serVme Java Developer Entrance Test

Troubleshooting

This section is divided up between Java, and SQL. Give as much information about your thought process as possible.

Question#1

Consider the following `Student` / `Staff` related classes

```
// Setters, getters and constructors omitted for brevity

public class Student {
  private String name;
  private String address;
  private String school;
  private String class;
  private double fee;
}

public class Staff {
  private String name;
  private String address;
  private String school;
  private String class;
  private double sallary;
}

public class StudentRepo {
  private List<Student> students = new ArrayList<>();
  public void addStudent(Student s){
    students.add(s);
  }
  public Student searchByName(String name) {
    // TODO
  }
}

public class StaffRepo {
  private List<Staff> staffs = new ArrayList<>();
  public void addStaff(Staff s){
    staffs.add(s);
  }
  public Student searchByName(String name) {
    // TODO
  }
}
```

1. Provide implementation for `searchByName()` methods
2. Looking at the above defined classes we can notice that there are a lot in common between them, is such a redundancy a bad or good practice? why?
3. Let us start by enhancing `Student` and `Staff` classes. They have common functionalities, we need to extract them to a separate `Person` class. Provide an implementation of such a class performing needed changes to `Student` and `Staff` classes.
4. Now let us move to`StudentRepo` and `StaffRepo` classes; they seem very similar, to remove this redundancy we want to replace them with a single `PersonRepo` class that can hold `Student` or `Staff` items (i.e. an instance of `PersonRepo` can hold either `Student` or `Staff` instances) in a type safe way. Provide an implementation for such a `PersonRepo`class.
5. Requirement was changed and now we want to enable `PersonRepo` class to hold both types `Student` and `Staff`. Provide a modified implementation to reflect this change.
6. Add a new `searchByName()` method that takes a second `kind` parameter (of type to be determined) returning `Student` type in case `kind` was student and returning `Staff` type in case `kind` was staff.

Question#2

Consider the following `Item` / `ItemRepo` implementation

```
// Setters, getters and constructors omitted for brevity

class Item {
  private int id;
  private String name;

}

class ItemRepo {
  private Set<Item> items = new HashSet<>();

  public void putItem(Item item){
    items.add(item);
  }

  public void removeItemById(int itemId){
    // TODO
  }

  public Item getItemById(int itemId){
    // TODO
  }
}
```

1. Add implementation for `removeItemById()`and `getItemById()` methods.
2. Consider the following code snippet:

```
ItemRepo repo = new ItemRepo();
repo.putItem(new Item(1001,"Joe"));
// Name 'Joe' was wrong name so we want to fix it as 'Jonathan'
repo.putItem(new Item(1001,"Jonathan"));
```

Did statement on line **4** fix the name inserted item at line **2** or we have now a duplicate item? If we have duplicate item then how can we fix this problem? we need to have any two items with same `id` be treated as same one.

3. Looking back at implementation of the different `ItemRepo` methods; Is `Set` best collection type to be used in terms of performance? Or we have a better alternative collection type?
4. Suppose that adding item to a repo is an expensive operation (see modified implementation of `putItem()` below) and we want to move it to a background thread. Modify the implementation `putItem()` to: 1) execute `actualPutItem()` on a background thread and 2) notify caller about the completion of the operation through a callback

```
public void putItem(Item item, SOME-TYPE callback){
    // TODO execute actualPutItem on a background thread and when
it finishes call 'callback'
    // TODO to notify the caller that the operation finished
}

private void actualPutItem(Item item){
    // Simulate an expensive operation
    Thread.sleep(2000);
    items.add(item);
}
```

Note: We need the implementation to be thread safe

5. Provide an alternative implementation `putItemAlt(Item item)` to use a `java.util.concurrent.CompletionStage` as return value instead of using a `callback` parameter to inform caller about the completion of the operation.
6. We want to enable interested parties to listen to put/remove item events through event listeners, a `Listener` is defined using the following interface

```
interface Listener {
    void itemPut(Item item);
    void itemRemoved(Item item);
}
```

Add a method `addListener(Listener listener)` to `ItemRepo` that adds a listener (multiple listeners can be added) to the repo. Do proper modifications so that listeners would be informed about all put/remove events.

7. We need add the ability to remove a listener when needed; to do so we need to modify `addListener()` method to return a `java.io.Closeable` instance; closing this `Closeable` would remove the listener.

MySQL

Question#1

Given the table structure and row data below, answer the follow up questions.

```
mysql> explain user_skill;

+-------------------------------+---------------------+-------+-----+
| Field                         | Type                | Null  | Key |
+-------------------------------+---------------------+-------+-----+
| user_skill_id                 | int(11)             |       | PRI |
| user_skill_last_modified      | timestamp(14)       | YES   |     |
| user_skill_date_created       | datetime            | YES   |     |
| user_id                       | int(11)             | YES   |     |
| skill_name                    | char(255)           | YES   |     |
| skill_level                   | char(255)           | YES   |     |
| skill_usage                   | char(255)           | YES   |     |
| skill_last_used               | char(255)           | YES   |     |
| user_skill_endorsed           | tinyint(1)          | YES   |     |
+-------------------------------+---------------------+-------+-----+
```

```
mysql> *************

+-----------------+-----------------+----------------------+
| user_firstname  | user_lastname   | skill_name           |
+-----------------+-----------------+----------------------+
| Kim             | Simpson         | PHP                  |
| Kim             | Simpson         | Perl                 |
| Kim             | Simpson         | Microsoft Word       |
| Kim             | Simpson         | Microsoft Access     |
| Kim             | Simpson         | Accounting/Billing   |
| Kim             | Simpson         | Java                 |
| Kim             | Simpson         | SQL                  |
| Kim             | Simpson         | CSS                  |
| Kim             | Simpson         | OO Programming       |
| Kim             | Simpson         | Microsoft Excel      |
+-----------------+-----------------+----------------------+
10 rows in set (0.00 sec)
```

1. Assuming that the data stored in `skill_name` in the `user_skill` table might be repeated for different users, what changes would you make to the database to normalize the `skill_name` and reduce repeat
storage? Show the structure of the new table(s).
2. Recreate the query that returned the 10 rows of data supplied. Speculate on tables that would be
needed that are not shown here.
3. Given the following query, how could it be optimized? List all assumptions:
`select c.* FROM companies AS c JOIN users AS u USING(companyid) JOIN jobs AS j USING(userid) JOIN useraccounts AS ua USING(userid) WHERE j.jobid = 123;`

Technical Sample Work

RESTful ToDo app

Applicant needs to create a RESTful API backend for a Todo application, using some JAX-RS implementation or Spring framework (preferably using quarkus framework)

The app should allow only authenticated users to manage their own todos, while being able to categorize them.

What are we expecting:

1. Login/Logout functionality to authenticate user
2. User registration, with the below fields;
    a. First Name
    b. Last Name
    c. Email
    d. Mobile number
    e. Gender
    f. Birthday
3. Todo item details
    a. Name
    b. Description
    c. Date time
    d. Status
    e. Category
4. After a successful login, we should be able to
    a. Add/Remove Categories
    b. Add/modify todos including changing their status (Initial, Started, Completed, Snoozed, Overdue)
    c. List todos per user with ability to filter per day and/or month, by categories and status

The following features are optional:

- Forgot password feature
- Email notifications (on signup and password reset)
- Pagination functionality

Criterias

- Code quality and management
- Methods of enforcing code quality
- Design Patterns
- Performance
- Security
- Testing
- Best practices
- State management

Deliverables

Applicant needs to send a GitHub repository link, a working master branch, including a docker-compose, and an accompanied README.md which details build instructions and any other necessary Information.