CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 2: Poker

1 Introduction

Consider a simplified version of a traditional five card poker game. For this game there are only two players (a human and a computer), and each player will only have two cards. To start a hand (a round), each player will be given their two cards face down (hidden from the other player). There is no betting, instead each player, in turn, can choose to fold (quit this hand) or stand (play out the hand). The order that the players get to make this choice will flip each hand (the first hand the human must make the choice first, for the second hand the computer goes first). If either player folds, the other player wins (so if the player with the first choice folds, the player with the second choice does not have to make a choice, they win automatically). If both players stand, they reveal their cards and a winner is determined based on the following rules:

- For this game we will only consider three possible families of *hands* (cards held by a player) ordered below from highest to lowest:

  1. Pair (two cards with the same rank)
  2. Flush (two cards with the same suit)
  3. High Card (two cards with different ranks and suits)

- Hands in a higher family beat hands of a lower family.

- If both players have hands in the same family, then the winner is determined by comparing the highest ranking card of each hand. If those are equal, then the lower ranking cards are compared. If both cards are equal, then the hand is a tie.

- The ranks are ordered from high to low as
  Ace, King, Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3, 2 ("deuce" – lowest)

1.1 Goals

Students will gain experience with the fundamentals of object-oriented programming in relation with basic class concepts such as:

- Classes vs Objects
- State vs Behavior
- Encapsulation
- Constructors and Instantiation
- Accessor and Mutators
- Overriding methods in `Object`, i.e `toString`
- UML Class Diagrams
- Test Driven Development using JUnit

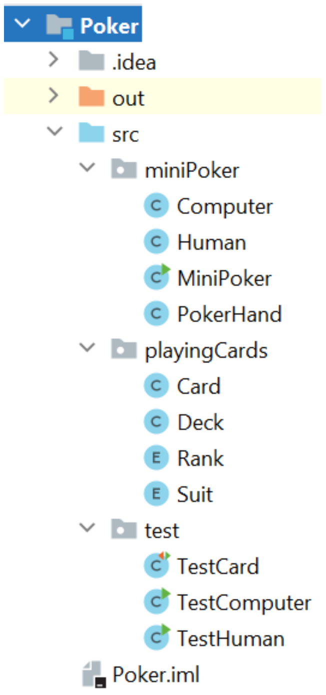
1.2 Provided Files

1. The **class_diagram.png** illustrates the program's design.
2. The **sample_run.txt** contains sample runs.
3. The **src.zip** contains the starter code.
4. The Javadoc documentation for the classes here.

2 Implementation

2.1 Project Structure

Your project should be structured as follows:



2.2 Design

Although there are several different ways that these classes could be written, for this homework, we are requiring you to closely follow the supplied class diagram (see provided files section). For more information about these classes and their methods, see the javadoc provided.

Notice that both this diagram, and the java documentation, exclude the private state and private behavior. You will need to design that on your own.

To understand the UML class diagram, read here.

2.3 Stand / Fold behavior

Human Player

The human has to query whether they want to stand or not. You should use the `Scanner` that is passed in from the main program, `Poker` in your `Human` class. The user should be prompted to enter "y" if they want to stand, or "n" if they want to fold.

Computer Player

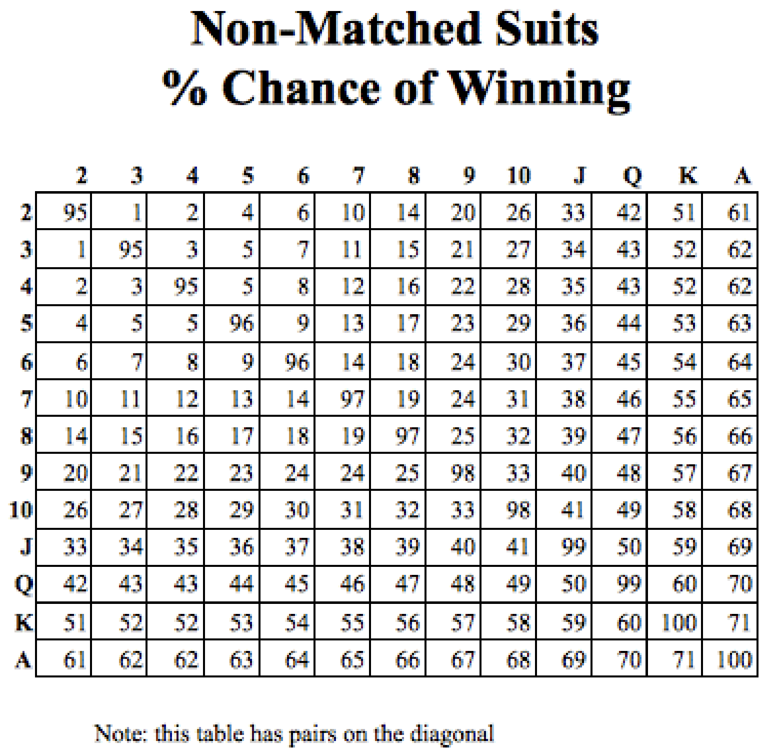
A computer will stand if statistically speaking their hand has a 50 percent chance or better of winning. Take a look at this table to see what the chance of winning is for a flush. The lowest probability for a flush hand has a 71 percent chance of winning (a Two and Three). This means a computer will stand on any flush hand.

Flush
% Chance of Winning

	2	3	4	5	6	7	8	9	10	J	Q	K	A
2	0	71	71	72	73	74	75	77	79	82	84	87	91
3	71	0	71	72	73	74	76	78	80	82	85	88	91
4	71	71	0	72	73	75	76	78	80	82	85	88	91
5	72	72	72	0	74	75	76	78	80	83	85	88	92
6	73	73	73	74	0	75	77	78	81	83	86	89	92
7	74	74	75	75	75	0	77	79	81	83	86	89	92
8	75	76	76	76	77	77	0	79	81	84	86	89	93
9	77	78	78	78	78	79	79	0	81	84	87	90	93
10	79	80	80	80	81	81	81	81	0	84	87	90	93
J	82	82	82	83	83	83	84	84	84	0	87	90	94
Q	84	85	85	85	86	86	86	87	87	87	0	90	94
K	87	88	88	88	89	89	89	90	90	90	90	0	94
A	91	91	91	92	92	92	93	93	93	94	94	94	0

Note: this table's diagonal is impossible

Take a look at this table of probabilities for a pair and "High card" hands. Obviously since a pair is even better than a flush, the computer will stand on all pairs. But now look at the "High card" combinations. When the player has a high card of Queen and a low card of Jack, they have exactly a 50 percent chance of winning. The computer should fold if they have anything lower than that combination.



Non-Matched Suits
% Chance of Winning

	2	3	4	5	6	7	8	9	10	J	Q	K	A
2	95	1	2	4	6	10	14	20	26	33	42	51	61
3	1	95	3	5	7	11	15	21	27	34	43	52	62
4	2	3	95	5	8	12	16	22	28	35	43	52	62
5	4	5	5	96	9	13	17	23	29	36	44	53	63
6	6	7	8	9	96	14	18	24	30	37	45	54	64
7	10	11	12	13	14	97	19	24	31	38	46	55	65
8	14	15	16	17	18	19	97	25	32	39	47	56	66
9	20	21	22	23	24	24	25	98	33	40	48	57	67
10	26	27	28	29	30	31	32	33	98	41	49	58	68
J	33	34	35	36	37	38	39	40	41	99	50	59	69
Q	42	43	43	44	45	46	47	48	49	50	99	60	70
K	51	52	52	53	54	55	56	57	58	59	60	100	71
A	61	62	62	63	64	65	66	67	68	69	70	71	100

Note: this table has pairs on the diagonal

You will need to use a formula that orders the values of all the possible card combinations. One approach is to use a method that favors the high card, and scales accordingly for pair and flush:

- Pair: `HIGH_CARD_VALUE * 14 + LOW_CARD_VALUE + 1000`
- Flush: `HIGH_CARD_VALUE * 14 + LOW_CARD_VALUE + 500`
- High card: `HIGH_CARD_VALUE * 14 + LOW_CARD_VALUE`

Using this formula, you must compute the value for a Queen/Jack non-flush. That value will work as a minimum threshold. The computer must fold for any value below it.

```
{ Queen, Jack }
= QUEEN_VALUE * 14 + JACK_VALUE
= 12 * 14 + 11
= 179
```

When you go to implement this in your `Computer` class, don't just hardcode this value, `179` (in case the values of cards changes). Instead use the values from the `Rank` enum to compute and define a constant for it.

2.3.1 Sample Run

You have been provided with a sample run of the final program for your reference. Keep in mind that

- Each run will be different since the cards are randomly shuffled.
- The provided sample run always shows your (the human's) cards first. You can tell to whom the cards were first dealt by seeing who was asked to stand or fold.
- This is just a suggested output format. As long as when your program is run and the necessary information is displayed to play the game, you will get credit. Just make sure it is easy to follow.

3 Testing

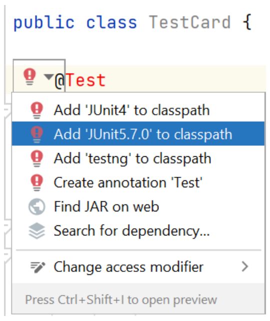
Because we are providing a complete JUnit set of tests for your homework, it is suggested you use the Bottom-up approach for developing. This means you start with the most basic classes that have no other dependencies, and then work your way up. Each time you develop a new class you should run the test cases and make sure that the module you just developed, plus the previous ones, all work correctly. This software development process is referred to as Test Driven Development.

3.1 JUnit

We are provided you with a tests folder that has 3 unit tests files for the core classes, `TestCard`, `TestComputer`, `TestHuman`. Import all those files in a `test` package in your project.

JUnit comes with IntelliJ but it is not in the path. Once you have stubbed out all the classes in your `playingCards` and `miniPoker` packages you can set up JUnit.

To set up JUnit, go to the TestCard source code. Highlight over a @Test annotation, hit the ALT + ENTER keys and select Add 'JUnit5' to the classpath.

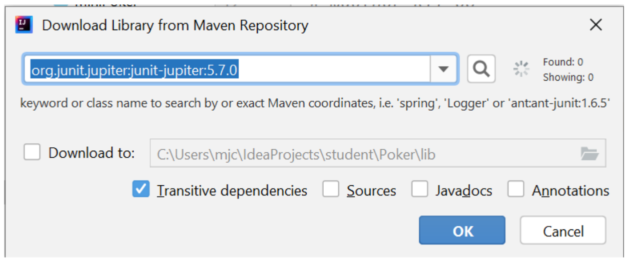


```
public class TestCard {

    @Test
```

Add 'JUnit4' to classpath
Add 'JUnit5.7.0' to classpath
Add 'testng' to classpath
Create annotation 'Test'
Find JAR on web
Search for dependency...
Change access modifier
Press Ctrl+Shift+I to open preview

Select that and a dialog should come up to download it - click the OK button.


Download Library from Maven Repository

org.junit.jupiter:junit-jupiter:5.7.0

Found: 0
Showing: 0

keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'

Download to: C:\Users\mjc\IdeaProjects\student\Poker\lib

[✓ Transitive dependencies] Sources Javadocs Annotations

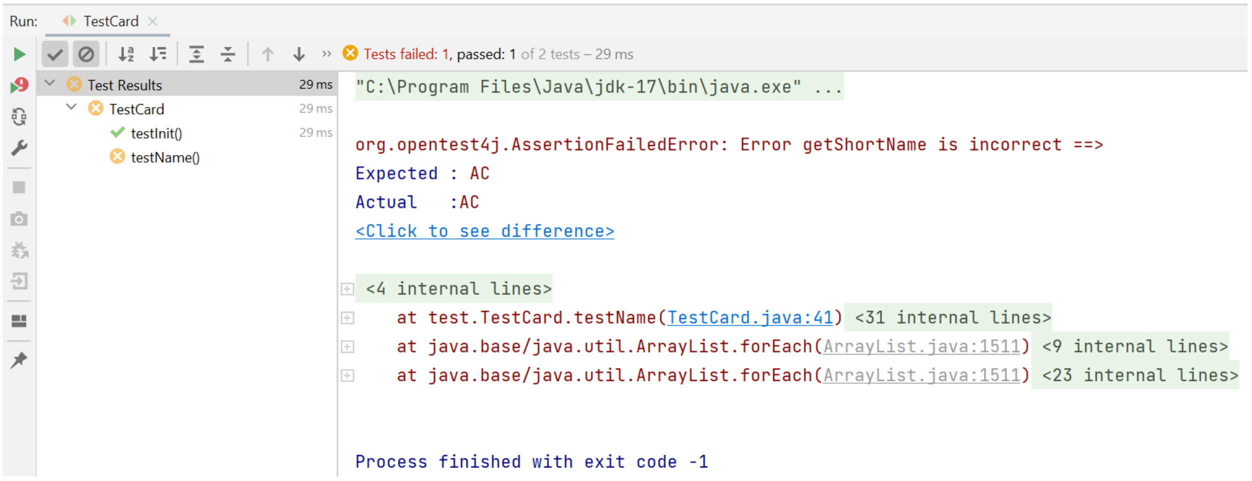
OK Cancel

At this point there should be no errors related to JUnit. You may have still have errors if you haven't implemented the `Card` class yet.

3.1.1 Run Tests

You can run all the tests at once by right clicking on the `test` package, or you can run each test case individually by right clicking on it.

When a test fails it shows you what it Expected, and the Actual which is what your program produced. These two things must match exactly in order to pass the test. If strings are being compared they must match character by character exactly.


Run: TestCard

Tests failed: 1, passed: 1 of 2 tests – 29 ms

```
Test Results        29 ms
  TestCard          29 ms
    testInit()      29 ms
    testName()
```

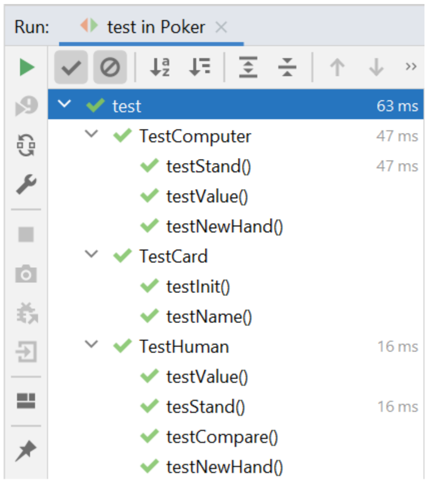
```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...

org.opentest4j.AssertionFailedError: Error getShortName is incorrect ==>
Expected : AC
Actual   :AC
<Click to see difference>

<4 internal lines>
   at test.TestCard.testName(TestCard.java:41) <31 internal lines>
   at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
   at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>


Process finished with exit code -1
```

Your goal is to make sure you pass all the tests, e.g.


Run: test in Poker

```
test                63 ms
  TestComputer      47 ms
    testStand()     47 ms
    testValue()
    testNewHand()
  TestCard
    testInit()
    testName()
  TestHuman         16 ms
    testValue()
    tesStand()      16 ms
    testCompare()
    testNewHand()
```

4 Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your src folder as a ZIP archive named “hw2.zip” (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework).

5 Grading

The grade breakdown for the entire assignment is:

- Design: 15%
  Criteria:
  - Well-focused set of responsibilities for each class and function
  - Minimal interdependencies between classes
- Functionality: 75%
- Documentation - Code Format, Style: 10%