2 Homework 2: Single-Class Design

Due: Sunday 10/01 11:59pm

Update (9/29 5pm): Gradescope Autograder section added.

Update (10/1 2pm): Added clarifications about autograder correctness tests (in Gradescope Autograder) and **`TwoLaneQueue`** *constructor (in* **`TwoLaneQueue`***). (Updates are marked with italicized comments.)*

In this homework, you will design (that is, plan, implement, and test three classes according to the rules for Single-Class Design (as discussed in Lectures 02 through 04).

Treat the three classes as three unrelated Single-Class Design exercises. That is, do not treat them as a single Multi-Class Design problem. They are in the same package, in the same project, purely for convenience.

This homework does not fully specify the signature of every method, the types you should use in the class representations, and so on. You are expected to make reasonable inferences about what to implement and how to implement it.

The homework will be graded based on the principles of class design, the rules for Class Design Documentation, the correctness of your implementation, the quality and comprehensiveness of your testing, and the reasonableness of your inferences about issues that the homework does not explicitly specify.

Additional rules:

- Each class must be in the **`cs410`** package.
- Classes and methods must be named exactly as written.
- All methods should be instance (non-**`static`**) methods except those explicitly described as static in the homework specification.
- Your work must be on a branch named `hw2`. See Making the `hw2` branch.
- On the `hw2` branch, there must be in a directory named `Homework2`, and the contents of that directory must follow the Maven Directory Layout. It must be possible to build and test your project using Maven (`mvn compile` and `mvn test`, respectively). IntelliJ users should see IntelliJ and Maven below.
- If your work is not in the correct branch, or not in the correct directory, or cannot be built with Maven, then it will not be graded and you will not receive credit for the assignment.

2.1 Duration

Design a class named **`Duration`** representing a duration of time, with a precision of seconds. That is, do not support fractional seconds. Duration objects should be immutable.

Define two static factory methods (both named **`of`**). One should take a single argument representing the total number of seconds in the duration. The other should take three arguments containing the numbers of hours, minutes, and seconds in the duration.

Define an **`add`** method that adds two **`Duration`** objects together.

Define a **`seconds`** method that returns the Duration’s total number of seconds.

Override **`toString`** to display the duration in `H:MM:SS` format. For example, the nominal duraction of a CS410 lecture is `"1:15:00"`. One million seconds corresponds to `"277:46:40"`. (Hint: See the **`String.format`** method.)

2.2 Rectangle

Design a class named **`Rectangle`** for representing a rectangle on a 2D grid with integer coordinates.

Define a static factory method named **`of`** that takes four integer arguments: the “x” and “y” coordinates of one corner of the rectangle, and the “x” and “y” coordinates of the opposite corner.

Define a method to calculate the **`area`** of the rectangle.

Define a method that determines whether a rectangle **`contains`** a point, represented by integer “x” and “y” arguments.

Define a method that determines whether a rectangle **`overlaps`** with another rectangle.

2.3 TwoLaneQueue

Design a class named **`TwoLaneQueue`**. It is like a standard mutable queue, except that it has two “lanes”: a slow lane and a fast lane. (Think of airport boarding lines with separate lanes for General vs Priority boarding, for example.)

Define a public constructor that takes zero arguments. *(Clarified 10/1 2pm for compatibility with the autograder tests. Since this change was posted late, reasonable alternatives will not be penalized during manual grading, but automated feedback through the autograder is only possible if the public constructor is defined.)*

The items contained by the lanes are simple **`String`** objects.

The **`enqueueFast`** operation adds an item to the end of the fast lane.

The **`enqueueSlow`** operation adds an item to the end of the slow lane.

The **`dequeue`** operation removes and returns an item from the front of one of the lanes. Specifically:

- If there is an item waiting in the fast lane, it is dequeued and returned.
- If there is an item waiting in the slow lane, it is dequeued and returned.

- If both lanes are empty, a `java.util.NoSuchElementException` is thrown.
- Except, as a kind of limited fairness, if the fast lane has produced three or more items in a row, and if the slow lane is not empty, then the slow lane gets to take a turn instead.

You must not place a bound on the number of items in the fast and slow lanes.

You may use any class present in `java.util.*` (as of JDK 11) in your implementation.

2.4 Making the `hw2` branch

You can use the following command to make a new branch called `hw2` that does not share any history with any of your other branches:

```
git switch --orphan hw2
```

2.5 IntelliJ and Maven

IntelliJ makes it fairly easy to satisfy the homework's requirements for directory structure and Maven-compatibility.

- Before you create the new project, make sure your git working copy is in the `hw2` branch.
- Create a new IntelliJ project named `Homework2`. The Location should be your git working copy's directory. Select **Maven** for the Build System option. IntelliJ will automatically create a `pom.xml` file, which contains information about your project for Maven.
- Add the following text to the bottom of your `pom.xml` file, just before the closing `</project>` tag:

  ```
  <build>
      <plugins>
          <plugin>
              <artifactId>maven-surefire-plugin</artifactId>
              <version>3.1.2</version>
          </plugin>
      </plugins>
  </build>
  ```

  IntelliJ should automatically download the necessary plugins.
- Create your testing classes by opening the source file of the class to test (eg, `Duration.java`), selecting the name of the class, and hitting Alt-Enter to open a special context menu, then selecting "Create Test". That will automatically create the `src/test/java` directory and mark it as a test source root. It will also name the testing file according to the convention expected by the Maven Surefire plugin (eg, `DurationTest`).

2.6 Gradescope Autograder

Section added 9/29 5pm.

There is now a Homework 2 assignment available on Gradescope with a limited autograder.

The autograder expects a single file named `hw2.json`, which should contain a JSON object with a single key named `"gitlab_user"`, whose value must be your Gitlab user name. It must be the same Gitlab user name you used for Homework 1. For example, my `hw2.json` file would have the following contents:

```
{ "gitlab_user": "rmculpepper" }
```

The autograder *currently* checks that your repo on Gitlab is accessible, that repo has the correct branch (`hw2`), that branch contains the correct directory (`Homework2`), and the project contains the expected directory structure and files, and the project can be built using Maven (`mvn compile`).

You can resubmit exactly the same file to make the autograder pull the most recent version of your code from Gitlab.

~~The autograder does not *currently* perform any tests about the correctness of your code. Correctness tests may be added before the homework deadline.~~ The autograder now runs correctness tests. *(Updated 10/1 2pm, tests added by 9am.)*

Passing the autograder checks means that your code can be graded, but the autograder does not assign the final grade for the homework. The grade depends on many other factors; grading criteria are listed at beginning of the homework assignment.