The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 270– Developing Object-Oriented Systems

Assignment 4

Date Due: October 18, 2024, 6:00pm

Total Marks: 77

General Instructions

- Assignments must be submitted using Canvas.
- Programs must be written in Java.
- VERY IMPORTANT: Canvas is very fragile when it comes to submitting multiple files. We insist that you package all of the files for all questions for the entire assignment into a ZIP archive file. This can be done with a feature built into the Windows explorer (Windows), or with the zip terminal command (LINUX and Mac). We cannot accept any other archive formats. This means no tar, no gzip, no 7zip, no RAR. Non-zip archives will not be graded. We will not grade assignments if these submission instructions are not followed.

Question 1 (10 points):

The objective of this question is to demonstrate correct use of git and adherance to an iterative, text-driven design process.

Your Tasks

At this point, we are assuming that you have your git repository and are familiar with the process.

You should start by creating a tag to indicate when you started the assignment. The following command will do this:

```
git tag A4
```

This will make it easy for instructors and markers to navigate your repository for purposes of marking and providing assistance, as well as simplify generating your gitlog at the end.

Generating GitLog

When generating your gitlog, you should only include the commits that are relevant to this particular assignment. There are a number of ways you could do this (e.g., create a separate branch for each assignment and generate a git log for the appropriate branch). If you tagged the repository before starting the assignment (see above section), you can print all commits since making the tag with the following command:

```
git log A4..HEAD > gitlog.txt
```

This command will generate a git log containing all commits after the A4 tag, and the output is redirected to a text file named `gitlog.txt`.

Expectations for Iterative Development

Your git log should demonstrate evidence of iterative development (i.e., document your work step-by-step). To get full marks here, your git log should demonstrate the process as discussed in Lecture 07 and Assignment 3. As assignments progress, we will be less explicit about the exact process you should be following, as you gain experience developing object oriented systems.

For **Question 2**, you should follow the process:

1. Create module named `Assignment4` (this name needs to be exactly this for some of the starter code to work correctly)
2. Create a class named `Stack` that implements the provided interface `StackOperations.java`
3. Declare the attributes/class variables and create method stubs (the IDE can automatically create method stubs for you by implementing the provided interface. This is a great reason why we would want to use interfaces)
4. Write regression tests for the methods
5. Implement the methods until all tests pass

The UML diagram in **Question 3** should be created **Before** working on Question 4. You will likely have trouble understanding the provided starter code, creating a UML diagram is an excellent way to become familiar with this code.

For **Question 4**, you should write regression tests for the method stubs you are being asked to implement (no need to test the provided methods) **before** you implement the methods.

Evaluation

10 marks : git log submitted illustrates student followed a reasonable iterative process when completing the assignment.

Files to Submit

A text file called `gitlog.txt`

Question 2 (17 points):

The objective of this question is to practice using Java's Collection framework and Generic typing to implement a familiar data structure.

Your Task

In previous courses you were likely introduced to Stacks and Queues. In the lecture, we implemented a generic Queue abstract data type (ADT) by creating a wrapper class around Java's `ArrayList` data structure. In this question, you will be creating a Stack class which will be a wrapper around a `LinkedList`.

In general, your `Stack` class should have the following features:

- It should use Generics to allow the Stack to store any datatype, defined when an instance of the Stack is created
- It should support the following methods:

  Stack : the constructor to perform any required initialization
  push : add an item to the top of the stack
  pop : remove an item from the top of the stack. This **should not return anything**.
  peek : return the item currently at the top of the stack
  size : return the number of items currently on the stack
  isEmpty : return true if the stack is empty, false otherwise
  toString : return a string representation visualizing the stack vertically

- All methods and class attributes/variables should have appropriate javadoc
- All preconditions should be checked and handled by throwing appropriate exceptions
- Include a regression test in the Stack class' `main` method that thoroughly tests each method (including testing that exceptions are correctly thrown)
- An interface file is provided called `StackOperations.java`. Your Stack class should implement this interface, to ensure your class will work with the marker test scripts
- Ensure your Stack class is working correctly, the markers will be using your Stack class to perform various tasks that require a Stack to ensure it works correctly.

Evaluation

2 marks : appropriate use of Java's `LinkedList` container class to store the stack contents
2 marks : appropriate use of exceptions to check invalid preconditions
3 marks : 0.5 marks each for correct implementation of required methods as listed above
3 marks : 0.5 marks each for correct javadoc for required methods as listed above
3 marks : 0.5 marks each for appropriate regression tests of required methods as listed above
4 marks : Class compiles and works correctly, implements StackOperations interface, works correctly on marker test scripts

Provided Files

StackOperations.java : interface that your Stack should implement

Files to Submit

Your submission should include a file named `Stack.java` containing the implementation of the Stack ADT, appropriate javadoc-style comments, and adequate regression tests for all implemented methods.

Please ensure you have included identifying information at the top of every .java file in the form of a comment (Name, nsid, student number, etc).

Ensure the files you submit are .java files from the src/ directory in IntelliJ. Any submitted .class files cannot be opened by the marker and therefore will result in grades of 0 where relevant in the rubric.

Question 3 (10 points):

The objective of this question is to practice creating UML diagrams as a way of understanding an existing system.

Your Task

In Question 4 you will be implementing some methods in some partially implemented classes. Before you do any coding, you need to have a good understanding of how the existing classes interact with each other. To do this, you will read through the existing code and create a UML diagram of the classes (including the `Achievement` class that you will be creating.

Evaluation

10 marks : UML diagram accurately reflects the Achievement system

Provided Files

AchievementManager.java : a skeleton for the AchievementManager class
AchievementDemo.java : a starting main program
AchievementFileReader.java : a class to handle reading and writing to a text file. **This file should be unmodified.** :

Files to Submit

Your submission should include a file named `achievement-UML.pdf` containing your complete UML diagram.

Please ensure you have included identifying information at the top of every .java file in the form of a comment (Name, nsid, student number, etc).

Question 4 (40 points):

The objective of this question is to practice reading and writing to a text file in Java, practice working with Java's Map collections, and practice creating and testing Java classes.

Your Task

A common component of modern video games are Achievements (or Trophies in PlayStation world). When the player achieves a specific feat in a game (e.g., use an ability a certain number of times, reach a specific part in the game's story, make a particular decision, etc) they will `unlock` a Trophy/Achievement which is recorded in their gaming profile. In this question, you will be designing and implementing a basic system that reads a list of achievements from a text file, allows a user to add achievements, unlock achievements, and save the modified achievement list to the original text file.

Note: the code to read and write to a text file will be provided to you, but specific steps need to be followed to make it work in IntelliJ. This will be covered in the IO tutorial (see the Tutorial table on Canvas for specific date).

Specifically, you will need to do the following:

Achievement Class

Create a class called `Achievement` which will model a single achievement in a game. It should have the following attributes:

achievementName : a String that represents the name of this Achievement
achievementDescription : a String that represents a written description of this Achievement
isUnlocked : a boolean that keeps track of whether or not this achievement has been unlocked yet

In addition, the Achievement class should have the following methods:

constructor : take in 2 strings as parameters (name, and description) and assign them to the appropriate class variables. The achievement should also default to a locked state.
getAchievementName : a getter for the achievement name class variable
getAchievementDescription : a getter for the achievement description class variable
checkUnlocked returns a boolean indicating whether or not the achievement is unlocked
unlockAchievement : change the achievement's state to unlocked
toString : override the toString method and return a string that indicates the name and description of the achievement, as well as its current locked state
main : include a full regression test in the main method, testing each method individually. Also, you should test any exceptions you decided to throw to ensure they are being thrown correctly.

AchievementManager Class

The AchievementManager class should be a wrapper around a Map collection. It should store Achievements that are keyed by an integer id, and support the ability to add an achievement, get an achievement, get a list of all achievements that are currently locked, get a list of all achievements that are currently unlocked, a toString method, and the main method with a full regression test. A skeleton of this class has been provided, with `// todo:` comments indicating where implementation is needed.

AchievementDemo Class

This will be the main program that handles reading a text file, saving to a text file, and interacting with the AchievementManager. A basic version of this class is provided, but it contains some `// todo:` comments that indicate what you need to add and where to get the entire system running and working properly. You should not modify this class except where explicitely indicated.

Evaluation

5 marks : everything compiles and runs with no major errors
9 marks : Achievement class implemented correctly
5 marks : Achievement class contains thorough regression tests
10 marks : Required methods in AchievementManager class correct
5 marks : AchievementManager class contains thorough regression tests
6 marks : Modifications to AchievementDemo class correct, nothing else modified except where specified

Provided Files

AchievementManager.java : a skeleton for the AchievementManager class
AchievementDemo.java : a starting main program
AchievementFileReader.java : a class to handle reading and writing to a text file. **This file should be unmodified.** :

Files to Submit

Your submission should include the following Java files:

Achievement.java
AchievementManager.java
AchievementDemo.java

Please ensure you have included identifying information at the top of every .java file in the form of a comment (Name, nsid, student number, etc).

Ensure the files you submit are .java files from the src/ directory in IntelliJ. Any submitted .class files cannot be opened by the marker and therefore will result in grades of 0 where relevant in the rubric.

Assignment Summary

Files Provided

StackOperations.java
AchievementManager.java
AchievementDemo.java
AchievementFileReader.java

What to Hand In

You must submit the following files, compressed in a .zip file named A4-abc123 (where abc123 is your NSID):

gitlog.txt
Stack.java
achievement-UML.pdf
Achievement.java
AchievementManager.java
AchievementDemo.java

Grading Rubric

Evaluation information can be found for each question above. See Canvas for the complete grading rubric.