

CS1310 Coursework ONE

Deadline: Tuesday 25 November 2008

The task in this coursework is to implement a program that can be used to play a card game called “Play your cards right”. The game is a two player game and each player gets five cards drawn at random from a pack and placed face down. Each player takes turns at playing.

To play, the first card is revealed and the player whose turn it is must guess whether the next card is higher or lower. If they guess correctly, they move on to the next card. One point is scored for each correct guess. The player’s turn ends when either:

1. There are no more cards to guess (score 4)
2. The player chooses to “freeze” and take their current score
3. The player guesses wrong and scores 0.

A BlueJ project is available on BlackBoard for you to use for this coursework. Three Java classes have already been completed for you. These are:

1. The class called **Reader** is used to read the commands that are typed at the keyboard. The **Reader** class has two methods:
 - (a) The method called **getCommand** asks the player to type a command and returns a String containing the text of that command. There are three valid commands which are; “higher”, “lower” and “freeze”. If the player types an incorrect command, he or she will be asked again to input a command.
 - (b) The method called **close** will close the input. You must call this method when a player has finished playing.
2. The class called **CommandWords** is used by the **Reader** class and you do not need to know anything about it.
3. The class called **PackOfCards** models a pack of 52 cards. It has a single method called **selectACard()** which returns a **Card** selected at random from the pack. Note that this class will not compile until you have completed the **Card** class.

What you need to do

There are three classes that you need to write, only one of which is provided with any existing code.

The Card class

The **Card** class has been started for you and the code contains four constants. You need to add two fields, one constructor and four methods as described below:

1. Create an integer field to hold the suit of the card. You can use the constants that have been defined for you to represent the four suits, hearts, spades, clubs and diamonds.

2. Create an integer field to hold the rank of the card. A card's rank is a number from 1 to 13 with 1 being an ace and 13 being a king.
3. The constructor for the **Card** class has a single parameter which is an integer between 0 and 51 (inclusive). You should initialise the suit and rank from the parameter whose value is interpreted as follows; 0 represents the ace of hearts, 1 represents the 2 of hearts, ..., 12 represents the king of hearts, 13 represents the ace of clubs, 14 represents the two of clubs, ..., 25 represents the king of clubs, 26 represents the ace of diamonds, 27 represents the two of diamonds, ..., 38 represents the king of diamonds, 39 represents the ace of spades, 40 represents the two of spades, ..., 51 represents the king of spades.
4. The method called **getCardName** has no parameters and returns the name given to the rank of this card as a **String**. If the card's rank is 1, the name is "Ace", 11 is "Jack", 12 is "Queen", 13 is "King" and the names of the other cards is simply the integer value of the rank returned as a **String** (ie "2", "3", etc).
5. The method called **getCardSuit** has no parameters and returns the suit of the card as a **String**. The method should return one of "Hearts", "Clubs", "Spades", or "Diamonds".
6. The method called **getRank** will return the rank of the card as an integer. The value returned will be between 1 and 13 (inclusive).
7. The method called **toString** returns a **String** describing this card. The return value should be of the form "*name of suit*", so valid **Strings** are "Ace of Spades", "King of Clubs", "3 of Diamonds", etc.

The Player class

The **Player** class has three fields:

1. A **String** representing the name of the player.
2. An array of **Card** objects, which initially does not contain any cards.
3. An integer representing the score of the player.

The constructor for the **Player** class has a single parameter whose value is the name of this player. The **name** field should be initialised with the name of the player, the array should be initialised to be an array that can hold five **Card** objects (no **Cards** are created until the player starts to play), and the player's score should be initialised to 0.

You need to write two accessor methods, **getScore** and **getName**.

The other method in the **Player** class is called **play** and has no parameters and does not return a value. It is a mutator method that will change the player's score:

- Construct a **PackOfCards** and fill the player's array by selecting 5 cards from the pack using a **while** loop. Create a private method to do this step and call the method from your **play** method.
- Construct a new **Reader** using a local variable.
- You need a boolean variable called **stillPlaying** which should be initialized to **true**

- The card whose index in the array is 0 should be revealed to the user by printing its `toString` value to the screen.
- You need a local variable that will be used as an index into the pack of cards. The value of this variable will represent the next card whose value must be guessed and should be initialised to 1.
- A `while` loop will continue iterating while `stillPlaying` is equal to `true`. On each iteration:
 - Read a command from the user (“higher”, “lower” or “freeze”)
 - Note that to compare the value of a string variable against another string, you must use the `equals` method of the `String` class and not `==`. For example:

```

1 String s1 = "what";
2 if (s1.equals("what")) {
3     ...
4 }

```

- If the user’s guess is correct, increment the user’s score by one and reveal the next card to the user by printing its `toString` value.
 - If the user’s guess is incorrect, set the user’s score to 0 and reveal the next card. Set `stillPlaying` to be `false`.
 - If the user wants to freeze, `stillPlaying` should be set to `false`
 - Before ending the loop, increment your count by one and if all the cards have been guessed, set `stillPlaying` to `false`.
- Before the end of the `play` method, close the `Reader` that you used to read this player’s commands.

The Game class

The `Game` class has two fields representing `Player` one and `Player` two.

The constructor for this class has two parameters which are the names of the two players. The constructor will initialise the two fields.

The `Game` class has a single method called `play` which has no parameters and does not return a value. This method should print out a suitable introduction and then:

1. Instruct player one to play first. The instruction should give the name of player one.
2. Invoke the `play` method of `Player` one.
3. Announce the score for `Player` one.
4. Instruct player two to play next. The instruction should give the name of player two.
5. Announce the score of `Player` two.
6. Print a message naming the winning player or announcing a draw depending on the scores of the two players.

Submission

The submission date is **Tuesday 25 November 2008**. Your work should be submitted to the coursework submission office.

You should submit the following printed on paper (ensure your name and student number is printed on each piece of paper submitted):

1. A listing of your **Card** class.
2. A listing of your **Player** class.
3. A listing of your **Game** class
4. Either:
 - (a) A printout from the BlueJ terminal window showing your game being played to the finish.
 - (b) A printout of any partial output in the BlueJ terminal window showing your game being played **PLUS** a written description of error, identifying the line number where any error occurs at run time or the line numbers of any compile time errors.

Marks will be lost if you only submit part of the required submission. It is your responsibility to ensure that all the lines of your code are visible in the printed copy that you submit. If the printer has cut off any long lines so that your code is not readable, you may lose marks.

You should also submit a ZIP file to BlackBoard, containing your finished BlueJ project. The ZIP file will be used to check that your program produces the output that you have submitted. The electronic submission is in *addition* to your paper submission and not an alternative.

Programming style

Your code should be neatly laid out and indented. Marks will be lost for poor indentation. Lines following an open curly bracket should be tabbed in by one tab and then tab out again when the closing curly bracket is reached. Your code should contain comments. Each class should have a documentation style comment explaining what it is for and each public method should contain a documentation style comment describing any parameters and return values. You should use the **@param** and **@return** keywords in your comments as described in the lectures.