Practice Final Exam for preparation #2

Programming Languages Java exam, practical part

Conditions

Do the following **right now**: make sure that no communication device is available to you.

- **Put away** phones, headphones, tablets etc.
- **Close** all chat programs, mail clients etc.
- **Keep these things off/away** during the exam.
- If you're found cheating (e.g. giving or receiving help) during or after the exam, you have failed the course.

During and after the exam.

- You are forbidden from sharing any part of your exam solution until the day after the exam.
- You are allowed to **search the Java API documentation here (https://docs.oracle.com/en/java/javase/19/docs/api)** .
  - Otherwise, you may not use any other sources (books, notes, sample codes, the Internet etc.).
- You are only allowed to use a "simple" text editor (that doesn't have advanced features like code completion or automatic compilation), so no IDEs.
- About the code.
  - Whenever a name is specifically given, use that name exactly.
  - Follow good practices.

Submitting.

- Solve the exercises in order.
- When the time is nearly up (with about 10 minutes left to go), zip the project that you created and upload it into Canvas.

Test cases

Click here to download the required `.jar` file. (Its name has been shortened.)

Compile and run the test cases like this:

```
javac -cp ".;junit5all.jar" <insert test case file path here>
java -jar junit5all.jar -cp . -c <insert fully qualified name of tester class here>
```

On Linux boxes, use `:` instead of `;`.

If the terminal doesn' support colours and the output is a garbled mess, add the `--disable-ansi-colors` option to the second command.

You have to write the test code yourself. You'll have to insert these lines near the beginning of the file.

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.*;
import org.junit.jupiter.params.*;
import org.junit.jupiter.params.provider.*;
```

Exercise

In this exercise, we are representing celestial bodies.

Celestial bodies

Create two enumeration types: `starexplorer.celestialbodies.CelestialBodyType` with the values `MS_STAR` (Main Sequence Star), `GAS_GIANT`, and `TERRESTRIAL_PLANET`, and `starexplorer.celestialbodies.WorldType` with the values `CONTINENTAL`, `OCEAN`, `LANDMASS`, `MOLTEN`, `FROZEN`, and `BARREN`.

Create the class `starexplorer.celestialbodies.CelestialBody`.

- The class has the following, public fields.
  - `name`: a text
  - `mass`: a real number
  - `massExponent`: integer, multiply `mass` by `10` to this power (10`massExponent`) to get the actual mass of the celestial body
  - `bodyType`: a `CelestialBodyType`
- Let the class have a constructor that takes initial values for these four fields.
  - Perform a check: the celestial body is invalid if its `name` is given as an empty text or is `null`. It is also invalid if `mass` is less than or equal to zero.
  - If an invalid celestial body is found, throw an `IllegalArgumentException`.
- Let the textual representation of a `CelestialBody` look like this: `Earth(5.972e24,TERRESTRIAL_PLANET)`.

In `starexplorer.StarTests`, create a JUnit 5 tester class.

- In it, create method `testEarth` that tests that Earth's textual representation looks as expected.

Let `starexplorer.celestialbodies.Star` be a child class of `CelestialBody`.

- It has a public field `surfaceTemperature`, an integer.
- Let the class have a constructor that takes initial values for the five fields.
  - The first four are as described in the parent class.
    - Check that `bodyType` is `MS_STAR`. If it isn't, throw an `IllegalArgumentException`.
  - The fifth one is the value for `surfaceTemperature`.

Create the class `starexplorer.celestialbodies.Planet`, a child of `CelestialBody`.

- Let it have a public `worldType` field of type `WorldType`, and three more fields (`oxygen`, `nitrogen`, `otherElements`, all integers).
- Let the constructor take everything needed for the initialisation of the base class, then all of the above as parameters.
  - This gives 8 arguments for the constructor altogether.
- Let its textual representation look like this: `Planet Earth(5.972e24,TERRESTRIAL_PLANET) of LANDMASS with (78 oxygen, 21 nitrogen, 1 other)`

In `StarTests`, method `testEarthAsPlanet` that tests that Earth's textual representation (as a planet) looks as expected.

Exploring the star system

Create the class `starexplorer.observation.StarSystem` with two public fields: `star` (a `Star`) and `planets`, a list of `Planet`s.

Create the class `starexplorer.observation.StarExplorer` that represents a spaceship that searches for habitable planets.

- Let it have a private field `starSystem` (`StarSystem`) that represents the star system currently under investigation.
- Its constructor takes a filename. It opens the file and processes it.
  - You may assume that the file is OK: it exists and its contents are formatted properly.
  - Example: the file `solarsystem.txt` may look like this.

```
Sun 1.989 30 MS_STAR 5778 5
Earth 5.972 24 TERRESTRIAL_PLANET CONTINENTAL 21 78 1
Venus 4.867 24 TERRESTRIAL_PLANET BARREN 0 4 96
JUPITER 1.898 27 GAS_GIANT FROZEN 0 0 100
```

  - The first line contains data about the `Star` of the system, the rest of the line describe the `Planet`s.
    - Their structure correspond to what the respective classes' constructors take as arguments.
  - Initialise the `starSystem` variable using the loaded system.
- Create the static method `isHabitable` that takes a planet and returns `true` exactly if all of the following conditions are met.
  - Its `bodyType` is `TERRESTRIAL_PLANET`
  - Its `worldType` is `CONTINENTAL`, `OCEAN`, or `LANDMASS`
  - Its `atmosphere.nitrogen` is between 70 and 80
  - Its `atmosphere.oxygen` is between 20 and 25
  - Its `atmosphere.otherElements` value is not greater than 5

- Create the instance level method `isHabitable` that takes a planet name and returns `true` exactly if all of the following conditions are met.
  - A planet by this name exists in the star system.
  - The other `isHabitable` method returns `true` for this planet.

In `StarTests`, method `testStarExplorer` that tests that after loading the file `solarsystem.txt` with the above contents, Earth is habitable, but all the other planets are not. Also test that the planet `XYZ` is not habitable, as it does not exist in the system.