Note:

All the questions are compulsory.
The duration of the test is 3 hours
Mode of Submission: Github Link
Don’t seek help from any person/resource during the test.

Marks Distribution is as follows:

Q1: 2 Marks
Q2: 3 Marks
Q3: 3 Marks
Q4: 2 Marks

Q1/- What do you mean by the IOC, explain the life cycle of the Spring Bean.

Q2/- Consider the following bean classes:

class Student:

roll:
name:
address:
email:
marks:

Class Course:

courseId:
courseName:
duration:
fee:

Write a Spring application to inject the following type of Dependency inside StudentService class.

class StudentService{

private Map theMap<Student, Course>; // inject 3 entries with valid details

private List<Student> theList; //inject List of 5 Student object

private String appName; //inject the AppName from the properties file

//Hint: Make use of @Bean annotation to inject theMap and theList;

public void printMap(){

//print all the student's and their course details from theMap

}

public void printList(){

//sort the List of Student according to the marks (make use of Lamda //expression).

//print all the sorted Student Details

}

public void printAppName(){

//print the injected appName

}

- **Inside the main method of the Demo class pull the StudentService class object and call all the methods of the StudentService class.**
- **To implement the application make use of Spring Annotation approach.**

Q3/- Design a Spring Application of Product Management using Layered Architecture:

Product.java:

productId:
productName:
quantity:
price:

ProductService.java(Interface) : //Service Layer, apply @Service annotation to the implementation class.

1. public boolean addProduct(Product product);
2. public List<Product> getAllProducts();
3. public Prodcut getProductById(int productId)throws ProductException
4. Public List<Product> getProductsBetweenPrice(int fromPrice, int toPrice)throws ProductException

Note: ProductException Should be the checked exception.

ProductRepo.java(Interface): Data Access Layer, apply @Repository annotation to the implementation class.

1. public boolean insertProductDetails(Product product);
2. public List<Product> getAllProductDetails();
3. public Product findProduct(int productId);
4. Public List<Product> getProductInPriceRange(int fromPrice, int toPrice);

Note: provide the implementation of the above ProductRepo interface using JPA with Hibernate:

```
class Presentation{    // Apply @Controller annotation to this class

    private ProductService pService; // dependency


    public void insertProduct(){

        //take the input from the user (Product Details without productId, productId
        //should be generated automatically)and call the appropriate method on
        the //pService object.

    }
```

```
    public void printAllProduct(){

        //call the getAllProducts method on the pService object and print all
        the //product details.

    }

    public void searchProduct(){

        // take the empId from the user and call the getProductById method
        on the //pService obj. And print the appropriate details.

    }

    public void GetProductsWithingPriceRange(){

        // take the price range (fromPrice and toPrice) from the user and call
        //the appropriate method on //pService object and print the Product
        //within the price range,

    }

}
```

Note:

- **Make use of the annotation approach.**
- **ProductService should have a dependency on ProductRepo.**
- **Inside the main method of the Demo class pull the Presentation class object and call each method one by one.**

Sample of persistence.xml: do the changes accordingly

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

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
        version="2.0">

   <persistence-unit name="productUnit" >


        <properties>

       <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
```

```
        <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/productdb" />
        <property name="javax.persistence.jdbc.user" value="" />
        <property name="javax.persistence.jdbc.password" value="" />

         <property name="hibernate.show_sql"  value="true"/>

         <property name="hibernate.hbm2ddl.auto"  value="update"/>

      </properties>

   </persistence-unit>

</persistence>
```

Sample of pom.xml: do the changes accordingly

```
 <properties>
   <maven.compiler.source>1.8</maven.compiler.source>
   <maven.compiler.target>1.8</maven.compiler.target>
 </properties>


<dependencies>


        <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.22</version>
        </dependency>



          <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.12.Final</version>
        </dependency>

          <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.28</version>
        </dependency>


  </dependencies>
```

Note: change the MySQL dependency version accordingly

Q4/- Create a spring project named Drawshapes with which we are able to draw different geometrical shapes. Having classes Triangle, Circle, Cylinder, etc. With a common function draw() in each of them which will print ("You have drawn Triangle") or YOUR_SHAPE.

Make the main Runner class(main class) from which we can draw any shape we want

to by keeping the classes loosely coupled. And you have to configure spring IOC for

dependency injection using xml approach and make use of Dependency injection using setter injection.

Sample of Spring configuration file: applicationContext.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
https://www.springframework.org/schema/beans/spring-beans.xsd">



 </beans>
```