|  |  |  |
| --- | --- | --- |
| **No** | **Question** | **Expected Answer** |
| 1 | **Description of the given use-case:**  Refactor the *AlertService*and *MapAlertDAO*classes:   * Create a new *package-private* interface, named *AlertDAO*, that contains the same methods as *MapAlertDAO*. * *MapAlertDAO*should implement the *AlertDAO*interface. * *AlertService*should have a constructor that accepts *AlertDAO*. * The *raiseAlert*and *getAlertTime*methods should use the object passed through the constructor.   **Code Snippet:**  import java.util.Date; import java.util.HashMap; import java.util.Map; import java.util.UUID;  class AlertService { private final MapAlertDAO storage = new MapAlertDAO();  public UUID raiseAlert() { return this.storage.addAlert(new Date()); }  public Date getAlertTime(UUID id) { return this.storage.getAlert(id); }  }  class MapAlertDAO { private final Map<UUID, Date> alerts = new HashMap<UUID, Date>();  public UUID addAlert(Date time) { UUID id = UUID.randomUUID(); this.alerts.put(id, time); return id; }  public Date getAlert(UUID id) { return this.alerts.get(id); }  } | **Solution:**  **package** com.ubique.javatest;  **import** java.util.Date;  **import** java.util.UUID;  **class** AlertService {  **private** **final** MapAlertDAO storage = **new** MapAlertDAO();  **public** UUID raiseAlert() {  **return** **this**.storage.addAlert(**new** Date());  }  **public** Date getAlertTime(UUID id) {  **return** **this**.storage.getAlert(id);  }  }  **package** com.ubique.javatest;  **import** java.util.Date;  **import** java.util.UUID;  /\*\*  \* **@author** Liyakath Ali  \*  \*/  **public** **interface** AlertDAO {    **public** UUID addAlert(Date time);    **public** Date getAlert(UUID id);  }  **package** com.ubique.javatest;  **import** java.util.Date;  **import** java.util.UUID;  **class** AlertService {  **private** **final** MapAlertDAO storage = **new** MapAlertDAO();  **public** AlertService(AlertDAO alertDAO) {  **super**();  }  **public** UUID raiseAlert() {  **return** **this**.storage.addAlert(**new** Date());  }  **public** Date getAlertTime(UUID id) {  **return** **this**.storage.getAlert(id);  }  } |
| 2 | Question(Medium level): Given below is a request JSON. Each object in request json has currency and amount field in it. The currency and amount are repeated as you can see in request JSON. Task is to write an api which accepts the this request JSON and responds back with distinct currency along with sum of respective amounts against the currency.  Request JSON:  [?](https://confluence.maerskdev.net/display/IXN/Backend+Developer+Technical+Test)   |  | | --- | | [    {      "currency":  "USD",      "amount": "10"  },    {      "currency":  "USD",      "amount": "20"  },    {      "currency":  "USD",      "amount": "30"  },    {      "currency":  "EUR",      "amount": "10"  },    {      "currency":  "INR",      "amount": "10"  },    {      "currency":  "EUR",      "amount": "10"  },    {      "currency":  "INR",      "amount": "60"    },    {      "currency":  "EUR",      "amount": "20"    }  ] |   If you look at the above json, the currency INR is repeated 2 time, with amount 10 and 60 INR. hence the output below is 10+60=70 against INR.  Response JSON:  [?](https://confluence.maerskdev.net/display/IXN/Backend+Developer+Technical+Test)   |  | | --- | | {     "EUR": 40,     "USD": 60,     "INR": 70  } | |  |
| 3 | **Description of the given use-case:**  Implement the function*findRoots*to find the roots of the quadratic equation: ax2+ bx + c = 0. If the equation has only one solution, the function should return that solution as both elements of the *Roots*. The equation will always have at least one solution.  The roots of the quadratic equation can be found with the following formula:A quadratic equation.  For example, the roots of the equation 2x2+ 10x + 8 = 0 are -1 and -4.  **Code Snippet:**  public class QuadraticEquation { public static Roots findRoots(double a, double b, double c) { throw new UnsupportedOperationException("Waiting to be implemented."); }  public static void main(String[] args) { Roots roots = QuadraticEquation.findRoots(2, 10, 8); System.out.println("Roots: " + roots.x1 + ", " + roots.x2); } }  class Roots { public final double x1, x2;  public Roots(double x1, double x2) {  this.x1 = x1; this.x2 = x2; } } | **Solution:**  **package** com.ubique.javatest;  **import** **static** java.lang.Math.*abs*;  **import** **static** java.lang.Math.*sqrt*;  **class** Roots {  **public** **final** **double** x1, x2;  **public** Roots(**double** x1, **double** x2) {  **this**.x1 = x1;  **this**.x2 = x2;  }  }  **public** **class** QuadraticEquation {  **public** **static** Roots findRoots(**double** a, **double** b, **double** c) **throws** IllegalArgumentException {  // If a is 0, then equation is not  // quadratic, but linear  **if** (a == 0) {  System.***out***.println("Invalid");  }  **double** result1 = 0;  **double** result2 = 0;  **double** d = b \* b - 4 \* a \* c;  **double** sqrt\_val = *sqrt*(*abs*(d));  **if** (d > 0) {  result1 = (**double**) (-b + sqrt\_val) / (2 \* a);  result2 = (**double**) (-b - sqrt\_val) / (2 \* a);  } **else** // d < 0  {  System.***out***.println("Roots are complex \n");  System.***out***.println(  -(**double**) b / (2 \* a) + " + i" + sqrt\_val + "\n" + -(**double**) b / (2 \* a) + " - i" + sqrt\_val);  }  **return** **new** Roots(result1, result2);  }  **public** **static** **void** main(String[] args) {  Roots roots = QuadraticEquation.*findRoots*(2, 10, 8);  **try** {  System.***out***.println("Roots: " + roots.x1 + ", " + roots.x2);  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |
| 1 | **Description of the given use-case:**  Modify the *createMovie*and *findMoviesByName* methods of the class *MovieRepository*.   * Using *JdbcTemplate*, *createMovie*method should insert a new row into the movies table with the specified data. * Using *JdbcTemplate, findMoviesByName*should return a list of all movies from the 'movies' table, in which the name of the movie contains the *likeName*string, using the LIKE operator. The returned list should contain *Movie*objects with name, year, and rating fields correctly set.   For example, executing the following code:  **Code Snippet:**  **AnnotationConfigApplicationContext config = new AnnotationConfigApplicationContext(); config.register(Config.class); config.refresh(); MovieRepository repository = config.getBean(MovieRepository.class); repository.createMovie("Some movie", 1974, 3); repository.createMovie("Some other movie", 1993, 2); List<Movie> movies = repository.findMoviesByName("Some%"); for(Movie movie : movies) { System.out.println(**[**movie.name**](http://movie.name/) **+ " - " + movie.year + " - " + movie.rating); }**  **It should print:**  Some movie - 1974 - 3  Some other movie - 1993 - 2  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.\*; import org.springframework.jdbc.core.\*; import org.springframework.jdbc.datasource.DriverManagerDataSource; import org.springframework.stereotype.Repository; import javax.annotation.PostConstruct; import java.sql.ResultSet; import java.sql.SQLException; import java.util.\*;  class Movie { public String name; public int year; public int rating;  public Movie(String name, int year, int rating) { [this.name](http://this.name/) = name; this.year = year; this.rating = rating; } } @Configuration @Import(MovieRepository.class) class Config { @Bean public DriverManagerDataSource dataSource() { DriverManagerDataSource ds = new DriverManagerDataSource(); ds.setDriverClassName("org.h2.Driver"); ds.setUrl("jdbc:h2:mem:test;DB\_CLOSE\_DELAY=-1"); return ds; } @Bean public JdbcTemplate jdbcTemplate(DriverManagerDataSource ds) { return new JdbcTemplate(ds); } } @Repository public class MovieRepository {  @Autowired private JdbcTemplate template;  @PostConstruct public void createTable() { template.execute("CREATE TABLE movies (id bigint auto\_increment primary key, name VARCHAR(50), year int, rating int)"); }  public void createMovie(String name, int year, int rating) {  }  public List<Movie> findMoviesByName(String likeName) { return null; }  public static void main(String[] args) { AnnotationConfigApplicationContext config = new AnnotationConfigApplicationContext(); config.register(Config.class); config.refresh(); MovieRepository repository = config.getBean(MovieRepository.class);  repository.createMovie("Some movie", 1974, 3); repository.createMovie("Some other movie", 1993, 2);  List<Movie> movies = repository.findMoviesByName("Some%"); for(Movie movie : movies) { System.out.println([movie.name](http://movie.name/) + " - " + movie.year + " - " + movie.rating); } } } | **Solution:**  //Modify the createMovie and findMoviesByName methods of the class MovieRepository.  //• Using JdbcTemplate, createMovie method should insert a new row into the movies table with the specified data.  //• Using JdbcTemplate, findMoviesByName should return a list of all movies from the 'movies' table, in which the name of the movie contains the likeName string, using the LIKE operator. The returned list should contain Movie objects with name, year, and rating fields correctly set.  package com.ubique.javatest;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Types;  import java.util.Collections;  import java.util.List;  import java.util.stream.Collectors;  import javax.annotation.PostConstruct;  import javax.sql.DataSource;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.\*;  import org.springframework.jdbc.core.\*;  import org.springframework.jdbc.datasource.DriverManagerDataSource;  import org.springframework.stereotype.Repository;  class Movie {  public String name;  public int year;  public int rating;  public Movie(String name, int year, int rating) {  this.name = name;  this.year = year;  this.rating = rating;  }  }  @Configuration  @Import(MovieRepository.class)  class Config {  @Bean  public DriverManagerDataSource dataSource() {  DriverManagerDataSource ds = new DriverManagerDataSource();  ds.setDriverClassName("org.h2.Driver");  ds.setUrl("jdbc:h2:mem:test;DB\_CLOSE\_DELAY=-1");  return ds;  }  @Bean  public JdbcTemplate jdbcTemplate(DriverManagerDataSource ds) {  return new JdbcTemplate(ds);  }  }  @Repository  public class MovieRepository {  @Autowired  private JdbcTemplate template;  @Autowired  private DataSource dataSource;  @PostConstruct  public void createTable() {  template.execute(  "CREATE TABLE movies (id bigint auto\_increment primary key, name VARCHAR(50), year int, rating int)");  }  /\*  \* public void createMovie(String name, int year, int rating) {  \*  \* }  \*/  public void createMovie(String name, int year, int rating) {  final String insertSql =  "INSERT INTO movies (" +  " name, " +  " year, " +  " rating) " +  "VALUES (?, ?, ?, ?)";  JdbcTemplate template = new JdbcTemplate(dataSource);  // define query arguments  Object[] params = new Object[] { name, year, rating };  // define SQL types of the arguments  int[] types = new int[] { Types.VARCHAR, Types.INTEGER, Types.INTEGER };  // execute insert query to insert the data  // return number of row / rows processed by the executed query  int row = template.update(insertSql, params, types);  System.out.println(row + " row inserted.");  }  public List<Movie> findMoviesByName(String name) {  JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);  return jdbcTemplate.query("select name, year, rating from movies where name like ?", new Object[] { name },  new RowMapper<Movie>() {  @Override  public Movie mapRow(ResultSet resultSet, int i) throws SQLException {  return new Movie(resultSet.getString("name"), resultSet.getInt("year"),  resultSet.getInt("rating"));  }  });  }  public static void main(String[] args) {  AnnotationConfigApplicationContext config = new AnnotationConfigApplicationContext();  config.register(Config.class);  config.refresh();  MovieRepository repository = config.getBean(MovieRepository.class);  repository.createMovie("Some movie", 1974, 3);  repository.createMovie("Some other movie", 1993, 2);  List<Movie> movies = repository.findMoviesByName("Some%");  for (Movie movie : movies) {  System.out.println(movie.name + " - " + movie.year + " - " + movie.rating);  }  }  } |
| 2 | **Description of the given use-case:**  Edit the WeatherForecastService, *FakeThermometer,*and the *Config* classes so that:   * The Spring container should always return a new instance of *FakeThermometer.* * *Config*class will configure Spring scheduling. * *takeTemperatureMeasurement*method should be executed every 50 milliseconds, using Spring scheduling.   import org.springframework.beans.factory.annotation.Autowired; import org.springframework.beans.factory.config.ConfigurableBeanFactory; import org.springframework.context.annotation.\*; import org.springframework.scheduling.\*; import org.springframework.scheduling.annotation.\*; import org.springframework.scheduling.concurrent.\*; import org.springframework.stereotype.\*;  @FunctionalInterface interface TemperatureMeasurementCallback { public void temperatureMeasured(int temperature); } interface Thermometer { public int measure(); }  @Configuration @Import({FakeThermometer.class, WeatherForecastService.class}) class Config {  @Bean public TemperatureMeasurementCallback callback() { return (temperature) -> System.out.println(temperature); } }  @Component class FakeThermometer implements Thermometer {  private int currentTemperature = 21;  @Override public int measure() { return currentTemperature++; } }  @Service public class WeatherForecastService {  @Autowired private Thermometer thermometer; @Autowired private TemperatureMeasurementCallback callback;  public void takeTemperatureMeasurement() { int temperature = thermometer.measure(); callback.temperatureMeasured(temperature); }  } | **Solution:**  /\*Edit the WeatherForecastService, FakeThermometer, and the Config classes so that:  • The Spring container should always return a new instance of FakeThermometer.  • Config class will configure Spring scheduling.  • takeTemperatureMeasurement method should be executed every 50 milliseconds, using Spring scheduling.\*/  **package** com.ubique.javatest;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.annotation.Import;  **import** org.springframework.context.support.PropertySourcesPlaceholderConfigurer;  **import** org.springframework.scheduling.annotation.Scheduled;  **import** org.springframework.stereotype.Component;  **import** org.springframework.stereotype.Service;  @FunctionalInterface  **interface** TemperatureMeasurementCallback {  **public** **void** temperatureMeasured(**int** temperature);  }  **interface** Thermometer {  **public** **int** measure();  }  @Configuration  @Import({ FakeThermometer.**class**, WeatherForecastService.**class** })  **class** Config1 {  @Bean  **public** TemperatureMeasurementCallback callback() {  **return** (temperature) -> System.***out***.println(temperature);  }  @Bean  **public** **static** PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {  **return** **new** PropertySourcesPlaceholderConfigurer();  }  }  @Component  **class** FakeThermometer **implements** Thermometer {  **private** **int** currentTemperature = 21;  **public** **int** measure() {  **return** currentTemperature++;  }  }  @Service  **public** **class** WeatherForecastService {  @Autowired  **private** Thermometer thermometer;  @Autowired  **private** TemperatureMeasurementCallback callback;  @Autowired  **public** WeatherForecastService(Thermometer thermometer) {  **this**.thermometer = thermometer;  }  @Scheduled(fixedDelay = 10000)  **public** **void** takeTemperatureMeasurement() {  **int** temperature = thermometer.measure();  callback.temperatureMeasured(temperature);  }  } |