Assignment #1 - Service locator

Implement a simple service locator library.

Create class io.pliant.internship2022.service.locator.ServiceLocatorImpl, which implements the provided io.pliant.internship2022.service.locator.ServiceLocator interface. This class should implement basic service registration and discovery functionality. Users must be able to register individual objects or classes to the locator and be able to get the registered objects, or objects, instantiated from the classes. Class instances are created lazily upon lookup. Once an instance is created it should be returned in subsequent lookups. Details on behavior can be found in Javadoc for each method of the interface.

Requirements:

- Create class io.pliant.internship2022.service.locator.ServiceLocatorImpl with public no arg constructor and implement the ServiceLocator interface.
- Create checked exception class io.pliant.internship2022.service.locator.ServiceLocatorException and a hierarchy of subclasses - one for each possible error condition in your code (e.g. registration is not found, more than one registration found, error instantiating class, etc.)
- On error, a specific ServiceLocatorException subclass instance should be thrown.
- Demonstrate the functionality of your code, showcasing all of the interface functions. We encourage you to use a test framework like JUnit.
- Document your code with Javadoc.

Technologies:

You can use helper and utility libraries like Lombok and Apache Commons for trivial tasks such as string manipulation (e.g. toCamelCase()), but the core business logic must be your own code.

Criteria:

- Correct implementation of ServiceLocator
- Model classes for internal structures
- Showcase the implementation
- Meaningful ServiceLocatorException hierarchy
- Code formatting
- Naming conventions

Assignment #2 - Minimum differences

Pliant teams PNT and PWE are following the SCRUM methodology. Their sprints are in a time interval of 2 weeks and start and end on the same dates for both teams. Each sprint each of the teams completes a different number of story points.

We define stretch of sprints T as N consecutive sprints from a sprint array starting at position *s*. Element difference D*i*(Ta, Tb, i) for element *i* is the absolute value of the difference between matching values of two stretches Ta(i) and Tb(i).

Given two arrays with integers - story points for teams PNT and PWE - and positive number N (N >= 2) find stretch Tpnt of N consecutive sprints from PNT and stretch Tpwe of N consecutive sprints from PWE, so that the sum of element differences between matching elements from the two stretches is minimal.

Input is read from the standard input (System.in) - a string formatted as a JSON object with the following structure (the string contains no new lines, here it is broken down for readability):

```
{
      "n": <number>,
      "pnt": <array of numbers>,
      "pwe": <array of numbers>
}
```

Output string representation of JSON object on the standard output (System.out) in format (output string must not contain new lines, here it is broken down for readability):

```
{
      "d.min": <number>,
      "s.pnt": <number>,
      "s.pwe": <number>
}
```

Where "d.min" is the minimum sum of element differences, "s.pnt" is the starting index of the PNT stretch Tpnt, "s.pwe" is the starting index of PWE stretch Tpwe. All indices are zero based.

Requirements:

- If either N < 2, or length of PNT or PWE array is less than N, return string representation of JSON object with single field "error" and value a string with meaningful error description of your choosing.
- If there is more than one solution, output one with the lowest value of the Tpnt starting index..

Examples:

Example 1:
Input: *{ “n”: 10, “pnt”: [11, 14, 10, 12], “pwe”: [10, 13]}*
Output: *{ “error”: “length of pnt array is less than 10”}*

Example 2:
Input: *{ “n”: 3, “pnt”: [11, 14, 10, 12], “pwe”: [12, 9, 8, 11, 9]}*
Output: *{ “d.min”: 7, “s.pnt”: 0, “s.pwe”: 2}*

Additional test inputs will be provided in this Google Drive folder on Saturday (21.05.2022).

Technologies:

You are encouraged to use Jackson or any JSON <=> POJO mapping library.

Criteria:

- Model classes for input and output.
- Usage of JSON <=> POJO mapping library.
- Code formatting.
- Naming conventions.

Extra:

Starting from Saturday (21.05.2022) test inputs will be published on the HiveMQ public MQTT broker. Test inputs will be published every 5 minutes to the topic “pliant.io/internship/2020/backend/a2/input” and will have the same format, described above, with addition of two fields:

- “testId”: <string> - identification of test data
- “replyTopic”: <string> - topic, to which results will be published.

Write a program that subscribes to the aforementioned topic, upon receiving input data, solves the assignment and publishes the result in the topic, specified by “replyTopic”. Published result is a string - JSON representation of an object with the same format, as described above output of the application, with addition of the following fields:

- “testId”: <string> - the same value as in the input
- “me”: <string> - your names
- “email”: <string> - your email address (the one you used in applying for this internship)

Upon receiving your output data, our automated system will check your solution and publish a string representation of JSON object to the topic “pliant.io/internship/2020/backend/a2/result/<email>”, where <email> is your email (without the angular brackets). This object will have the following fields:

- “testId”: <string> - identification of test data, the same as in the input and the result
- “solution”: <string> - with value “correct” for correct solution, “incorrect” for wrong solution or “invalid” for test inputs, published more than 5 minutes ago.

If your result is malformed (not a JSON, missing field or field with wrong type) - no message will be published.

There is no restriction on using MQTT related libraries and technologies.

Assignment #3 - FiboНачи

The famous Fibonacci sequence is defined by the following rules:

```
Fib(1) = 1, Fib(2) = 1, Fib(n) = Fib(n-1) + Fib(n-2), n > 0
```

Write a program that generates the following meme:

Your program should read an image file to be used as a template (example file “fibonacci.png” is provided as an asset), write the string “ФибоНачи” on the top-center of the image, then extend the image downwards with black rectangle to accommodate (M - N + 1) strings. Each string will represent i-th Fibonacci number “Като зема <Fib(i)> дърво …” or “Като зема <Fib(i)> дървета …” depending on the value of Fib(i). The strings are generated for the interval i ∈ [N, M]. In the provided example N = 1, M = 7. After the image is generated, write it to file (preferably in PNG format).

For simplicity of the implementation you can hardcode the N and M parameters in the main method of the application.

Requirements:

- Implement image manipulation in class, different from the main class
- The (M -N + 1) strings must fit nicely into the created black rectangle and be left justified.

Technologies:

You are free to use any graphics manipulation library.

General requirements

Use Java SE, whatever version you are comfortable with. Your projects should include a readme file with short description and instructions on how to build and run them. Ideally the projects and their dependencies should be managed by Maven or Gradle.

You must upload your work on the assignments to a public Git repository (Github, Bitbucket, Gitlab) and send an email to jobs@pliant.io with the link to the repository and a subject “Pliant Backend Internship - Assignment”.

Please ensure that the link to your work is publicly accessible. Any assignments that can't be opened will not be reviewed, and we will not send you back an e-mail asking for access.

We encourage you to send back whatever result you achieve, as we will take into account and grade all submissions, even if they’re not completed at 100%.

Good luck!