Java 2

rajdeep777acharya@gmail.com Switch accounts

Draft saved

*Required

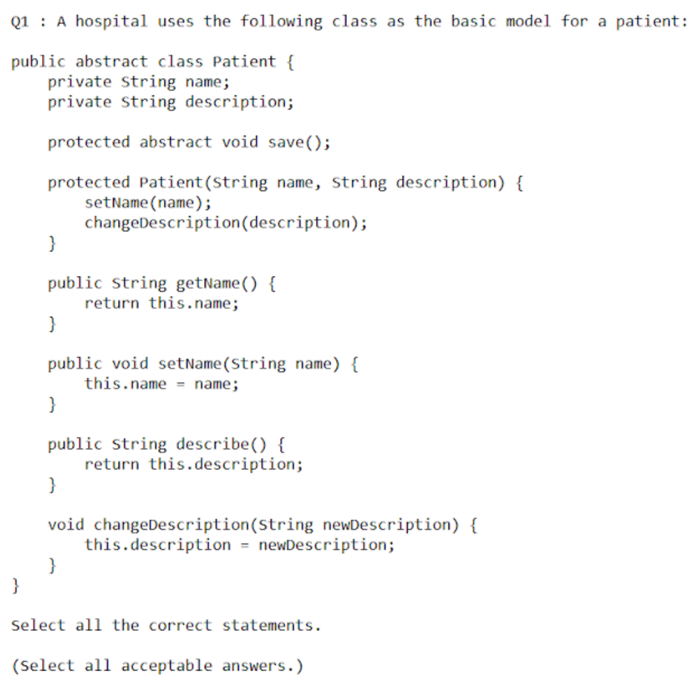
Email *

rajdeep777acharya@gmail.com

Name *

Rajdeep Acharya

1. A hospital uses the following class as the basic model for a patient:



```
Q1 : A hospital uses the following class as the basic model for a patient:

public abstract class Patient {
    private String name;
    private String description;

    protected abstract void save();

    protected Patient(String name, String description) {
        setName(name);
        changeDescription(description);
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String describe() {
        return this.description;
    }

    void changeDescription(String newDescription) {
        this.description = newDescription;
    }
}

Select all the correct statements.

(Select all acceptable answers.)
```

[✓ 1. Non-abstract classes inheriting directly from Patient must provide their implementation of the save method.]

☐ 2. Classes inheriting from Patient must provide their implementation of the setName method.

[✓ 3. The name field uses JavaBeans naming conventions for getters and setters.]

☐ 4. The description field uses JavaBeans naming conventions for getters and setters.

☐ 5. Classes inheriting from Patient can call, from their constructor, super() without arguments.

[✓ 6. The Patient class cannot be instantiated as a new Patient(“John Doe“, “John is a new patient“).]

2. Consider the following method:

```
public static <T> ArrayList<T> stackToList(Stack<T> stack, boolean reverseItems) {
    ArrayList<T> items = new ArrayList<T>(stack.size());
    while (stack.size() > 0) {
        T item = stack.pop();

        if (reverseItems) {
            items.add(0, item);
        } else {
            items.add(item);
        }
    }
    return items;
}

Select the computational complexity of the stackToList method if the reverseItems argument is true:
```

- ☐ 1. O(1)
- ☑ 2. O(n)
- ☐ 3. O(n*log(n))
- ☐ 4. O(n^2)

3. Consider the following code snippet:

```
static String readFirstLine(String path) {
    BufferedReader file = null;
    String buffer = null;
    try {
        file = new BufferedReader(new FileReader(path));
        buffer = file.readLine();
    } catch (IOException e) {
        System.out.println("Error reading from " + path + ". Message = " + e.getMessage());
    } finally {
        if (file != null) {
            try {
                file.close();
            } catch (IOException ex) {
                // ignore this exception
            }
        }
    }
    return buffer;
}

Select all the statements that are correct if the FileReader constructor throws an IOException exception.
```

- ☐ 1. The result of file.readLine() will be assigned to the variable buffer.
- ☑ 2. An error message will be printed to the console.
- ☑ 3. The file variable will be compared to null.
- ☑ 4. The file.close() function will be called.
- ☑ 5. The function will return null.

4. We want to use some of the Spring Boot Actuator features for monitoring their Spring microservices. We want to expose the following

```
Service health
Database status
Which beans exist in Spring context
Select all the ways to accomplish this.
```

- [✓ 1. Use the ping health indicator built in Actuator to check if the service is running.]
- [✓ 2. Use the datasource health indicator that is built in Actuator.]
- [✓ 3. Activate and use the /beans endpoint to get Spring beans.]
- 4. Use /dump to get a core dump of the Java process, which will contain Spring beans.
- 5. Activate and use /diagnostic to get the database status.
- 6. Use JVM indicator, built in Actuator, which can be used to check if JVM is alive, and with it, the services.

5. Which are the correct observations about the following controller?

```
@Controller
public class PasswordAdviceController {

    @Autowired
    private ComplexityCalculator complexityCalculator;

    @RequestMapping(value = "/calculateComplexity", method = RequestMethod.POST)
    @ResponseBody
    public String createAppointment(@RequestParam("password") String password) {
        return String.valueOf(complexityCalculator.calculate(password));
    }
}
```

- [✓ 1. A new instance of PasswordAdviceController will be created each time a HTTP request is served by it.]
- 2. PasswordAdviceController cannot be used to serve requests made over HTTPS.
- 3. The method argument "password" can be provided by a query string.
- [✓ 4. Spring will throw an exception if the parameter "password" is not present in the request.]

6. We are creating a new Java web project and wants to use Spring MVC, Spring Boot, and JPA for data access.After adding all dependencies, select the answer which will make Spring Boot automatically configure available services.

- [✓] 1. Add the @EnableAutoConfiguration annotation to an @SpringBootConfiguration class.
- [✓] 2. Add the @Autowired annotation to an @Config class.
- [ ] 3. Add the @Entity annotation to an @Config class.
- [✓] 4. Add the @Qualifier annotation to an @SpringBootConfiguration class.
- [ ] 5. Add the @Autowired annotation to an @SpringBootConfiguration class.
- [✓] 6. Add the @Config annotation to a class.

7. Consider the following class:

```
@Controller
@RequestMapping("/appointments")
public class AppointmentController {

    @Autowired
    private AppointmentService appointmentService;

    @RequestMapping(method = RequestMethod.GET)
    public String listAppointments() {
        return "appointmentsView";
    }

    @RequestMapping(value = "/create", method = RequestMethod.POST)
    @ResponseBody
    public Appointment createAppointment(@RequestBody Appointment appointment) {
        return appointmentService.save(appointment);
    }
}


Select the statements that are correct.
```

- [ ] 1. listAppointments() will be called when a user visits /appointments/get.
- [ ] 2. The content served by listAppointments() will be returned with the HTTP header "Content-Type: text/plain".
- [✓] 3. createAppointment() will be called only if a user submits an HTTP POST request to /appointments/create.
- [ ] 4. Calling createAppointment() will result in an error as the @RequestBody annotation can only be used with either a String or primitive types.
- [✓] 5. Values returned to the client by @ResponseBody methods are automatically mapped to a specific data format (e.g. XML, JSON) depending on Spring's configuration and the accept HTTP header.

8. A Spring Boot application needs tests for its data layer that uses JPA.Select all the annotations that should be added to the test class, that would allow you to inject EntityManager into it.

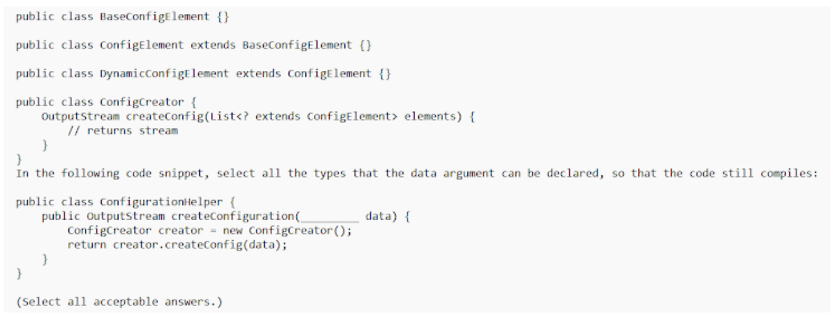
- ☐ 1. @SpringBootApplicationTest
- ☑ 2. @DataJpaTest
- ☑ 3. @WebMvcTest
- ☐ 4. @JpaTest
- ☑ 5. @SpringBootTest

9. Consider the following Java code:

```
public class MilesToKmConverter {
    public final double milesToKm(double miles) {
        return getMilesToKmFactor() * miles;
    }

    public double getMilesToKmFactor() {
        return 1.609;
    }
}

public class NauticalMilesToKmConverter extends MilesToKmConverter {
    @Override
    public double getMilesToKmFactor() {
        return 1.852;
    }
}

Select all the correct answers.
```

- ☑ new NauticalMilesToKmConverter().milesToKm(1) will return 1.852.
- ☑ ((NauticalMilesToKmConverter)new MilesToKmConverter()).milesToKm(1) will return 1.852.
- ☑ new MilesToKmConverter().milesToKm(1) will return 1.609.
- ☐ ((MilesToKmConverter)new NauticalMilesToKmConverter()).milesToKm(1) will return 1.609.

10. Consider the following model for programmatically creating configuration files:



```
public class BaseConfigElement {}

public class ConfigElement extends BaseConfigElement {}

public class DynamicConfigElement extends ConfigElement {}

public class ConfigCreator {
    OutputStream createConfig(List<? extends ConfigElement> elements) {
        // returns stream
    }
}
In the following code snippet, select all the types that the data argument can be declared, so that the code still compiles:

public class ConfigurationHelper {
    public OutputStream createConfiguration(________ data) {
        ConfigCreator creator = new ConfigCreator();
        return creator.createConfig(data);
    }
}

(Select all acceptable answers.)
```

[✓ 1. List<ConfigElement>]

[✓ 2. List<DynamicConfigElement>]

3. ArrayList<Object>

4. List<Object>

[✓ 5. ArrayList<ConfigElement>]

[✓ 6. ArrayList<DynamicConfigElement>]

7. List<BaseConfigElement>

8. ArrayList<BaseConfigElement>

Submit Clear form