Lab Exercise 7
CS 2334

October 8, 2015

Introduction

In this lab, you will experiment with the use of generics. Generics allow you to abstract over types. More specifically, they allow types (classes and interfaces) to be parameters when defining classes, interfaces and methods. You will create a card game that makes use of a generic Deck class to create decks of cards of two different types: one based on the color of the card and the other based on the face value of the card. Although we will have two very distinct types of Decks, both will need a standard set of operations (including shuffling and drawing). By using Java generics to implement a generic Deck class, we only need to implement these methods once and the implementation will work for many different types of card decks.

As part of the Deck implementation, we will also make use of the Stack class from the Java Collections Framework to store stacks of used and unused cards.

Learning Objectives

By the end of this laboratory exercise, you should be able to:

1. Create classes using generics in Java
2. Use generic classes to solve larger problems
3. Use the Stack class to store and retrieve objects

Proper Academic Conduct

This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Import the existing lab7 implementation into your eclipse workspace.

   (a) Download the lab7 implementation: `http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab7/lab7-initial.zip`

   (b) In Eclipse, select *File/Import*

   (c) Select *General/Existing projects into workspace*. Click *Next*

   (d) Select *Select archive file*. Browse to the lab7-initial.zip file. Click *Finish*

The Card Game

The game has two decks of cards: a poker deck and color deck. To play the game:

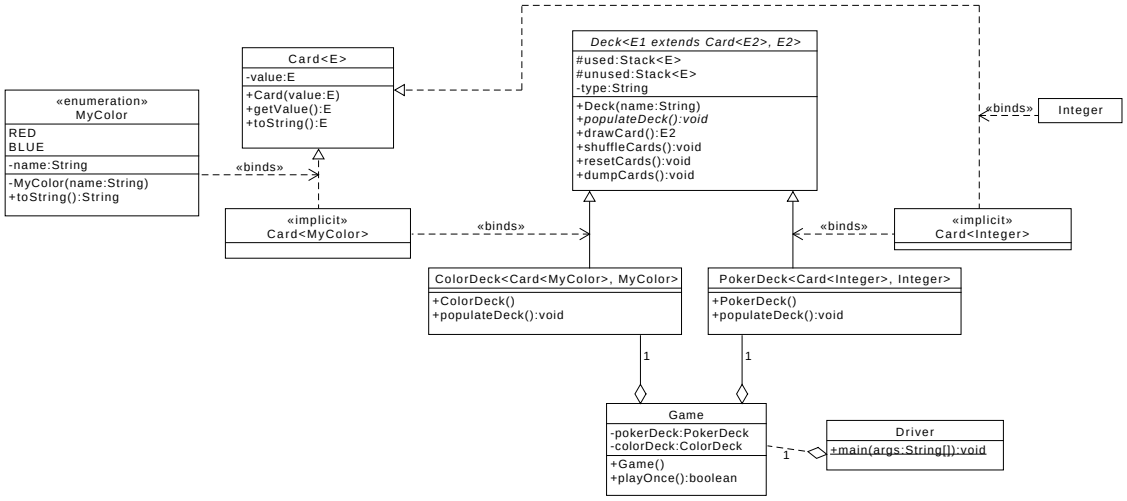
- The player draws one card from each deck.
- If the color card is RED, the player wins if the poker card has an even value.
- If the color card is BLUE, the player wins if the poker card is divisible by three.

The full implementation of our program will first create one color and one poker deck, and shuffle both. Your program will then play our game 10 times, reporting the result of each play to the console.

Class Design

Below is the UML representation of the set of classes that you are to implement for this lab. Note that we have introduced a couple of new bits of notation:

- We are explicitly representing both our generic class with the generic type undefined **AND** our generic class with the generic type bound to some other type (e.g., `Card<E>` vs `Card<MyColor>`).
- Although we will explicitly implement the generic class (`Card<E>`), we don't provide an explicit implementation of the bound class (`Card<MyColor>`) – the compiler does this for us! Hence, the bound class is labeled as "`<<implicit>>`".
- The lines labeled "`<<binds>>`" tell us explicitly what the binding is for our generic type.



The key classes in our project are:

- **MyColor** is an enumerated data type that defines a set of colors.
- The **Card** generic class is defined by a generic type, **E**. The two arrows from `Card<MyColor>` and `Card<Integer>` to **Card** indicate that the generic type of **Card** is bound in two different ways: one with **MyColor** and the other with **Integer**.

- **Deck** is an abstract class that is defined by a pair of generic types: **E1** and **E2**. **E2** defines the underlying generic type of **Card** (in our case, **Integer** or **MyColor**) and **E1** is a form of **Card** (specifically, **E1** is-a `Card<E2>`).
- The lines from **ColorDeck** and **PokerDeck** to **Deck** indicate that **ColorDeck** and **PokerDeck** must implement **Deck**.

Lab 7: Implementation Steps

Start from the class files that are provided in lab7-initial.zip.

1. The classes **Card**, **MyColor** and **PokerDeck** have been fully implemented and **should not be modified**.
2. The class **Deck** is a generic and abstract class. This class is partially implemented for you. You will need to complete the implementation of each method provided.
3. Create and implement the class **ColorDeck**. The method *populateDeck()* should populate the deck by placing a full set of cards into the deck and shuffling. Specifically, this method should place two each of type *RED* and *BLUE*.
4. Create and implement the class **Game**. The method *playOnce()* plays one game and reports whether the player wins or loses.
5. Create and implement the class **Driver** that contains your **main** method. It should play the game 10 times and report the results of each play.
6. Implement a JUnit test class *Lab7Test* to thoroughly test all of your code
    - You need to convince yourself that everything is working properly
    - Make sure that you cover all the classes, methods and cases within the methods while creating your test. Keep in mind that we have our own tests that we will use for grading.

Final Steps

1. Generate Javadoc using Eclipse.
   - Select *Project/Generate Javadoc...*
   - Make sure that your project (and all classes within it) is selected
   - Select *Private* visibility
   - Use the default destination folder
   - Click *Finish*.
2. Open the *lab7/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that all of your classes are listed and that all of your documented methods have the necessary documentation.
3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Sunday, October 11th.
- Prepare your submission file:
   1. Select the project in the *Package Explorer* window.
   2. Right-click. Select *Export*
   3. Select *General/Archive File*
   4. Expand your project and verify that both the *src* and *doc* folders are selected, as well as all of their contents
   5. Enter the archive file name: *lab7.zip* (note that you may want to browse to a different destination folder)
   6. Select *Save in zip format*
   7. Click *Finish*
- Submit your zip file to the lab7 folder on D2L.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 35 points

Program formatting: 5 points

(5) The program is properly formatted (including indentation, curly brace and semicolon locations).

(3) There is one problem with program formatting.

(0) The program is not properly formatted.

Data types and method calls: 10 points

(10) The program is using proper data types and method calls.

(7) There is one error in data type or method call selection.

(4) There are two errors in data type or method call selection.

(0) There are three or more errors in data type and method call selection.

Required Methods: 10 points

(10) All of the required methods are implemented.

(7) A component of one required method is missing.

(4) A method is not implemented or components are missing from two methods.

(0) Two or more required methods are not implemented or components from three or more methods are missing.

Unit Tests: 10 points

(10) A full set of unit tests has been implemented.

(8) One key component is not tested properly by the unit tests.

(6) Two key components are not tested properly by the unit tests.

(4) Three key components are not tested properly by the unit tests.

(2) Four key components are not tested properly by the unit tests.

(0) Five or more required methods are not implemented or components from three or more methods are missing.

Proper Execution: 30 points

Output: 15 points

(15) The program passes all tests.
(10) The program fails one test.
(5) The program fails two tests.
(0) The program fails three or more tests.

Execution: 15 points

(15) The program executes with no errors.
(8) The program executes, but there is one minor error.
(0) The program does not execute.

Documentation and Submission: 35 points

Project Documentation: 5 points

(5) The java file contains all of the required documentation elements at the top of the file.

(3) The java file is missing one of the required documentation elements.

(2) The java file is missing two of the required documentation elements.

(0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 10 points

(10) Every method contains all of the required documentation elements ahead of the method prototype.

(7) The method documentation is missing one of the required documentation elements.

(3) The method documentation is missing two of the required documentation elements.

(0) The method documentation is missing more than two of the required documentation elements.

Inline Documentation: 10 points

(10) Every method contains appropriate inline documentation.

(7) There is one missing or incorrect line of inline documentation.

(3) There are two missing or incorrect lines of inline documentation.

(0) There are more than two missing or incorrect lines of inline documentation.

Submission: 10 points

(10) The correct zip file name is used and has the correct contents.

(5) The correct zip file name is used, but one required component is missing.

(0) An incorrect zip file name is used or more than one required component is missing.