Title : **Project 5A - Lay Your Cards on the Table**
Modules : **09 Simple Class Types & 10 Advanced Classes**
Point Value : **50**
Deadline : **Due before 11pm on Thursday, Nov 17th**

Topics Covered : Writing instantiable classes, using arrays of objects

Introduction

Project 5 has two parts. This is the first of the two. Through the course of Project 5, you will develop programs that work with playing cards and decks of cards. In this part, you define and test a `Card` class and a `Deck` class that encapsulates an array of cards. We then use classes that are similar to these to implement a card game in your final assignment (Project 5B). The development process is broken into six tasks to guide your efforts.

Task 1a

Write a class whose instances represent a single playing card from a standard deck of cards. There are 52 cards in a standard deck. Each card belongs to one of four *suits* and one of 13 *ranks*. The suits are `Spades`, `Hearts`, `Diamonds`, and `Clubs`. The ranks are `Ace`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `Jack`, `Queen`, and `King`.

Your class must be called `Card` and it must have (at least) two attributes (fields): `rank` and `suit`. Define these attributes as integers to encode the ranks and suits:

- use `{1, 2, 3, 4}` to represent `{Clubs, Diamonds, Hearts, Spades}` respectively.
- each of the numerical ranks (`2` through `10`) maps to the corresponding integer;
- map face cards: `{Ace, Jack, Queen, King}` to `{1, 11, 12, 13}`, respectively.

The `Card` attributes must be **private**, so they cannot be accessed from other classes. You must provide *getters* to allow other classes to find out the `rank` (`int getRank()`) and `suit` (`int getSuit()`) values.

The `Card` class must be **immutable**, that is, one must not be able to transform one card into another. So do *not* define any public *setter* methods.

Moreover, the attributes must be set when you create an instance of a `Card`. Therefore, you must provide only one constructor with two input parameters to set `rank` and `suit`, in that order. If either of the input parameters to your constructor is out of range, set both the rank and suit attributes to 0 to indicate an invalid card. **Remember to write test cases for this behavior!**

Provide the following instance methods as well.

- A `toString` method to display `Card` objects in a way that humans can read easily. Hint: you will need to map the integer codes to words.

```
@Override
public String toString()
```

- An `equals` method: two `Cards` are equal if they have both the same `suit` and `rank`. Note that the parameter's type is Object for reasons we'll see later. For now, inside the method, you'll need to immediately use typecasting to convert it to Card type.

```
@Override
public boolean equals(Object other)
```

The methods toString and equals are part of the Object class. Refer to the related section in zyBook and Oracle's Java Object class specification for a more detailed description.

Task 1b

Recall you have learned about using the `compareTo` method for String comparison. In this task, you are asked to implement a similar method for the `Card` class.

- Implement a `compareTo` method to make `Cards` comparable

```
/**
 * Compare this Card with the specified otherCard for order.
 * @param otherCard the other Card object to be compared.
 * @return a negative integer, zero, or a positive integer as
 * this object is less than, equal to, or greater than the otherCard.
 */
public int compareTo(Card otherCard) {
    // TODO implement me!
}
```

We have to decide which is more important: `rank` or `suit`. The choice is arbitrary, and it might be different for different games. But when you buy a new deck of cards, it comes sorted with all the Clubs together, followed by all the Diamonds, and so on. So for now, let's say that `suit` is more important, meaning that any Club is less than any Diamond, and so on.

Task 2

Do not forget the importance of testing! Create a class `TestCard` that you will submit for grading. Use this class to thoroughly test all of the public methods of the `Card` class (think of what tests you may want to write). Below is a small example of test code. It presumes that your test program has the same `assertEquals` method that we used in the `TestTime` coding exercises in the module 09 classwork. You are strongly encouraged to copy that code and consider adding similar methods for other data type comparisons as needed.

```
Card kingofClubs = new Card(13, 1);
System.out.println(kingOfClubs);     // must print "King of Clubs"
assertEquals(13, kingOfClubs.getRank(), "Rank is correctly assigned.");
```

Task 3a

Write a class whose instances represent a deck of cards. The `Deck` class encapsulates an array of `Cards` [Note: you must use an array in the implementation of your `Deck` class, not an `ArrayList` or any other Java collection type.] You must provide an appropriate constructor to create an array and instantiate the 52 cards in a standard deck. A good next step is to write a `toString` method to print out all 52 cards in the deck. Be sure to use the `Card` class `toString` in it! Here is some starter sample code to test the `Deck` class.

NOTE: you are not required to submit a `TestDeck.java` file but should certainly be creating and using one in your development process. You ARE required to submit a `TestCard.java` file (Task 2 above).

```
/** Unit tests for Deck class. */
public class TestDeck {

    /**
     * Execution starts here.
     * @param args command-line arguments.
     */
    public static void main(String[] args) {
        Deck deck = new Deck();
        System.out.println(deck);
    }
}
```

The output must be:

```
Ace of Clubs
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
7 of Clubs
8 of Clubs
9 of Clubs
10 of Clubs
Jack of Clubs
Queen of Clubs
King of Clubs
Ace of Diamonds
```

```
2 of Diamonds
3 of Diamonds
4 of Diamonds
5 of Diamonds
6 of Diamonds
7 of Diamonds
8 of Diamonds
9 of Diamonds
10 of Diamonds
Jack of Diamonds
Queen of Diamonds
King of Diamonds
Ace of Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Spades
2 of Spades
3 of Spades
4 of Spades
5 of Spades
6 of Spades
7 of Spades
8 of Spades
9 of Spades
10 of Spades
Jack of Spades
Queen of Spades
King of Spades
```

Task 3b

For most card games you need to be able to shuffle the deck; that is, put the cards in random order. Create an instance method `shuffle` for the `Deck` class that implements exactly the following algorithm to shuffle the cards. This algorithm represents picking a random card from a resizable collection of cards (the over-sized array named “copy”) and putting it into the “cards” array.

```
// create a copy of the cards array
// let k = 51
// for i in [0, 51] (inclusive)
//    generate a random index j in [0, k] (inclusive)
//    cards[i] = copy[j]
//    copy[j] = copy[k]
//    k--
```

NOTE: Java has built-in utilities to copy and shuffle collections. You are **not** permitted to use them.

NOTE: You must implement the *algorithm* exactly as above, but feel free to use more helpful index variable names.

You must be able to run the following code in a main (test or client) program. The output must display all the 52 playing cards in random order.

```
Deck deck = new Deck();
deck.shuffle();
System.out.println(deck);
```

Task 3c

Now that we have shuffled the deck, we need a way to put it back in order. Create an instance method `sort` for the `Deck` class using exactly the following algorithm. The idea is to traverse the array of cards from back to front repeatedly and at each iteration move the i^{th} "smallest" card (among the remaining cards) and down to position "i".

```
// for each index j from 0 to 51
//    c gets cards[j]
//    i gets j - 1
//    repeat as long as i >= 0 and cards[i] > c
//       cards[i + 1] gets cards[i]
//       decrement i
//    end repeat
//    cards[i + 1] gets c
// end for
```

NOTE: Java has built-in utilities to sort collections. You are **not** permitted to use them.

- You must use the `compareTo` method of the `Card` class for comparing two `Card` objects.

You must be able to run the following code in a main (test or client) program. The output must

display all the 52 playing cards in order.

```
Deck deck = new Deck();
deck.shuffle();
deck.sort();
System.out.println(deck);
```

Submission & Grading

- You must submit three required java files:
  - `[21 points] Card.java`
  - `[8 points] TestCard.java`
  - `[21 points] Deck.java`
- Do not submit any `.class` files.
- All code submitted should be well-documented and 100% Checkstyle-compliant.
- You will receive 0 points for a task in which your submitted code does not compile.