Introduction to Programming: Mini-Project 2

Following the American Psychological Association’s Guidelines

Markham M. Shofner

Johns Hopkins University

Introduction to Programming: Mini-Project 2

Design and Analysis:

1. General program design.
   1. The program is organized following the principles outlined in the first portion of modules in “Intro to Programming using Java”. The functionality of the program is written in an object-oriented manner, defining classes and instances, where the data contained therein in manipulated and managed by methods within those structures.
   2. The major data structures used are:
      * The main() method, which initiates an instance of the project class, and triggers the raceController() function.
      * The raceController() method, which initiates the contestants, the track, and manages the state of the race.
      * The contestants (Hare and Tortoise classes) which return moves based on random numbers.
      * The Track class, that keeps track of the Boolean status of the race, the winning condition, the position of the contestants, and can also display the current state of the race track.
2. Alternative approaches considered and reason for rejection:
   1. I considered using some sort of Contestant superclass that the Hare and the Tortoise would both branch off of, but ended up not using this option because the inheritance implementation become more complicated than I believe the situation needed the code to be.
3. Things learned from the project, and things that could have been done differently:
   1. Finding a way to properly manage the state of the program was a critical win in the early stages of writing the program (before getting into the deeper more specific parts of the code). There are probably more elegant ways to manage program state, and I look forward to building a more mature understanding of these nuances.
   2. I think I gained a better understanding of where certain functionality should reside within a program (track functions in the track class, move functions in the contestant classes, etc…) but at times it felt like functions could reside in more than one place (for example the display track function) but that it comes down to the coder’s decision of where is the most correct place for a function to be (and where it will be easiest to track down, expand, and debug in the future).

-------------------------------------------------------------------

1) Include a UML class diagram of your solution. Be sure to show the important associations between classes. The diagram should be pasted into a .docx or a .pdf file.

2) What alternative design approaches were considered and why were they rejected?

Each public class must be contained in a separate Java source file.

Only one source file will have a main() method and this source will be named BlackjackGameSimulator.java.

~~Other source/class names are up to you following the guidelines specified so far in the course.~~

~~The format of the Java source must meet the general Java coding style guidelines discussed so far during the course.~~

~~Pay special attention to naming guidelines, use of appropriate variable names and types, variable scope (public, private, protected, etc.), indentation, and comments.~~

~~Please use course office hours or contact the instructor directly if there are any coding style questions.~~

Classes and methods should be commented with JavaDoc-style comments (see below).

JavaDocs: Sources should be commented using JavaDoc-style comments for classes and methods.

Each class should have a short comment on what it represents and use the @author annotation.

Methods should have a short (usually 1 short sentence) description of what the results are of calling it.

Parameters and returns should be documented with the @param and @return annotations respectively with a short comment on each.

JavaDocs must be generated against every project Java source file.

They should be generated with a - private option (to document all protection-level classes) and a –d [dir] option to place the resulting files in a javadocs directory/folder at the same level as your source files.

See the JavaDocs demonstration for more details.

This project involves writing a program to simulate a blackjack card game.

You will use a simple console-based user interface to implement this game.

A simple blackjack card game consists of a player and a dealer.

A player is provided with a sum of money with which to play.

A player can place a bet between $0 and the amount of money the player has.

A player is dealt cards, called a hand.

Each card in the hand has a point value.

The objective of the game is to get as close to 21 points as possible without exceeding 21 points.

A player that goes over is out of the game.

The dealer deals cards to itself and a player.

The dealer must play by slightly different rules than a player,

and the dealer does not place bets.

A game proceeds as follows:

A player is dealt two cards face up.

If the point total is exactly 21 the player wins immediately.

If the total is not 21,

the dealer is dealt two cards, one face up and one face down.

A player then determines whether to ask the dealer for another card (called a “hit”) or to “stay” with his/her current hand.

A player may ask for several “hits.”

When a player decides to “stay” the dealer begins to play.

If the dealer has 21 it immediately wins the game.

Otherwise, the dealer must take “hits” until the total points in its hand is 17 or over, at which point the dealer must “stay.”

If the dealer goes over 21 while taking “hits” the game is over and the player wins.

If the dealer’s points total exactly 21, the dealer wins immediately.

When the dealer and player have finished playing their hands, the one with the highest point total is the winner.

Play is repeated until the player decides to quit or runs out of money to bet.

References

Schildt, Herbert (2014). Java The Complete Reference – Ninth Edition.