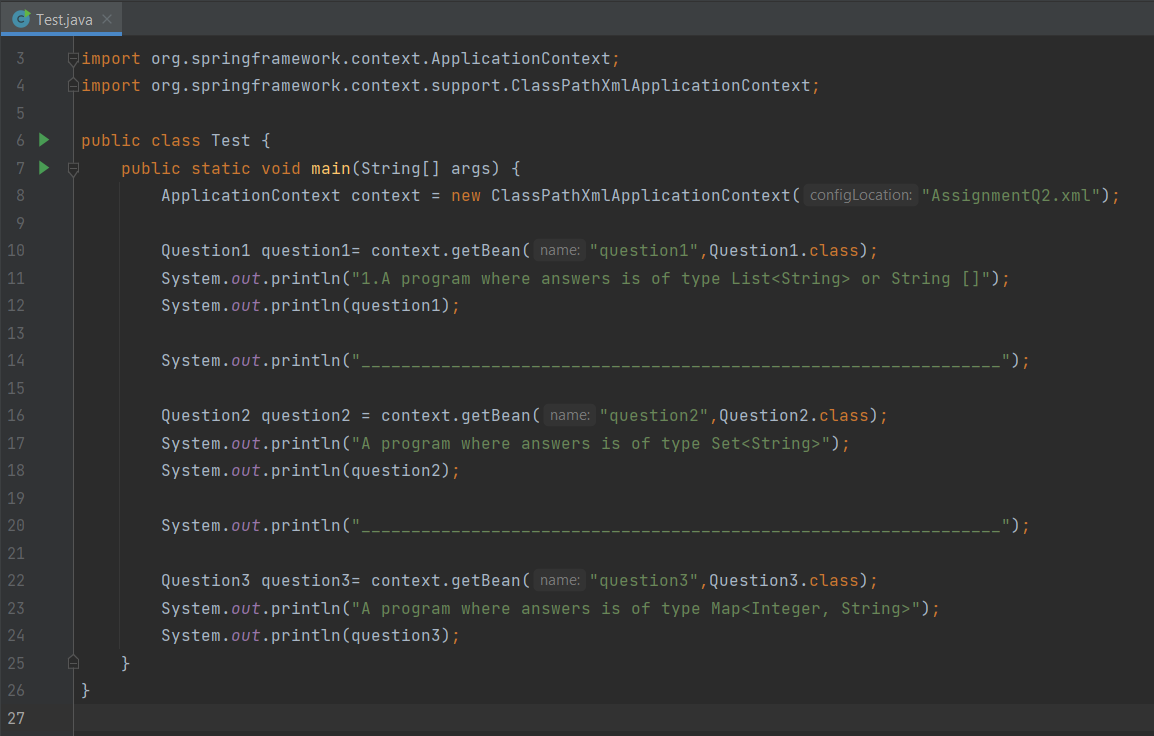






d. Create a Test class with main() method, get Question bean from ApplicationContext object and print question and its answers.

e. Also write the JUnit Test cases for above program. - Use XML based configuration.



**XML FILE**

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

<bean class="com.AssignmentQ2.Question1" name="question1">  
 <property name="questionId" value="1"/>  
 <property name="question" value="What is Spring Framework?"/>  
 <property name="answers">  
 <list>  
 <value>Spring is a powerful open-source, loosely coupled, lightweight, java framework meant for reducing the complexity of developing enterprise-level applications.</value>  
 <value>This framework is also called the “framework of frameworks” as spring provides support to various other important frameworks like JSF, Hibernate, Structs, EJB, etc.</value>  
 <value>Spring handles all the infrastructure-related aspects which lets the programmer focus mostly on application development.</value>  
 </list>  
 </property>  
</bean>  
  
 <bean class="com.AssignmentQ2.Question2" name="question2">  
 <property name="questionId" value="2"/>  
 <property name="question" value="What are the features of Spring Framework?"/>  
 <property name="answers">  
 <set>  
 <value>Spring is a lightweight, java based, loosely coupled framework.</value>  
 <value>Spring provides generic abstraction layer for transaction management that is also very useful for container-less environments.</value>  
 </set>  
 </property>  
 </bean>  
   
 <bean class="com.AssignmentQ2.Question3" name="question3">  
 <property name="questionId" value="3"/>  
 <property name="questionName" value="Explain the difference between constructor and setter injection?"/>  
 <property name="answers">  
 <map>  
 <entry key="1">

<value>In constructor injection, partial injection is not allowed whereas it is allowed in setter injection.</value>  
 </entry>  
 <entry key="2">  
 <value>The constructor injection doesn’t override the setter property whereas the same is not true for setter injection.</value>  
 </entry>  
 <entry key="3">  
 <value>Constructor injection creates a new instance if any modification is done. The creation of a new instance is not possible in setter injection.</value>  
 </entry>  
 <entry key="4">  
 <value>In case the bean has many properties, then constructor injection is preferred. If it has few properties, then setter injection is preferred.</value>  
 </entry>  
  
 </map>  
 </property>  
 </bean>  
   
</beans>

3) Example on autowiring Design and Develop a Banking Application as follows:

a. Create a BankAccount class with following attributes: accountId, accountHolderName, accountType, accountBalance

b. Create an interface BankAccountRepository with following methods: public double getBalance(long accountId) public double updateBalance(long accountId, double newBalance)

: Note: Above method returns updated balance.

c. Create a class BankAccountepositoryImpl that implements BankAccountRepository interface. You can use database or any collection object as persistence store.

d. Create an interface BankAccountService with following methods: public double withdraw(long accountId, double balance) public double deposit(long accountId, double balance) public double getBalance(long accountId) public boolean fundTransfer(long fromAccount, long toAccount, double amont)

e. Create a class BankAccountServiceImpl that implements BankAccountService interface.

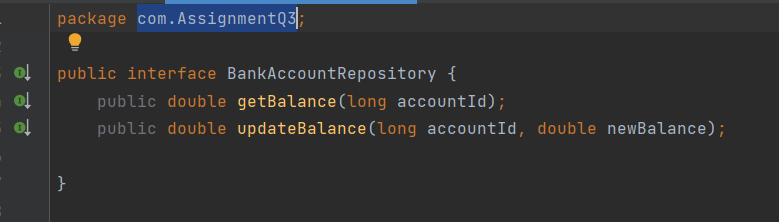
f. Create a class BankAccount controller with following operations: public double withdraw(long accountId, double balance) public double deposit(long accountId, double balance) public double getBalance(long accountId) public boolean fundTransfer(long fromAccount, long toAccount, double amont)

g. Create a Test class with main() method, get BankAccountController bean object from ApplicationContext and perform all the operations. h. Also write the JUnit Test cases for above program. - Use XML based configuration and perform autowiring with different types. (byName, byType and constructor). Use one autowiring type at a time.

**BankAccount.java class**

package com.AssignmentQ3;  
  
public class BankAccount {  
 private int accountId;  
 private String accountHolderName;  
 private String accountType;  
 private double accountBalance;  
  
 public BankAccount() {  
 super();  
 }  
  
 public int getAccountId() {  
 return accountId;  
 }  
  
 public void setAccountId(int accountId) {  
 this.accountId= accountId;  
 }  
  
 public String getAccountHolderName() {  
 return accountHolderName;  
 }  
  
 public void setAccountHolderName(String accountHolderName) {  
 this.accountHolderName= accountHolderName;  
 }  
  
 public String getAccountType() {  
 return accountType;  
 }  
  
 public void setAccountType(String accountType) {  
 this.accountType= accountType;  
 }  
  
 public double getAccountBalance() {  
 return accountBalance;  
 }  
  
 public void setAccountBalance(double accountBalance) {  
 this.accountBalance= accountBalance;  
 }  
  
 @Override  
 public String toString() {  
 return "BankAccount: [" + '\n' +  
 "accountId=" + accountId + '\n' +  
 ", accountHolderName='" + accountHolderName + '\n' +  
 ", accountType='" + accountType + '\n' +  
 ", accountBalance=" + accountBalance + '\n' +  
 ']';  
 }  
}

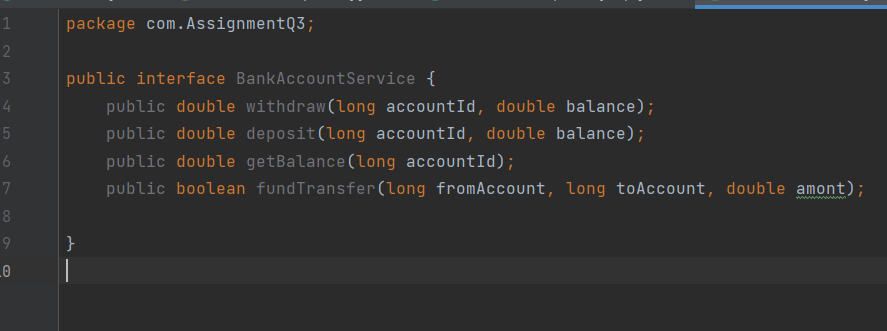
**Interface BankAccountRepository:**

****

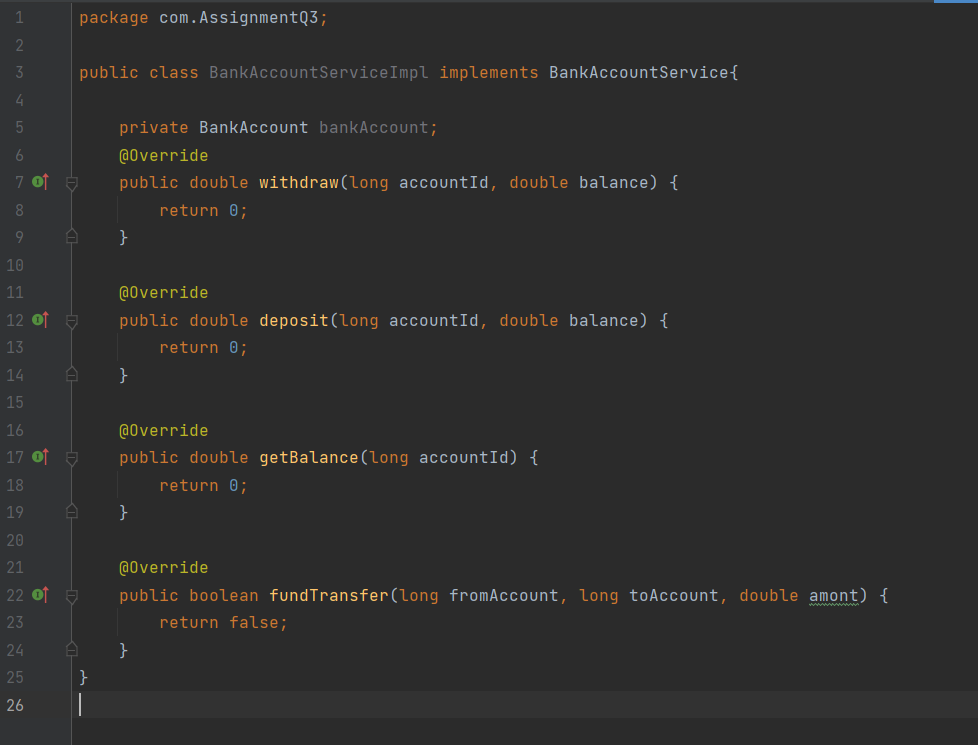
**BankAccountRepositoryImpl:**

package com.AssignmentQ3;  
  
public class BankAccountRepositoryImpl implements BankAccountRepository{  
  
 private BankAccount bankAccount1;  
 private BankAccount bankAccount2;  
  
 public BankAccountRepositoryImpl() {  
 }  
  
 public BankAccountRepositoryImpl(BankAccount bankAccount1, BankAccount bankAccount2) {  
 this.bankAccount1 = bankAccount1;  
 this.bankAccount2 = bankAccount2;  
 }  
  
 @Override  
 public double getBalance(long accountId) {  
 if(this.bankAccount1.getAccountId()==accountId){  
 return bankAccount1.getAccountBalance();  
 }  
 else {  
 return bankAccount2.getAccountBalance();  
 }  
 }  
  
 @Override  
 public double updateBalance(long accountId, double newBalance) {  
 double updateBal= 0;  
 if(this.bankAccount1.getAccountId()==accountId){  
 updateBal= newBalance;  
 bankAccount1.setAccountBalance(updateBal);  
 }  
 else {  
 updateBal= newBalance;  
 bankAccount2.setAccountBalance(updateBal);  
 }  
 return updateBal;  
 }  
}

**Interface BankAccountService:**

****

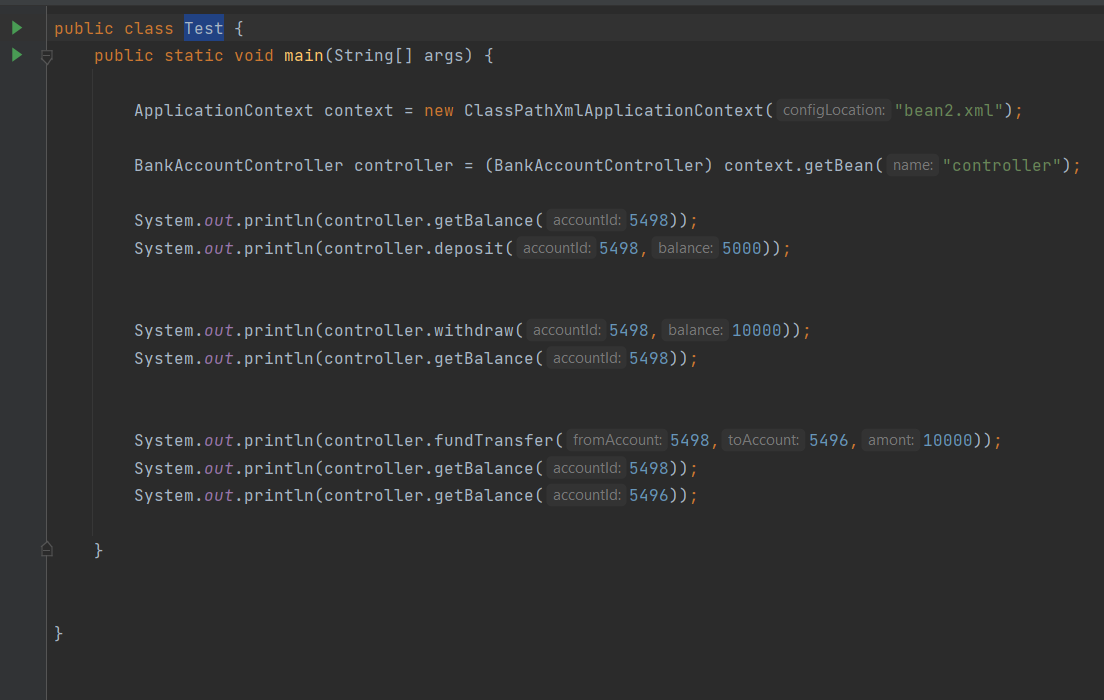
**Class BankAccountServiceImpl :**

****

**Class BankAccount Controller:**

package com.AssignmentQ3;  
  
import org.springframework.beans.BeansException;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.ApplicationContextAware;  
import org.springframework.context.support.AbstractApplicationContext;  
  
public class BankAccountController implements ApplicationContextAware{  
 private ApplicationContext context;  
  
 public double withdraw(long accountId, double balance){  
 double newBalance = 0;  
 BankAccountRepositoryImpl accountRepository= (BankAccountRepositoryImpl) context.getBean("BankRepo");  
  
 if(accountRepository.getBalance(accountId)>=balance){  
 newBalance = accountRepository.getBalance(accountId)-balance;  
 accountRepository.updateBalance(accountId,newBalance);  
 }  
 return newBalance;  
 }  
 public double deposit(long accountId, double balance){  
 BankAccountRepositoryImpl bankAccountRepository= context.getBean("BankRepo",BankAccountRepositoryImpl.class);  
 double newBalance = bankAccountRepository.getBalance(accountId)+balance;  
 return bankAccountRepository.updateBalance(accountId,newBalance);  
 }  
 public double getBalance(long accountId){  
 BankAccountRepositoryImpl bankAccountRepository= (BankAccountRepositoryImpl) context.getBean("BankRepo");  
 return bankAccountRepository.getBalance(accountId);  
 }  
 public boolean fundTransfer(long fromAccount, long toAccount, double amont){  
 BankAccountRepositoryImpl accountRepository= (BankAccountRepositoryImpl) context.getBean("BankRepo");  
 if(accountRepository.getBalance(fromAccount)>=amont){  
 double updateAmt= amont+accountRepository.getBalance(toAccount);  
  
 accountRepository.updateBalance(fromAccount,accountRepository.getBalance(fromAccount)-amont);  
 accountRepository.updateBalance(toAccount,accountRepository.getBalance(toAccount)+amont);  
 return true;  
 }  
 return false;  
 }  
 @Override  
 public void setApplicationContext(ApplicationContext applicationContext) throws BeansException{  
 this.context= applicationContext;  
 }  
}

**Test.java**

****

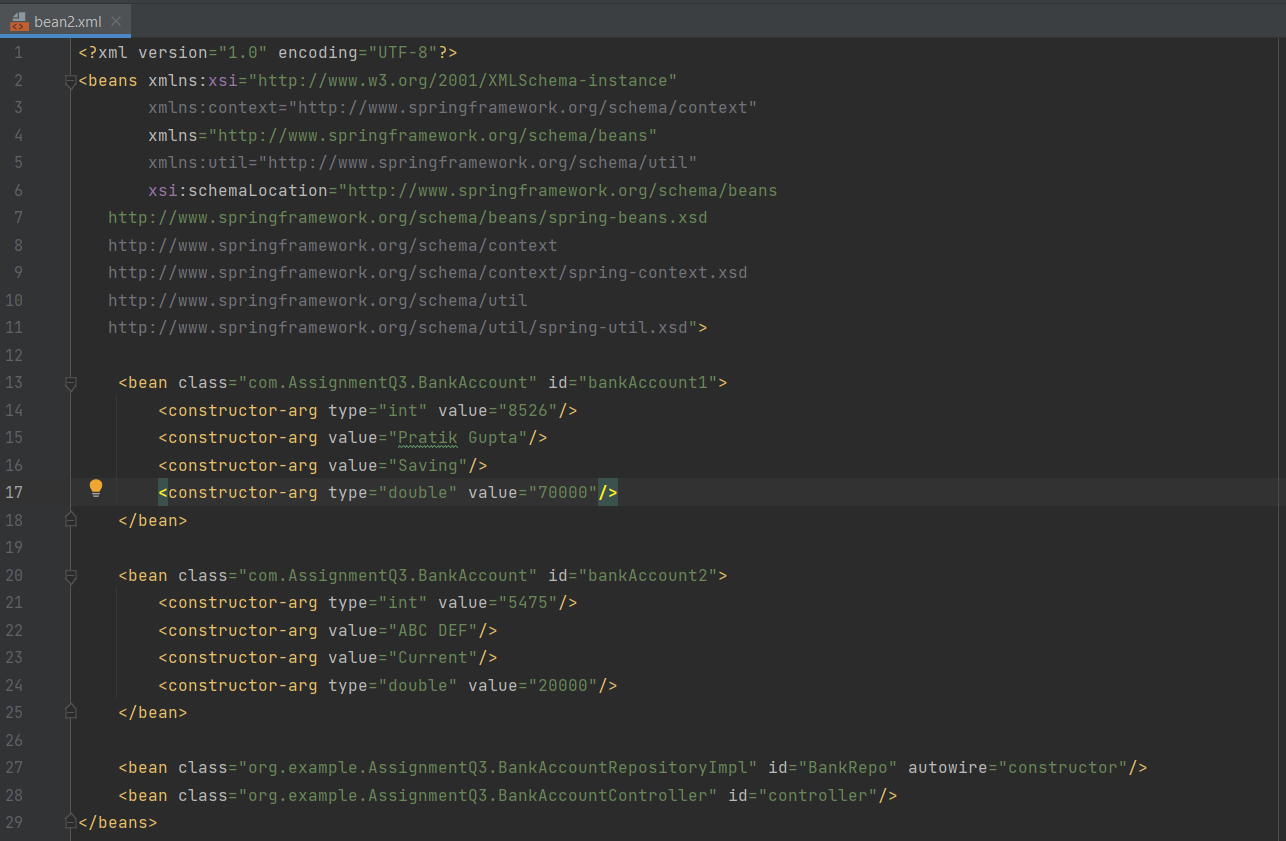
**Using byName:**

****

**Using byType:**

****

**Using constructor:**

****

4) Example on @Controller, @Service, @Repository, @Autowired, @Configuration and @Bean Modify the above application, use annotations and java based configuration.

**BankAccount.java**

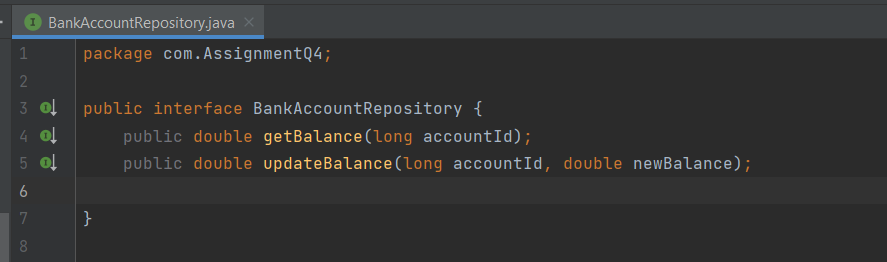
package com.AssignmentQ4;  
  
public class BankAccount {  
 private int accountId;  
 private String accountHolderName;  
 private String accountType;  
 private double accountBalance;  
  
 public BankAccount() {  
 super();  
 }  
  
 public BankAccount(int accountId, String accountHolderName, String accountType, double accountBalance) {  
 this.accountId= accountId;  
 this.accountHolderName= accountHolderName;  
 this.accountType= accountType;  
 this.accountBalance= accountBalance;  
 }  
  
 public int getAccountId() {  
 return accountId;  
 }  
  
 public void setAccountId(int accountId) {  
 this.accountId= accountId;  
 }  
  
 public String getAccountHolderName() {  
 return accountHolderName;  
 }  
  
 public void setAccountHolderName(String accountHolderName) {  
 this.accountHolderName= accountHolderName;  
 }  
  
 public String getAccountType() {  
 return accountType;  
 }  
  
 public void setAccountType(String accountType) {  
 this.accountType= accountType;  
 }  
  
 public double getAccountBalance() {  
 return accountBalance;  
 }  
  
 public void setAccountBalance(double accountBalance) {  
 this.accountBalance= accountBalance;  
 }  
  
 @Override  
 public String toString() {  
 return "BankAccount: [" + '\n' +  
 "accountId=" + accountId + '\n' +  
 ", accountHolderName='" + accountHolderName + '\n' +  
 ", accountType='" + accountType + '\n' +  
 ", accountBalance=" + accountBalance + '\n' +  
 ']';  
 }  
}

**BankAccountController.java**

package com.AssignmentQ4;  
  
import org.springframework.beans.BeansException;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.ApplicationContextAware;  
import org.springframework.stereotype.Service;  
  
@Service("service")  
public class BankAccountController implements ApplicationContextAware,BankAccountService{  
 private ApplicationContext context;  
  
 @Override  
 public double withdraw(long accountId, double balance){  
 double newBalance = 0;  
 BankAccountRepositoryImpl accountRepository= (BankAccountRepositoryImpl) context.getBean("BankRepo");  
  
 if(accountRepository.getBalance(accountId)>=balance){   
 newBalance = accountRepository.getBalance(accountId)-balance;  
 accountRepository.updateBalance(accountId,newBalance);  
 }  
 return newBalance;  
 }  
  
 @Override  
 public double deposit(long accountId, double balance){  
 BankAccountRepositoryImpl bankAccountRepository= context.getBean("BankRepo", BankAccountRepositoryImpl.class);  
 double newBalance = bankAccountRepository.getBalance(accountId)+balance;  
 return bankAccountRepository.updateBalance(accountId,newBalance);  
 }  
  
 @Override  
 public double getBalance(long accountId){  
 BankAccountRepositoryImpl bankAccountRepository= (BankAccountRepositoryImpl) context.getBean("BankRepo");  
 return bankAccountRepository.getBalance(accountId);  
 }  
  
 @Override  
 public boolean fundTransfer(long fromAccount, long toAccount, double amont){  
 BankAccountRepositoryImpl accountRepository= (BankAccountRepositoryImpl) context.getBean("BankRepo");  
 if(accountRepository.getBalance(fromAccount)>=amont){  
 double updateAmt= amont+accountRepository.getBalance(toAccount);  
  
 accountRepository.updateBalance(fromAccount,accountRepository.getBalance(fromAccount)-amont);  
 accountRepository.updateBalance(toAccount,accountRepository.getBalance(toAccount)+amont);  
 return true;  
 }  
 return false;  
 }  
 @Override  
 public void setApplicationContext(ApplicationContext applicationContext) throws BeansException{  
 this.context= applicationContext;  
 }

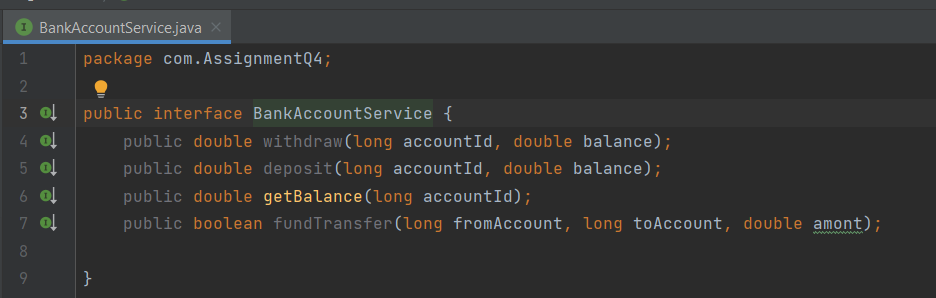
}

**BankAccountRepository.java**



package com.AssignmentQ4;  
  
import com.AssignmentQ3.BankAccount;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Repository;  
  
@Repository("BankRepo")  
public class BankAccountRepositoryImpl implements BankAccountRepository {  
  
 @Autowired  
 private BankAccount bankAccount1;  
 @Autowired  
 private BankAccount bankAccount2;  
  
 public BankAccountRepositoryImpl() {  
 }  
  
 public BankAccountRepositoryImpl(BankAccount bankAccount1, BankAccount bankAccount2) {  
 this.bankAccount1 = bankAccount1;  
 this.bankAccount2 = bankAccount2;  
 }  
  
 @Override  
 public double getBalance(long accountId) {  
 if(this.bankAccount1.getAccountId()==accountId){  
 return bankAccount1.getAccountBalance();  
 }  
 else {  
 return bankAccount2.getAccountBalance();  
 }  
 }  
  
 @Override  
 public double updateBalance(long accountId, double newBalance) {  
 double updateBal= 0;  
 if(this.bankAccount1.getAccountId()==accountId){  
 updateBal= newBalance;  
 bankAccount1.setAccountBalance(updateBal);  
 }  
 else {  
 updateBal= newBalance;  
 bankAccount2.setAccountBalance(updateBal);  
 }  
 return updateBal;  
 }  
}

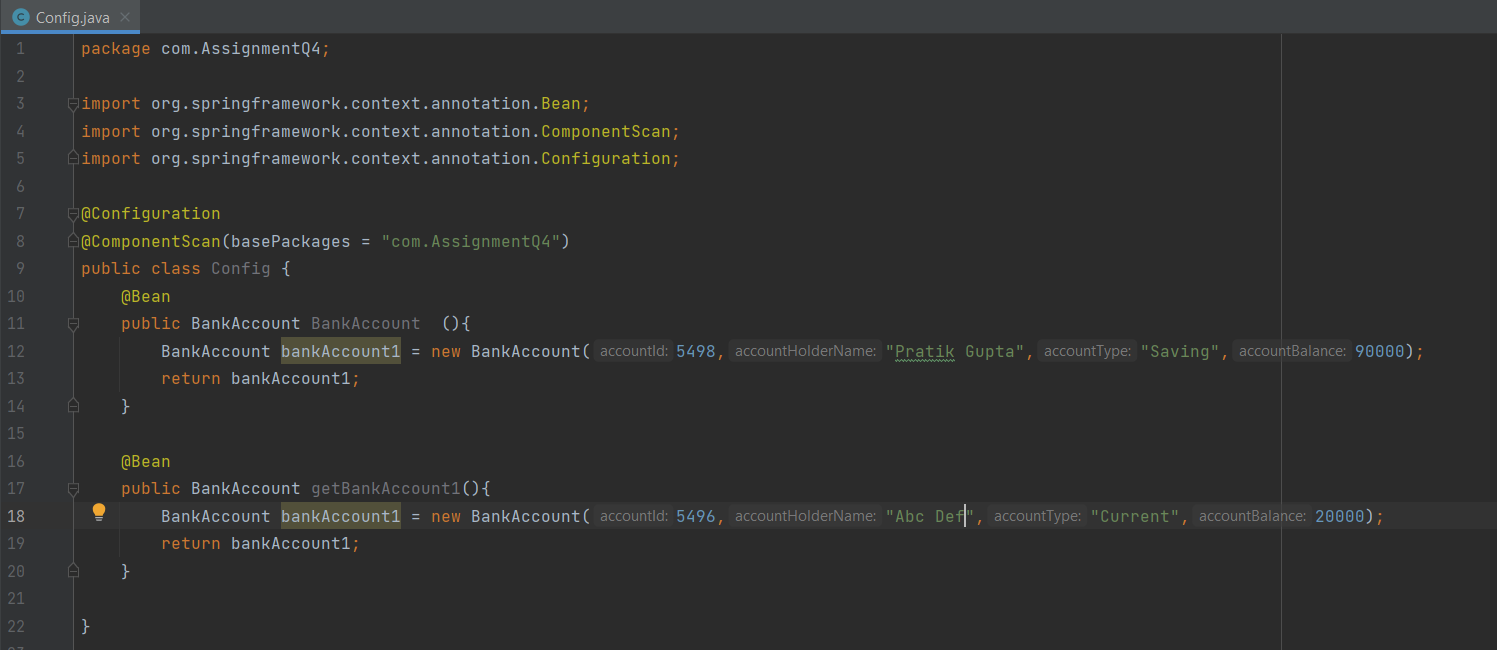
**BankAccountService.java**



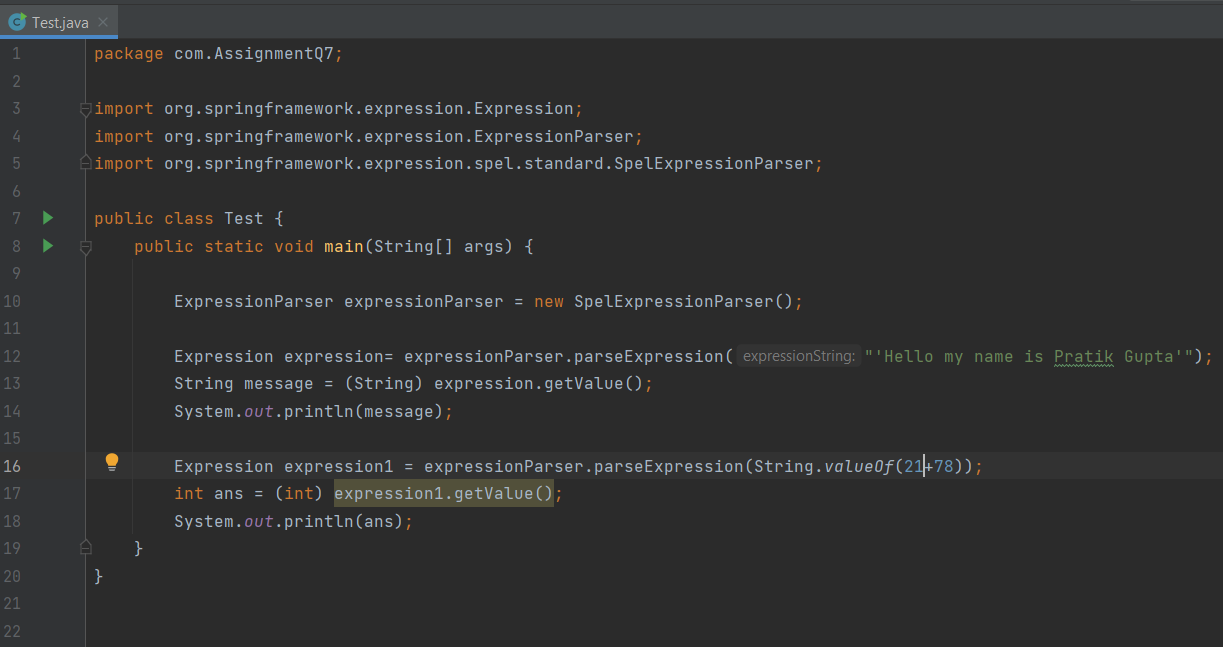
**BankAccountServiceImp.java**

package com.AssignmentQ4;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
public class BankAccountServiceImp implements BankAccountService {  
  
 @Autowired  
 private BankAccount bankAccount1;  
  
 @Autowired  
 private BankAccount bankAccount2;  
  
 public BankAccountServiceImp(){  
 }  
  
 public BankAccountServiceImp(BankAccount bankAccount1, BankAccount bankAccount2) {  
 this.bankAccount1 = bankAccount1;  
 this.bankAccount2 = bankAccount2;  
 }  
  
 @Override  
 public double withdraw(long accountId, double balance) {  
 return 0;  
 }  
  
 @Override  
 public double deposit(long accountId, double balance) {  
 return 0;  
 }  
  
 @Override  
 public double getBalance(long accountId) {  
 if(this.bankAccount1.getAccountId()==accountId){  
 return bankAccount1.getAccountBalance();  
 }  
 else {  
 return bankAccount2.getAccountBalance();  
 }  
 }  
  
 @Override  
 public boolean fundTransfer(long fromAccount, long toAccount, double amont) {  
 return false;  
 }  
}

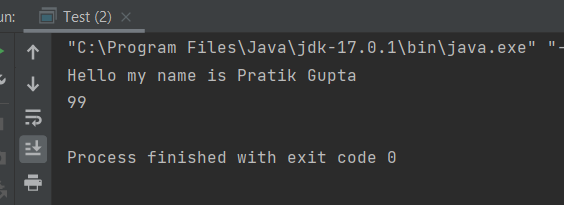
**Config.java**



7) Write a Java program to demonstrate SPEL (Spring Expression language)

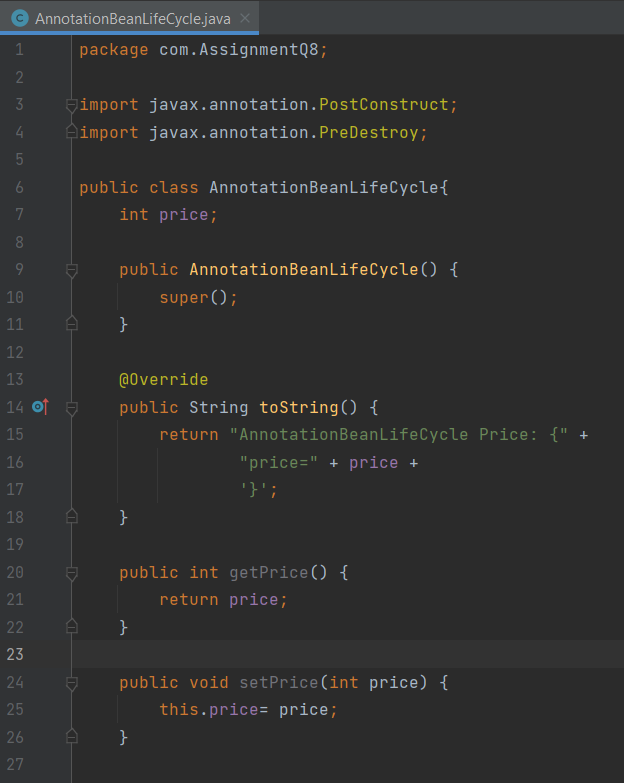
****

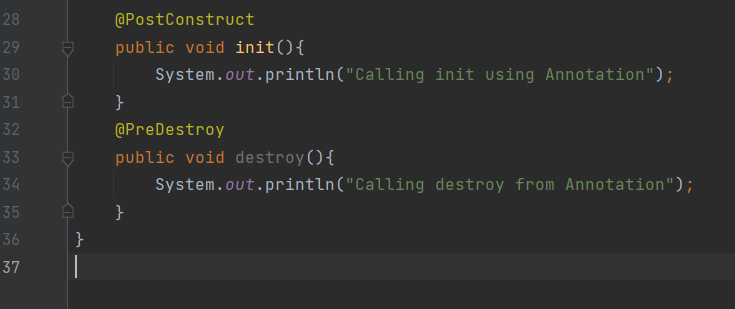
**OUTPUT :-**

****

8) Write a Java program to demonstrate InitializingBean and DisposableBean. Try Different ways: (Use init-method and destroy-method in xml config file) (Use @PostConstruct and @PreDestroy)

**AnnotationBeanLifeCycle.java**

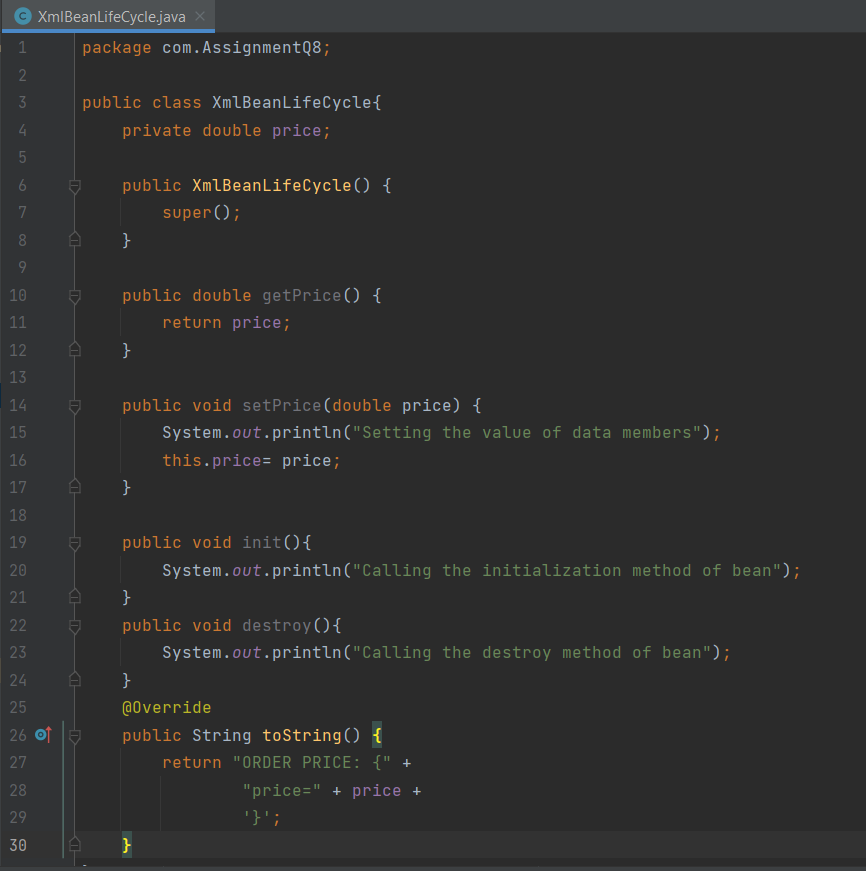
****

****

**InterfaceBeanLifeCycle.java**

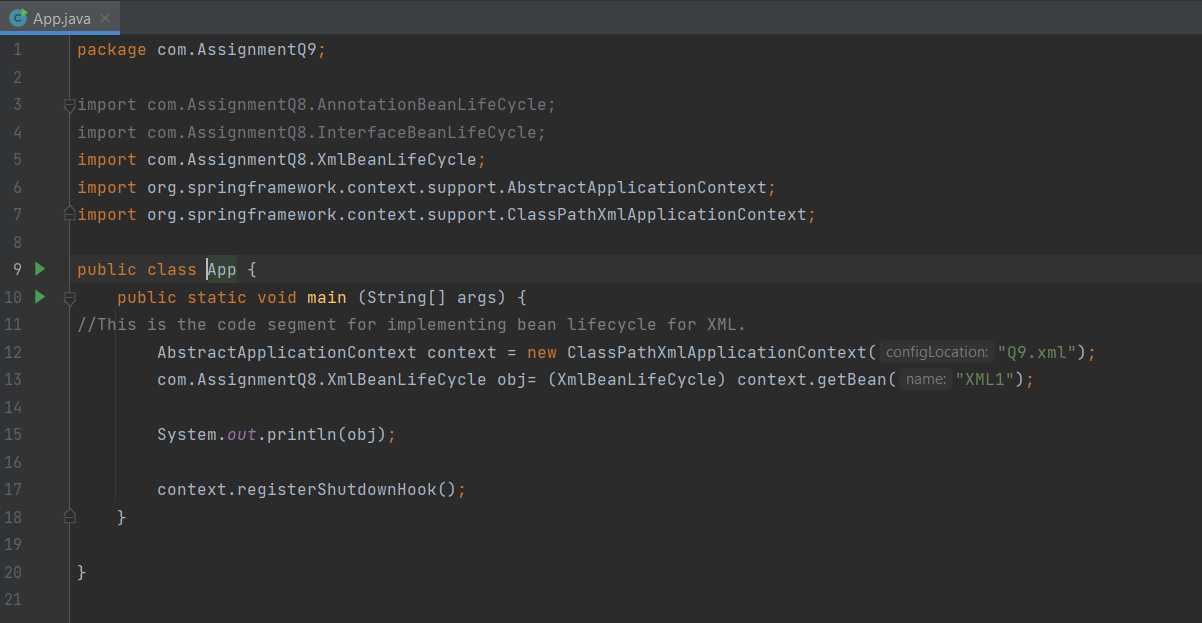
package com.AssignmentQ8;  
  
import org.springframework.beans.factory.DisposableBean;  
import org.springframework.beans.factory.InitializingBean;  
  
public class InterfaceBeanLifeCycle implements InitializingBean , DisposableBean{  
 private double price;  
  
 public InterfaceBeanLifeCycle() {  
 super();  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price= price;  
 }  
  
 @Override  
 public String toString() {  
 return "Interface Price: {" +  
 "price=" + price +  
 '}';  
 }  
  
 @Override  
 public void afterPropertiesSet() throws Exception {  
 System.*out*.println("Init method using interface");  
 }  
  
 @Override  
 public void destroy() throws Exception {  
 System.*out*.println("Destroy method using interface");  
 }  
}

**XmlBeanLifeCycle.java**

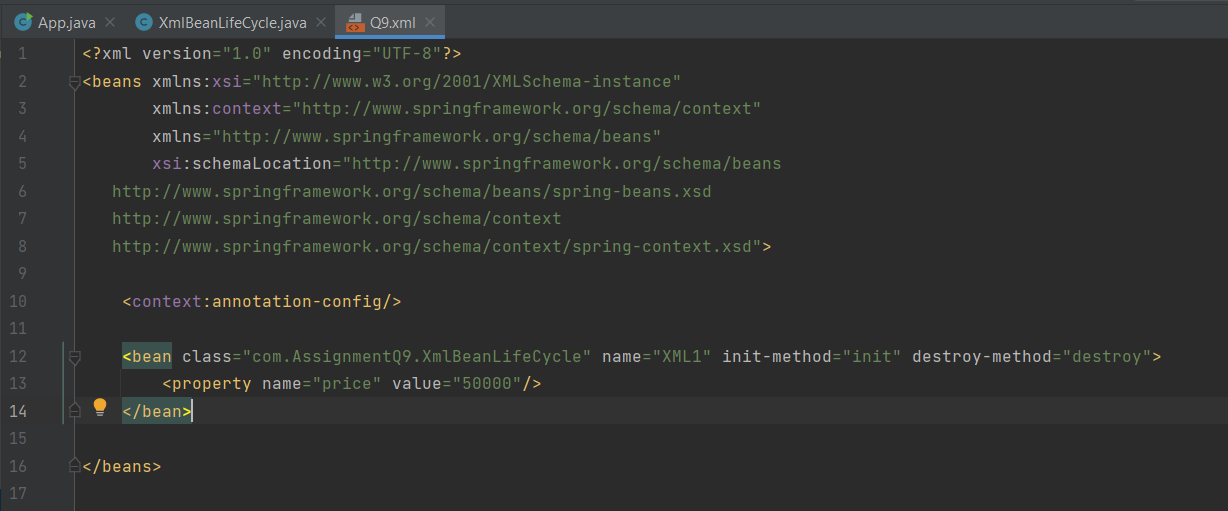
****

9) Write a Java program to demonstrate Complete Bean Life cycle.

App.java

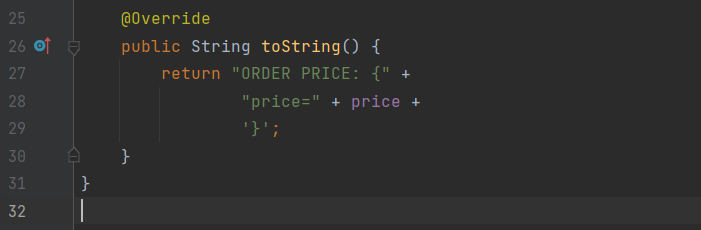


Q9.xml



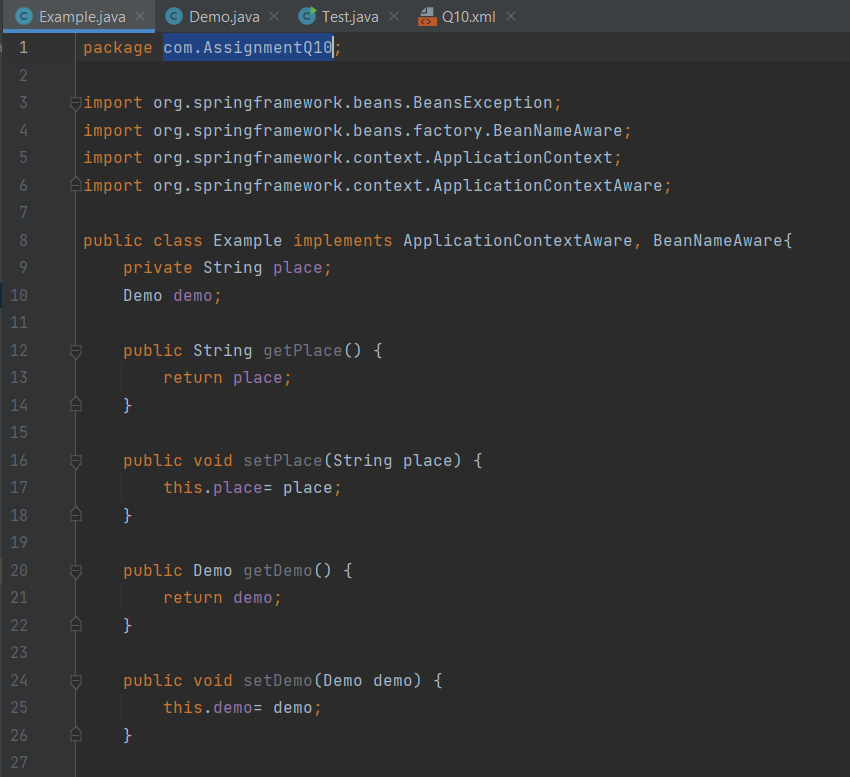
XmlBeanLifeCycle.java





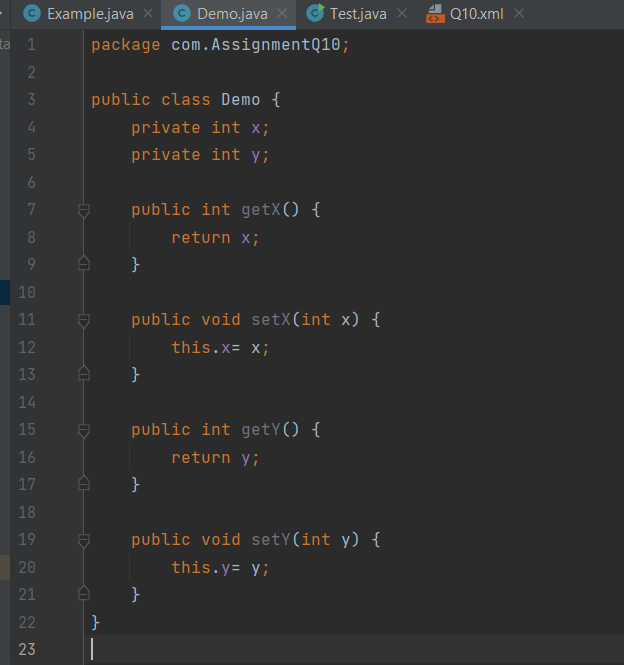
10) Write a java program to demonstrate ApplicationContextAware interface.

**Example.java**

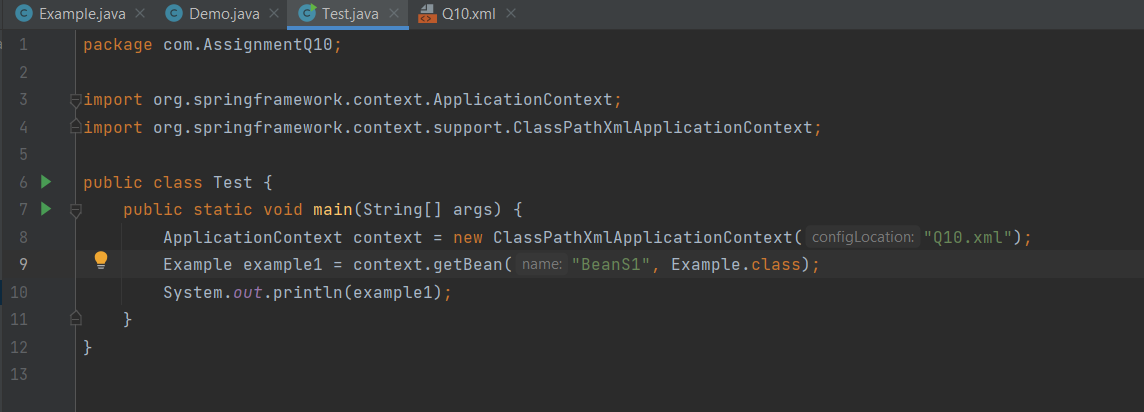
****

****

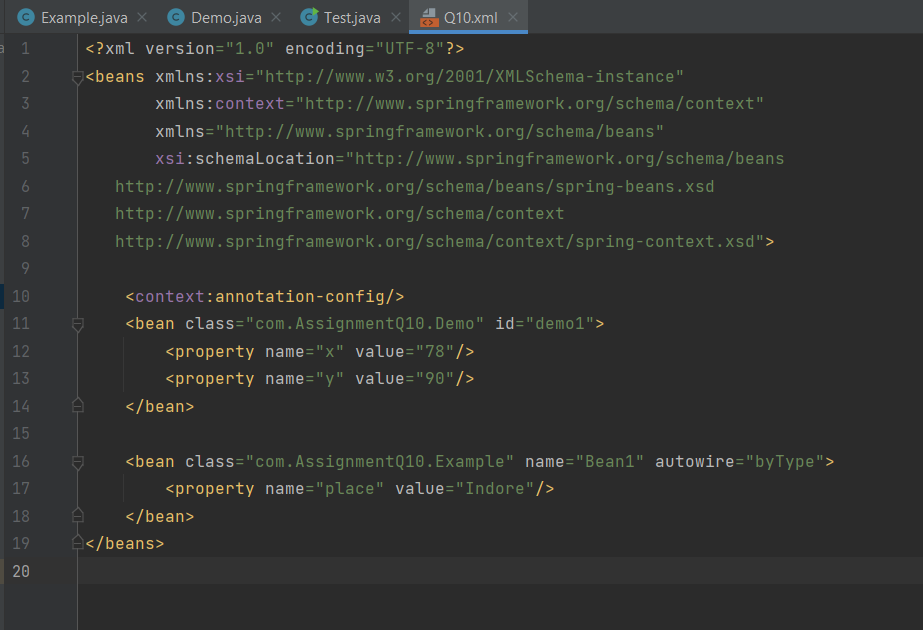
**Demo.java**

****

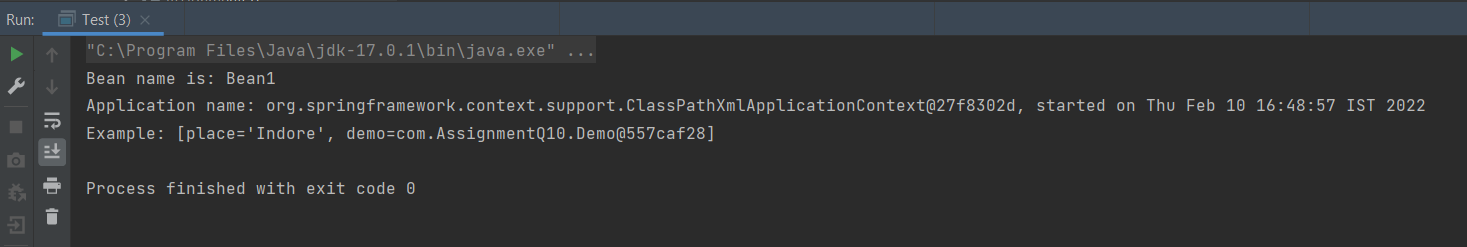
**Test.java**

****

**Q10.XML file:**

****

**Output :-**

****