1 Java Practice

To continue to give you practice with the syntax of Java, for the first part of this assignment, you’re going to write a few stand-alone methods. Create a new project in Eclipse called assign2.2 (look at assignment 1, part 1 if you need a refresher on how to do this). Create a new class called WarmUp, which you will put your methods into.

Write the following as public static methods in your WarmUp class:

1. Write a method called countHearts that takes an array of Cards as a parameter and counts how many of the cards have the suit “hearts”.

   For example, if we ran the following code:

   ```java
   Card[] cards = new Card[4];
   cards[0] = new Card(1, "clubs");
   cards[1] = new Card(1, "hearts");
   cards[2] = new Card(3, "hearts");
   cards[3] = new Card(2, "diamonds");
   System.out.println(countHearts(cards));
   ```
2. Write a method called `addArraysSameLength` that takes two arrays of `doubles` as parameters. The method should make a new array where the \( i \)th entry in this array represents the sum of the \( i \)th entries of the two input arrays and return this new array. You may assume that the arrays have the same length.

For example, if we ran the following code:

```java
double[] array1 = {1, 2, 3, 4, 5};

double[] answer = addArrays(array1, array1);
System.out.println(Arrays.toString(answer));
```

we would get \([2.0, 4.0, 6.0, 8.0, 10.0]\).

3. Write a method called `reverseArray` that takes an array of `Strings` as a parameter and reverses the order of the array. It should NOT return a new array. Instead, it should mutate (i.e. change) the array that was input.

For example, if we ran the following code:

```java
String[] words = {"I", "love", "my", "CS", "classes", "!"};

System.out.println("Before: " + Arrays.toString(words));
reverseArray(words);
System.out.println("After: " + Arrays.toString(words));
```

we would see:

```
Before: [I, love, my, CS, classes, !]
After: [!, classes, CS, my, love, I]
```

4. **OPTIONAL:** If you want some extra practice (NOT REQUIRED), try writing another function called `addArrays` that adds two arrays as described above, but removes the requirement that they be the same length. It should produce an array the size of the longest input array. If there isn't a corresponding entry in one of the arrays because the other is longer, then it should just use the value of the longer array.

For example, if we ran the following code:

```java
double[] array1 = {1, 2, 3, 4, 5};
double[] array2 = {10, 10};

double[] answer = addArrays(array1, array2);
System.out.println(Arrays.toString(answer));
```

we would get \([11.0, 12.0, 3.0, 4.0, 5.0]\).
For the rest of the assignment, we're going to write a game I call “Flippy Card”. When the game starts, you deal five cards face down. Your goal is to achieve as high a score as possible. Your score only includes cards that are face up. Red cards (hearts and diamonds) award positive points, while black cards (clubs and spades) award negative points. Cards 2-10 have points worth their face value, Jack, Queen and King 10 and Ace 11.

The game is played by flipping cards (either from face down to face up or from face up to face down). As you play, you are told your total score (i.e. total of the face up cards) AND the total score of the cards that are face down. The challenge is that you only get a fixed number of flips and then the game is over.

Here is an example of the output from playing the game:

FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 0
Face down total: -5
Number of flips left: 5
Pick a card to flip between 1 and 5 (-1 to end game): 1

Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 4
Pick a card to flip between 1 and 5 (-1 to end game): 2

Queen of hearts | Jack of clubs | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 0
Face down total: -5
Number of flips left: 3
Pick a card to flip between 1 and 5 (-1 to end game): 2

Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 2
Pick a card to flip between 1 and 5 (-1 to end game): 10

10 is not a valid card
Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 2

1If you have a better name, please, please let me know :)

2 Flippy Card: The Game
Pick a card to flip between 1 and 5 (-1 to end game): 0

0 is not a valid card
Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 2
Pick a card to flip between 1 and 5 (-1 to end game): 5

Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | 3 of hearts
Face up total: 13
Face down total: -18
Number of flips left: 1
Pick a card to flip between 1 and 5 (-1 to end game): -1

----------------------
Your score: 13
Best possible score: 15

At each point where the program asks the user to pick a card, the game waits for user input.

3 Requirements

Below are the requirements for your Flippy Card game program:

- The code should be in a class called FlippyCard
- The class should have one constant for the number of cards (5).
- You should include a constructor (i.e. a method called public FlippyCard) for the game that takes the number of flips for the game as a parameter.
- You should have one public method called playGame. When called, the game should start playing and continue until the game finishes.
- You must also implement at least three private methods, although I encourage you to implement more. These methods should help make your code more readable and make it so that all of the code isn’t in the playGame method. For these methods, make sure to make proper use of the instance variables.
- None of the above methods should be static!
- You must have at least one instance variable (and likely will need more).
- You must use the CardDealer class from class to get the cards for the game.
• Your game must follow the interaction defined above.

• Your game should handle incorrect input properly (that is numbers that are not -1 or 1-5). This should NOT count towards their tries. You may assume, however, that the user always enters an integer.

• When the game finishes, you should print out the player’s score and the best possible score they could have gotten (the sum of all of the red cards).

4 Card Maintenance

To make our lives easier, we’re going to add a few extra methods to the Card class. You may either use your version from the previous assignment (you’ll have to rename it from MyCard to Card), or you can download a copy that I put together:

http://www.cs.middlebury.edu/~dkauchak/classes/cs201/assignments/assign2/part2/Card.java

Add the following methods to the Card class:

• A method called isRedCard that returns whether or not the card is red (i.e. “hearts” or “diamonds”).

• A method called getCardValue that gives us the numerical value that the card is worth, that is the face value for 2-10, 10 for jack, queen and king, and 11 for aces.

We could have just implemented this functionality in our game class, however, it’s better to put it in the Card class. It asks questions about data in the Card class, we can reuse it for other applications if we’d like, and it can be more easily implemented in the Card class.

5 The Rest

Now that the Card class is fixed up, it’s time to write the game. I’m going to give you a fair amount of flexibility in how you implement it as long as it meets the requirements specified above. Regardless of how you implement it, though, I strongly suggest that you incrementally build up the functionality of your program. Do NOT try and code it all up at once. Instead, get some very basic things working, test them, then add the next feature.

Here is my suggestion on how to implement it:

– Before coding, think a bit about the design of your class. What methods do you want to have? How should the flow of the program work? What instance variables do you need, that is, what data do you need to keep track of as the game is played? What parameters should the constructor have? I usually spend 10-15 minutes thinking about this before coding and it can save you a lot of headache down the road.
• Create the CardDealer class in Eclipse and copy the source code from the class web page. Make sure you understand how to use this class to get new cards. Look at the static methods in this class if you’re unsure.

• Create the FlippyCard class in Eclipse.
  – Add the constants and instance variables.
  – Write the constructor.
  – Write a private method that prints the current cards to the screen. If a card is face down, it just displays “FACE-DOWN”. If it is face up, then it shows the actual card. You can either do this by generating one string and then printing it out, or you can use the System.out.print method, which doesn’t print a newline after. Test this method and make sure it works properly.
  – Write a very basic version of the playGame function that displays the cards, gets a card choice from the user (don’t worry about checking for invalid numbers for now), flips that card and then loops for a fixed number of tries. Again, test to make sure it works.
  – Add functionality to print out the totals for the face up and face down cards. (Hint: this might be a good place for some more methods.
  – Add code that prints out the the user’s score after finishing and the best possible score. The best possible score is just the sum of all of the red cards. (Hint: this might be a good place for some methods.
  – Add code to check for valid input. You can still do your looping with a for loop, but it may be more natural to tackle this with a while loop. Think through this one! There are some very nice ways of solving this that won’t require a lot of changes to your code.
  – Add code to handle exiting early when -1 is entered.
  – Test is out some more, make sure your code is commented well, and play the game!

Once you’re done, you should be able to run your game as:

```java
default public static void main(String[] args) {
    FlippyCard game = new FlippyCard(5);
    game.playGame();
}
```

6 When You’re Done

You should now have four files in your project (WarmUp, Card, CardDealer and FlippyCard). Put comments at the top of each of these files that you wrote with your name and the assignment number.

To group these files into a single file, you need to export your project. To do this:
1. Right-click on the project (on Mac, ctrl+click) and select Export.

2. You’ll see a number of options. Open up the Java folder and select JAR file and click Next.

3. You should just see your project selected. Below, make sure only the following two options are checked:
   - “Export Java source files and resources”
   - “Compress the contents of the JAR File”

4. Click on the Browse... button and pick a location to save the output file. Give the file a name like “kauchak1.jar”, where “kauchak” is your last name and “1” is the assignment number.

5. Click Finish.

If all went well this will generate a single .jar file wherever you picked to save it.

Submit this JAR file as assignment number “2.2” via the online submission mechanism on the course web page.