Homework 06

Authors: Ishuma, Brittney, Jacob, Dhruv

Activity 1: JavaFX

Purpose

Congratulations! Your coding skills landed you a position as the Travel Advertiser for the city of OOP-Topia. For some unknown reason this city has seen a lack in tourism lately, and the Mayor needs your expertise. You decide to think outside of the box and use your newfound knowledge of JavaFX to create the ultimate travel brochure. To complete this assignment, you will apply your knowledge of **JavaFX shapes, text, and images** to implement a GUI Application that will represent a travel brochure.

Solution Description

For this activity, you will be creating a JavaFX GUI Application to reflect a travel brochure. You will also have to adhere to proper Checkstyle including Javadocs.

`Brochure.java`

This class contains a JavaFX GUI Application that displays a travel brochure for OOP-Topia. Remember to consult the Java and JavaFX API to find information about classes and methods that are available to you to use. In this class you will do the following:

- Create a window with the title of "TravelBrochure".
- At the top of the window there should be a **blue** `Rectangle` with "`WELCOME TO OOP-TOPIA!`" in **white** text on top of it.
- In the center (or close to the center) of the window there should be an `Image` of your choosing with a sentence below it detailing how the `Image` relates to what a tourist can do and/or find in OOP-Topia.
    - Please **submit any image files you use to Gradescope** and assume the image files are located in the current working directory when you refer to them in your code.
- To the left and right of the `Image` there should be two of the *same* `Shape` in *different* `Colors`.
    - These `Shape` objects may NOT be `Rectangles` and neither one may be the `Color` blue.

Activity 2: Recursion

Purpose

You decide to go undercover as a tourist in OOP-Topia to truly fulfill your advertising duties. You intend to use your **recursion** skills to find out how many activities in the city you can accomplish and gain some intel about what goes on inside OOP-Topia.

Solution Description

For this activity, you will be creating a recursive method within a Java file called `UndercoverTourist.java`. Based on a travel budget and catalog of activities with their prices, you need to recursively determine how many activities you can accomplish in the city. You are provided with a defined class called `TravelActivity.java` that represents an activity you can complete in OOP-Topia and contains information about its cost. You will also have to adhere to proper Checkstyle including Javadocs.

`UndercoverTourist.java`

This class will contain one **static** method that will handle the logic of recursively determining how many activities you can accomplish with a given budget.

Methods:

- `int calculateNumActivities(ArrayList, double, int)`
  - This method should be `static` and take in three parameters in the following order:
    - An `ArrayList` of type `TravelActivity` representing a catalog of activities that can be purchased in the city.
      - ◊ Each `TravelActivity` has a `name` and `price`.
      - ◊ You may assume the passed-in `ArrayList` is not null and that each `TravelActivity` in `ArrayList` is unique.
    - A `double` representing your current budget.
    - An `int` value representing the current number of activities already purchased.
  - You are to recursively determine the largest number of different activities you could purchase with your current budget.
    - Each time you recurse, you should purchase the *cheapest* available `TravelActivity` and update your budget and number of purchased activities accordingly. If there are multiple TravelActivities that have the same price, choose the TravelActivity that appears first in the catalog to purchase.
    - You SHOULD remove an activity from the catalog once you purchase it to ensure you do not repeat an activity.
  - This method returns an `int` value representing the largest number of activities you could purchase with your current budget.
  - **Note:** Searching for the cheapest activity to complete in each recursive call will result in completing the maximum possible number of activities. This is an example of a Greedy Algorithm: https://www.geeksforgeeks.org/greedy-algorithms/
  - **Note:** Each `TravelActivity` has its own `name` and `price`. You can view (but do NOT edit) the `TravelActivity.java` file.
  - **Note:** You may want to use a helper method to do your searching and keep your code organized.

Clarifications and Example Output

Please refer to the pinned Piazza thread for HW06 Clarifications for any necessary clarifications about the requirements outlined in this pdf. We may also post some example output that you can use to check your code against on the same Piazza post.

Allowed Imports

- anything from the javafx library for Activity 1
- `java.util.ArrayList` for Activity 2

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

- `SceneBuilder`

Checkstyle and Javadocs

You must run Checkstyle on your submission. The Checkstyle cap for this assignment is **15 points**. If you don't have Checkstyle yet, download it from Canvas -> Modules -> Checkstyle Resources (all sections) Place it in the same folder as the files you want Checkstyled. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository or website. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- Brochure.java
- Any image files needed to run Brochure.java
- UndercoverTourist.java

Please only submit the individual files—do not put them into a package or you will risk getting a zero on the assignment. Make sure you see the message stating "HW06 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit .class files
- Test your code in addition to the basic checks on Gradescope
- Run Checkstyle on your code to avoid losing points
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Feature Restrictions" to avoid losing points
- Check on Piazza for a note containing all official clarifications