Assignment 2, CPS109

Based on Programming Project 7.1, page 331 (3E), page 323 (4E), page 372 (5E) of Big Java.

Poker Simulator. In this assignment, you will implement a simulation of a popular casino game usually called video poker. The card deck contains 52 cards, 13 of each suit. At the beginning of the game, the deck is shuffled. You need to devise a fair method for shuffling. (It does not have to be efficient.) Then the top five cards of the deck are presented to the player. The player can reject none, some, or all of the cards. The rejected cards are replaced from the top of the deck, so that the player has again five cards. The rejected cards can go in a discard pile so that they are not re-used until a new deck is initiated and shuffled, which would occur when the current deck is empty, or if you prefer, you can start with a new shuffled deck at the beginning of each new hand. Now the hand is scored. (After scoring the hand, the hand can go on the discard pile, along with the rejected cards). Your program should pronounce the hand to be one of the following:

- **No pair**: The lowest hand, containing five separate cards that do not match up to create any of the hands below.
- **One pair**: Two cards of the same value, for example two queens.
- **Two pairs**: Two pairs, for example two queens and two 5's.
- **Three of a kind**: Three cards of the same value, for example three queens.
- **Straight**: Five cards with consecutive values, not necessarily of the same suit, such as 4, 5, 6, 7, and 8. The ace can either precede a 2 or follow a king.
- **Flush**: Five cards, not necessarily in order, of the same suit.
- **Full House**: Three of a kind and a pair, for example three queens and two 5's
- **Four of a Kind**: Four cards of the same value, such as four queens.
- **Straight Flush**: A straight and a flush: Five cards with consecutive values of the same suit.
- **Royal Flush**: The best possible hand in poker. A 10, jack, queen, king, and ace, all of the same suit.

Implement a wager. The player pays a JavaDollar for each game, and wins according to the following payout chart:

Hand	Payout	Hand	Payout
Royal Flush	250	Straight	4
Straight Flush	50	Three of a Kind	3
Four of a Kind	25	Two Pair	2
Full House	6	Pair of Jacks or Better	1
Flush	5		

Thus, if at the end of one hand, the player had a Full House, then the player has paid 1 JavaDollar and won 6 JavaDollars, so the player has 5 more JavaDollars than his/her initial amount. You can decide what the initial amount is.

Your program should state how much the money the player won on the current hand, and how much the player has won overall (minus the cost of playing).
Graphics is not required for this assignment and there are no bonus points for graphics. We are not looking for anything fancy in terms of output, but rather good programming style that does the job required.

Marking:
1 point: Use of javadoc comments for every class, method and constructor.
1 point: Use of proper indentation, proper capitalization for variable names and classes and methods, proper mnemonic variable names, use of constants rather than magic numbers.
1 point: Use of object-oriented approach, so that there is a separate class for each significant entity in this problem. (Not counting the tester and simulator classes, there are at least three obvious classes).
2 points: The tester class (called PokerTester) which sets up some situations (hands or decks) and runs methods of objects, printing the result and printing the "Expected:" values, so that we can see that the methods are doing what they are supposed to do. The tester class does not have to interact with the user.
2 points: The simulator class (called PokerSimulator) interacts with the user, playing as many hands as the user wants, and keeping track of how much money has been won. This class does the I/O and controls the run. There should be no I/O from any class other than the simulator and the tester.
3 points: The other classes that do the real work.

What to hand in on blackboard:

1. the java files (not the class files)
2. exampleRun.txt (a text file showing an example run of the simulator) (*deduction -1 if you omit this or if you submit instead a .doc or .docx or .rtf or anything but a simple .txt file*)
3. exampleTest.txt (a text file showing the output from the tester class) (*deduction -1 if you omit this or submit a format other than a simple .txt file.*)