SYSC 2004 Object-0riented Software Development

Lab 6

Lab 6:

Background Reading

- *Objects First with Java*, Chapters 1-4, 5.4, 6, and 7.

Objective

The objective of this lab is for you to develop some classes for use in card games. The classes you'll build are:

`Card`: models a playing card.
`Deck`: models a deck of 52 cards
`Hand`: models the cards held by one player

Incomplete implementations of these classes are posted in a zip file on cuLearn, along with three JUnit test classes (`CardTest`, `DeckTest`, and `PileTest`) .

Getting Started

1. Download file `cards.zip` from cuLearn. Save the file to the desktop.
2. Right-click on the `cards.zip` folder and select Extract All... to extract all the files into a folder called `cards`.
3. Launch BlueJ.
4. Open the *cards* project in BlueJ.

Part 1 - Class `Card`

An Anglo-American deck of playing cards contains 13 cards for each of four suits (hearts, diamonds, clubs and spades), for a total of 52 cards. Each of the 13 cards in a given suit has a unique rank (ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king). The objective of this part is to implement a Java class that models these playing cards.

1. Double-click the icon for class `Card` to open the incomplete implementation of the class in the BlueJ editor. **Make sure you update the documentation at the beginning of the class** (i.e., add your name as an author, and update the version number and date). Read the documentation for the constructor and methods. Notice that the fields are called `suit`

and `rank`, and the constructor parameters have the same names. Type the following statements in the body of the the `Card` constructor (there shouldn't be any other statements in the constructor):

```
rank = rank;
suit = suit;
```

Do you think the constructor will correctly initialize the newly created card? Compile `Card`. After the class compiles without errors, create a new `Card` object and place it on the object bench. Use the object inspector to examine the object's fields. Was your hypothesis correct? Did the constructor correctly initialize the `Card` object?

2. Edit the constructor body to look like this:

   ```
   this.rank = rank;
   this.suit = suit;
   ```

   Compile `Card`. Create a new `Card` object and place it on the object bench. Use the object inspector to examine the object's fields. Did the constructor correctly initialize the `Card` object? Is variable `this.rank` the field called `rank` or the parameter called `rank`? Is variable `rank` the field called `rank` or the parameter called `rank`? You can confirm your answer by reading Section 3.12.2. If a constructor or a method has a parameter with the same name as a field, how do we write statements that correctly use both variables?

3. Write the `rank()` and `suit()` methods. Be sure to add the "@return" javadoc statements. Don't write `hasSameRank()` or `isEqualTo()` yet.
4. Compile unit test class `CardTest`. Run the test cases in `CardTest` (right-click on the `CardTest` icon and select Test All from the pop-up menu). `testCreateAllCards` should pass, but the other test cases will fail (you haven't written the methods that those test cases exercise). If necessary, modify your class until it passes `testCreateAllCards`.
5. Write `hasSameRank()`. Don't forget the "@param" and "@return" java doc statements. Run all the tests in `CardTest`. `testSameRank` should now pass.
6. Write `isEqualTo()`. Remember that you must use the "equals" method (not "==" to compare Strings). Don't forget the "@param" and "@return" java doc statements. Run all the tests in `CardTest`. `testIsEqualTo` should now pass.
7. Generate the javadoc (from the BlueJ window, select Tools, then Project Documentation) and ensure that it is complete and correct. Show your javadoc, test cases passed, and code to the TA for part 1.

Part 2 - Class `Deck`

1. Open the incomplete implementation of `Deck` in the BlueJ editor and read the documentation for the constructor and methods. Notice that this class has a single field called `cards`. **The class must not define any additional fields - the fields called `cards` is sufficien**t. Of course, your constructor and methods may need to define one or more local variables. Don't litter your class with magic numbers. Several constants have been defined in this class - make sure you use them. **Make sure you update the documentation at the beginning of the class** (i.e., add your name as an author, and update the version number and date).
2. Write constructor `Deck()`, `buildSuit()`, `isEmpty()` and `size()`. Don't forget the javadoc comments (@param, @return). The constructor must call private method `buildSuit()` four times (once for each suit). Each time `buildSuit()` is called, 13 new cards will be added to the deck. The constructor does not shuffle the deck of cards. Compile the `Deck` class. After the class compiles without errors, compile `DeckTest`. Run the test cases in `DeckTest`. `testCreateDeck` should pass, but the other test cases will fail.
3. Write `dealCard()`. `dealCard()` should always remove and return the card at the front of the deck, i.e. the first card in the ArrayList. It should not randomly pick the card it will remove. Run all the tests in `DeckTest`. `testDeal` and `testHas52Cards` should now pass. If both tests fail, the problem is probably in your `dealCard()` method. If `testDeal` passes but `testHas52Cards` fails, the problem is probably in your constructor or `buildSuit()` method (you're likely not initializing the deck with 52 different cards). Don't forget the javadoc comments (@param, @return).
4. Write `shuffle()`. The method should create one instance of `java.util.Random()` and repeatedly (**at least 10,000 times**) invoke `nextInt(int n)` to generate the positions of two cards in the deck to interchange. Run all the tests in `DeckTest`. `testShuffle` should now pass. Don't forget the javadoc comments (@param, @return).
5. Generate the javadoc and ensure that it is complete and correct. Show your javadoc, test cases passed, and code to the TA for part 2.

Part 3 - Class `Hand`

1. Open the incomplete implementation of `Hand` in the BlueJ editor and read the documentation for the constructor and methods. Complete the implementation of the class, including the javadoc comments. As you write each method, run the tests in `HandTest` to provide you with feedback.

   Notes:

   a. The class must not define any additional fields - the fields called `cards` is sufficient. Of course, your methods may need to define one or more local variables.

   b. Most of the methods in `Hand` will be very short; for the most part, they will simply call the appropriate method on the hand's `ArrayList` object.

   c. Notice that the `String` returned by `toString()` should have a single space between each number (card rank), and should have no leading or trailing spaces.

2. Generate the javadoc and check that it is complete and correct. Show your javadoc, test cases passed, and code to the TA for part 3.
