SPRING CORE ASSIGNMENT

SECTION A

1) Create an Address class with the following attributes:- street, city, state, zip, country
2) Create an Customer class with the following attributes:- customerId, customerName, customerContact, customerAddress.
3) Inject the Address bean into Customer bean using setter injection
4) Create a Test class with main() method, get Customer bean from ApplicationContext object and print details of Customer.
5) Also write the JUnit Test cases for above program.
6) - Modify the above application and inject the bean using constructor injection
   - Use XML based Configuraion.

Address.java

```
public class Address {
	private String street;
	private String city;
	private String state;
	private String country;
	private int zip;

	//constructor
	public Address(String street,String city, String state, String country, int zip) {
		this.street = street;this.city=city;this.state=state;
		this.country=country;this.zip=zip;}
	//getters and setters
	public String getStreet() {return street;}
	/** public void setStreet(String street) { this.street = street; }*/
	public String getCity() {return city;}
	/** public void setCity(String city) { this.city = city; }*/
	public String getState() {return state;}
	/** public void setState(String state) { this.state = state; }*/
	public String getCountry() {return country;}
	/** public void setCountry(String country) { this.country = country; }*/
	public int getZip() {return zip;}
	/** public void setZip(int zip) { this.zip = zip; } */
}
```

Customer.java

```
public class Customer {
	private int customerId;
	private String customerName;
	private int customerContact;
	private String customerAddress;
	Address addressone;

	//getters and setters
	public int getCustomerId() {return customerId;}
	public void setCustomerId(int customerId) {this.customerId = customerId;}
	public String getCustomerName() {return customerName;}
```

```
	public void setCustomerName(String customerName) {this.customerName = customerName;}
	public int getCustomerContact() {return customerContact;}
	public void setCustomerContact(int customerContact) {this.customerContact = customerContact;}
	public String getCustomerAddress() {return customerAddress;}
	public void setCustomerAddress(String customerAddress) {this.customerAddress = customerAddress;}

	public void showInfo() {
		System.out.println("Customer ID: "+getCustomerId()+" Customer Name: "+ getCustomerName() +
"Customer Contact: "+getCustomerContact()+ "Customer Address: "+getCustomerAddress());
		System.out.println(addressone.getCity());
		System.out.println(addressone.getCountry());
		System.out.println(addressone.getState());
		System.out.println(addressone.getStreet());
		System.out.println(addressone.getZip());}
}
```

Test.java

```
public class Test {
	public static void main(String args[]) {
	ApplicationContext context = new ClassPathXmlApplicationContext("assignments.xml");
	Customer c= new Context.getBean("customer");
	c.showInfo();}
	}
```

Assignments.xml

```
<Beans>
	<bean id="customer" class="org.assignment.Customer">
		<property name="customerId" value="21"/>
		<property name="customerName" value="Lokesh"/>
		<property name="customerContact" value="9900660055"/>
		<property name="customerAddress" value="shaitaan gali"/>
		<property name="addressone" ref="address"/>
	</bean>
	<bean id="address" class="org.assignment.Address">
		<!-- <property name="street" value="new Street"/>
		<property name="city" value="ldh"/>
		<property name="state" value="PB"/>
		<property name="country" value="IN"/>
		<property name="zip" value="141005"/> -->
		<constructor-arg index="0" value="new Street"/>
		<constructor-arg index="1" value="Ldh"/>
		<constructor-arg index="2" value="PB"/>
		<constructor-arg index="3" value="IN"/>
		<constructor-arg index="4" value="141005"/>
	</bean>
</Beans>
```

TestTest.java

```
@Nested
```

```
@DisplayName("checking customer Class")
class TestTest {
	Customer c=new Customer();

	@Test
	void test() {
		assertEquals("shaitaan gali",c.getCustomerAddress());
		assertEquals("21",c.getCustomerId());
		assertEquals("Lokesh",c.getCustomerName());
		assertEquals("9900660055",c.getCustomerContact());
	}
}
```

SECTION B

Example of Injecting collections (List, Set and Map)

1) Create a class Question with following attributes: questionId, question, answers.
2) There are 3 cases for above program.
   a. Write a program where answers is of type List or String []
   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.
   f. Use XML based configuration.

Question.java

```
public class Question {
	private int questionid;
	private String question;
	//private List<String> answers;
	//private Set<String> answers;
	private Map<Integer,String> answers;
	//getters and setters
	public int getQuestionid() {return questionid;}
	public void setQuestionid(int questionid) {this.questionid = questionid;}
	public String getQuestion() {return question;}
	public void setQuestion(String question) {this.question = question;}
	/*public List<String> getAnswers() { return answers; }
	public void  setAnswers(List<String> answers) { this.answers = answers; }
	public Set<String> getAnswers() {return answers;}
	public void setAnswers(Set<String> answers) {this.answers = answers;}
	public void showInfo() {for(String ans: answers) {System.out.println(ans);}}*/
	for(Map.Entry m:map.entrySet()){System.out.println(m.getKey()+" "+m.getValue());}}}
```

Assignment.xml

```
<bean id="question" class="org.assignment.Question">
		<property name="questionid" value="1"/>
```

```
        <property name="question" value="what is java"/>
        <property name="answers">
        <!--   <List>
                    <value>java is a language</value>
                    <value>java is a technology</value>
                    <value>java is everything</value>
              </List> -->
        <Set>
              <value>java is a language</value>
              <value>java is a technology</value>
              <value>java is everything</value>
        </Set>
        <map>
              <entry key="1"  value="Java is a programming language"></entry>
              <entry key="2" value="Java is a Platform"></entry>
        </map>
        </property>
</bean>
```

Test.java

```
ApplicationContext context = new ClassPathXmlApplicationContext("assignments.xml");

Question q = (Question) Context.getBean("question");

q.showInfo();
```

SECTION C

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

BankAccount.java

```
package org.bank;

public class BankAccount {
	private int accountId,accountBalance;
	private String accountHolderName, accountType;
	//getters and setters
	public int getAccountId() {return accountId;}
	public void setAccountId(int accountId) {this.accountId = accountId;}
	public int getAccountBalance() {return accountBalance;}
	public void setAccountBalance(int accountBalance) {this.accountBalance = accountBalance;}
	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;}
}
```

BankAccountRepository.java

```
package org.bank;

public interface BankAccountRepository {
	public double getBalance(long accountId);
	public double updateBalance(long accountId, double newBalance);
}
```

BankAccountRepositoryimpl.java

```
package org.bank;

public class BankAccountepositoryImpl implements BankAccountRepository{
	BankAccount bankaccount;
	@Override
	public double getBalance(long accountId) {
		accountId=bankaccount.getAccountId();
		return bankaccount.getAccountBalance();
	}

	@Override
	public double updateBalance(long accountId, double newBalance) {
		accountId=bankaccount.getAccountId();
		return (bankaccount.getAccountBalance()+newBalance);
	}

}
```

BankAccountService.java

```java
package org.bank;

public interface BankAccountService {
	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);
}
```

BankAccountServiceImpl.java

```java
package org.bank;

public class BankAccountServiceImpl implements BankAccountService{
	BankAccount bankaccount;
	@Override
	public double withdraw(long accountId, double balance) {
		accountId=bankaccount.getAccountId();
		return (bankaccount.getAccountBalance()-balance);
	}
	@Override
	public double deposit(long accountId, double balance) {
		accountId=bankaccount.getAccountId();
		return (bankaccount.getAccountBalance()+balance);
	}
	@Override
	public double getBalance(long accountId) {
		accountId=bankaccount.getAccountId();
		return bankaccount.getAccountBalance();
	}
	@Override
	public boolean fundTransfer(long fromAccount, long toAccount, double amont) {
		fromAccount=(long)bankaccount.getAccountId();
		toAccount=(long)bankaccount.getAccountId();
		bankaccount.getAccountBalance();
		return true;
	}

}
```

BankAccountController.java

```java
package org.bank;

public class BankAccountcontroller {
	BankAccount bankaccount;
	public double withdraw(long accountId, double balance) {
		return bankaccount.getAccountBalance()-balance;
	}
	public double deposit(long accountId, double balance) {
		return bankaccount.getAccountBalance()+balance;
	}
```

```
	public double getBalance(long accountId) {
		return bankaccount.getAccountBalance();
	}
	public boolean fundTransfer(long fromAccount, long toAccount, double amont) {
		return true;
	}
}
```

Bank.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans>

	<bean id="BankAccount" class="org.bank.BankAccount" autowire="byName">
		<property name="accountId" value="100"/>
		<property name="accountBalance" value="10000"/>
		<property name="accountHolderName" value="jai prakash"/>
		<property name="accountType" value="savings"/>
	</bean>
	<bean id="bankaccountepositoryimpl" class="org.bank.BankAccountepositoryImpl"/>
	<bean id="bankaccountserviceimpl" class="org.bank.BankAccountServiceImpl"/>

</beans>
```

Test.java

```
package org.bank;
public class test {

	public static void main(String[] args) {
		ApplicationContext context = new ClassPathXmlApplicationContext("bank.xml");
		BankAccountcontroller b=(BankAccountcontroller)context.getBean("BankAccount");
		b.deposit(100, 100000);
		b.fundTransfer(100, 100, 5000);
		b.getBalance(100);
		b.withdraw(100, 50);
	}

}
```

SECTION D

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

Bankaccountcontroller.java

```
@Controller
public class BankAccountcontroller {

public void setBankaccount(BankAccount bankaccount) {
	this.bankaccount = bankaccount;
}}
```

BankAccountrepository.java

```
@Repository
public class BankAccountepositoryImpl implements BankAccountRepository{
	BankAccount bankaccount;
	@Autowired
	public void setBankaccount(BankAccount bankaccount) {
		this.bankaccount = bankaccount;
	}

}
```

Bankaccountservice.java

```
@Service
public class BankAccountServiceImpl implements BankAccountService{ }
```

```
@Configuration
public class BankAccount {
	private int accountId,accountBalance;
	private String accountHolderName, accountType;
	@Bean
	public void setAccountId(int accountId) {this.accountId = accountId;}
```

SECTION E

5) Write a program to demonstrate use of @Resource, @Inject, @Required annotations

```
  public class BankAccountcontroller {
	BankAccount bankaccount;
	//GETTERS and setters
	public BankAccount getBankaccount() {return bankaccount;}

	@Required
	@Resource(name="BankAccount")
	public void setBankaccount(BankAccount bankaccount) {
```

```
        this.bankaccount = bankaccount;
    }
```

SECTION F

Example of @Component, @Value, @PropertySource & Environment

a. Create a dbConfig.properties file which contains database configuration details like driver class name, dburl, username, password.

b. Create a Java class in which you have to read all properties and display on a console. (Use @Component, @Value or Environment and @PropertyResource).

SECTION G

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

```
import org.springframework.expression.Expression;
import org.springframework.expression.ExpressionParser;
import org.springframework.expression.spel.standard.SpelExpressionParser;
public class Test {
public static void main(String[] args) {
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Hello SPEL'");
String message = (String) exp.getValue();
System.out.println(message);  } }
```

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)

InitializingBean and DisposableBean

Test.java

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

```
		Customer c= (Customer)Context.getBean("customer");
		c.showInfo();}}
```

Customer.java

```
public class Customer implements InitializingBean, DisposableBean{
public void afterPropertiesSet() throws Exception{
	System.out.println("Initialising bean through propertySet");     }
	public void destroy() throws Exception{
		System.out.println("Destruction of bean through destroy method");}}
@PostConstruct
	public void init() {
		System.out.println("init method"); }
	@PreDestroy
	public void destroy() {
		System.out.println("destroy method");   }
```

assignments.xml

```
<context:annotation-config/>
<bean id="customer" class="org.assignment.Customer" init-method="init" destroy-method="destoy">
```

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

assignments.xml

```
<bean id="address" class="org.assignment.Address" init-method="init" destroy-method="destoy"/>
```

address.java

```
public class Address{
public void init() {
	System.out.println("init method"); }
```

```
public void destroy() {
        System.out.println("destroy method");   }
}
```

10) Write a java program to demonstrate ApplicationContextAware interface

```
public class Address implements ApplicationContextAware{
       ApplicationContext context= null;
       @Override
       public void setApplicationContext(ApplicationContext context) throws BeanException{
              this.context= context;
       }
```