COMP1006/1406 – Summer 2016

Submit a single file called `assignment3.zip` to cuLearn.
There are 105 marks possible marks. The assignment it out of 100.

Note: In this assignment, do NOT use any Java classes other than `Arrays`, `Math`, `Random`, or `String`. In particular, you should not be using `ArrayList` to store your cards. You should be using an array.

1: Written [10 marks]

Write your solutions to both questions (A and B) in a PDF file called `Problem1.pdf`. Do NOT submit a .doc, .docx, .rtf or any other kind of file. Note: You can easily generate a PDF from microsoft word.

A) Explain the similarities and differences between constructors and methods. Give a high level description first and then give particular details as they pertain to the Java language.

B) For each of the following situations, explain whether an array would be a necessary part of an efficient solution or not. Be sure to **concisely** explain why (or why not).

(i) Reading a large sequence of numbers entered by a user and, when the user is finished, reporting the minimum number entered.

(ii) Reading a large sequence of numbers entered by a user and, when the user if finished, reporting the average of all the numbers entered.

(ii) Reading a large sequence of numbers entered by a user and, when the user if finished, reporting the standard deviation of all the numbers entered.

(iv) Reading a large sequence of numbers entered by a user and, when the user if finished, reporting the numbers back in reverse order.

Include your `Problem1.pdf` file in your `assignment3.zip`.

Crazy Eights

The remainder of this assignment consists of problems that culminate in a working game of crazy eights. (Basic game `https://en.wikipedia.org/wiki/Crazy_Eights`)

Our version of the game is different from the wiki page and is specified as follows. The terminology introduced here will be used throughout the assignment.

The game consists of two to seven players and a deck of cards. The deck is initially shuffled and eight cards are dealt from the top of the deck to each player (one card at a time to each player). The next card in the deck is put on the `top` of the discard `pile`.

Each player then takes a turn. A turn consists of taking zero or more cards from the deck (adding to the player's hand) and then playing a single valid card to be placed as the new top card on the pile.

For a given top card on the pile, a valid card is any card with either the same suit, the same rank, an 8 (of any suit) or a joker card.

There are also several additions to the game that we will consider.

- When a 2 (of any suit) is played the next player will be given two cards from the deck and their turn is then skipped to the next player. That is, the next player picks up two cards but does not discard any card.
- When a 4 (of any suit) is played the next player misses their turn.
- When an 8 (of any suit) is played that card's suit is changed to whichever suit the player wants. An 8 of that chosen suit will be placed on the top of the discard pile and that player's turn ends.
- When a joker is played the card is changed into the same card as the current top of the discard pile.

All of this special behaviour for these cards will be handled by the `Game` class that directs the actual game play.

The game ends when a player discards all of the cards in their hand. The game also ends if at any point the deck of cards is completely exhausted (all used). In this case the winner of the game is the player whose cards (in their hand) has the lowest sum of ranks. As soon as the deck empties, the player that took the last card is allowed to make one last play if they can and then the game ends.

2: ♠♥♣♦ Cards ♦♣♥♠ [20 marks]

A **standard** deck of playing cards consists of 52 cards. Each card has a rank (2, 3, . . . , 9, 10, Jack, Queen, King, or Ace) and a suit (spades ♠, hearts ♥, clubs ♣, or diamonds ♦).

In our game of crazy eights, the deck will consist of one or two standard decks in addition to zero or more joker cards.

The ordering of the cards is first specified by the suit and then by rank if the suits are the same. The suits and ranks are ordered as follows:

suits: The suits will be ordered

$$\text{diamonds } ♦ < \text{clubs } ♣ < \text{hearts } ♥ < \text{spades } ♠$$

ranks: The ranks will be ordered

$$2 < 3 < \cdots < 9 < 10 < \text{Jack} < \text{Queen} < \text{King} < \text{Ace}$$

A joker card is greater than any other card in the deck (any two jokers are equal to each other). A joker has no suit ("None" from Card.RANKS). Internally, a joker will have rank 1 or as a string `Card.RANKS[1]`. Again, the overall ordering for non-joker cards is specified by suit first and then rank; for example, all club cards are "less than" all heart cards. Two cards with the same rank and suit are considered equal.

➡ Write a Java class called `MyCard` that `extends` the provided `Card` class. Your class must have two constructors:

```
public MyCard(String rank, String suit)
  // purpose: creates a card with given rank and suit
  // preconditions: suit must be a string found in Cards.SUITS
  //                rank must be a string found in Cards.RANKS

public MyCard(int rank, String suit)
  // purpose: creates a card with the given rank and suit
  // preconditions: suit must be a string found in Cards.SUITS
  //                rank is an integer satisfying 1 <= rank <= 14, where
  //                    1 for joker, 2 for 2, 3 for 3, .., 10 for 10
  //                   11 for jack, 12 for queen, 13 for king, 14 for ace
```

Note that the case of strings is important here. The input strings must be exactly the same as those found in `Card.SUITS` or `Card.RANKS`.

The specification for the three `abstract` methods in the `Card` class are given by:

```
public int getRank()
  // Purpose: Get the current card's rank as an integer
  // Output: the rank of the card
  //           joker -> 1, 2 -> 2, 3 -> 3, ..., 10 -> 10
  //           jack -> 11, queen -> 12, king -> 13, ace -> 14
```

```
public String getRankString()
  // Purpose: Get the current card's rank as a string
  // Returns the cards's rank as one of the strings in Card.RANKS
  //         (whichever corresponds to the card)

public String getSuit()
  // Purpose: Get the current card's suit
  // Returns the card's suit as one of the strings in Card.SUITS
  //         (whichever corresponds to the card)
```

Do not change the abstract `Card` class. Do not add any other public attributes, methods or constructors to the class. You may add any protected attributes/methods as needed.

Mark breakdown: 20 marks for correctness

Put your `MyCard.java` file in your `assignment3.zip` file.

Example:

```
MyCard c = new MyCard("Queen", "Diamonds");
c.getRank(); --returns--> 12
c.getRankString(); --returns--> "Queen"
c.getSuit(); --returns--> "Diamonds"
System.out.println(c); --displays--> 12D
Card d = new MyCard("4", "Spades");
c.compareTo(d); --evaluates to--> some negative int
d.compareTo(c); --evaluates to--> some positive int
MyCard e = new MyCard("Jack", "Spades");
d.compareTo(e); --evaluates to--> some negative int
e.compareTo(e); --evaluates to--> 0
e.getRank(); --evaluates to--> 11
e.getSuit(); --evaluates to--> "Spades"
Card j = new MyCard(1, "None");
System.out.println(j); --displays--> "J"
j.getRankString(); --returns--> "Joker"
j.getRank(); --returns--> 1
j.getSuit(); --returns--> "None"
e.compareTo(j); --evaluates to--> some negative integer
```

3: Deck [20 marks]

In this problem you will complete a class that models a deck of cards to be used in a card game. The deck may consist of zero or more standard decks (52 cards as described in the Cards problem) and zero or more jokers.

➡ Complete the provided `Deck` class. There are two constructors and six methods to complete. The specifications are provided in the skeleton `Deck.java` file.

Some extra comments:

a) A deck's initial ordering of cards is not important. Just be sure that all the cards are present.

b) The `numCards` method returns the number of cards left in the deck when called. The `pop` method removes cards from the deck so the number of cards in the deck will change when this method is called.

c) The `getCards` method returns an array with all the cards in the deck. The front of the array (small index values) is the bottom of the deck and the back of the array (large index values) is the top of the deck.

When the deck has been sorted, the low valued cards should appear in the front of the array (bottom of the deck) and high valued cards should be at the back of the array (top of the deck). For example, for a full standard deck, after sorting, the card in index 0 of the array should be the 2 of Diamonds and the card in the largest index should be the Ace of Spades.

Mark breakdown: 20 marks for correctness

Put your completed `Deck.java` file in your `assignment3.zip` file.

Example:

```
Deck standard = new Deck(1,0);
System.out.println(standard.numCards());   displays → 52
standard.sortDeck();
Card c = standard.pop();   → c is now the Ace of Spades
System.out.println(standard.numCards());   displays → 51
```

4: Hand [20 marks]

The Hand class models a players hand in the game. (In fact, it also represents the player.) A hand will consist of the cards that the player has and actions that the player can take during their turn in the game. The class must have the following single constructor that initializes a hand with the given input cards.

```
public Hand(Card[] cards)
   // initialize hand with the given input cards
```

The Hand class will implement the HandActions interface. Some of the methods have behaviour that is specific to our game of crazy eights (other are more general). Methods that are specific to our game have the additional requirements:

```
Card playCard(Card[] pile)
   // purpose: to play a card in the game
   // input: pile is the top card of the discard pile (a single card)
   // output: returns null if the player either needs to take a card from the deck
   //         or chooses to take a card from the deck.
   //         otherwise, the player has a (valid) card in their hand
   //         to play on the top card and returns this card.
   // side effects: when the method returns a card that card is removed from their hand.

Card[] getCard(Card[] cards)
   // add the cards in the input to the hand
   // always returns null for our game

String message(String question)
   // There is only question that will be asked in our game of crazy eights
   // (Q: what suit do you want the 8 you just played to be?)
   // Any non-null input string will be considered to ask this question.
   //
   // Purpose: After playing an 8 (of any suit) this method, when called,
   // will return one of the four valid suits in Card.SUITS, to define
   // what suit the played 8 should be.
   //
   // You will build logic to decide which suit is best
```

A "valid" move is defined earlier in the assignment. The other methods are specified in the provided HandActions interface.

➡ Complete the provided Hand.java class that implements the provided HandActions interface as specified above.

Note: First implement this so that a player always plays a valid card if they can. If you wish to make your player more clever and sometimes take a card instead of playing a valid card then change the simpleLogic variable in the provided skeleton class to be false. Testing will rely on this.

Mark breakdown: 20 marks for correctness

Put your `Hand.java` file in your `assignment3.zip` file.

Example:

```
Hand hand = new Hand( new Card[]{new Card(3, "Spades"), new Card(8, "Hearts")});
hand.playCard(new Card[]{new Card(7, "Diamonds")}); returns -> the card 8 of Hearts
hand.message("any string"); returns -> likely return "Spades" since the hand has a spade in it
hand.playCard(new Card[]{new MyCard(2, "Spades")}); returns -> the card 3 of Spades
hand.playCard(new Card[]{new MyCard(6, "Spades")}); returns -> null
hand.getCard(new Card[]{ new MyCard(3, "Clubs")}); returns -> null
hand.playCard(new Card[]{new MyCard(3, "Spades")}); returns -> the card 3 of Clubs
```

5: Game [30 marks]

Implement a program called `Game` that plays a game of crazy eights.

The game should take a single command line parameter that specifies the number of players in the game (expecting 2, 3, 4, 5, 6 or 7). If any other argument is given the program ends with a message of how to run the program correctly.

The game will use a deck with a single standard deck plus two jokers if the game has five or fewer players. For 6 or 7 players, the deck will use two standard decks along with four jokers.

While the game is playing, useful information should be displayed. For example, each players hand should be (concisely) displayed before and after each turn showing cards picked up and played so that the logic of the game is easily followed. Messages, such as "player 2 must pick up two cards" or "player 3 misses their turn", when a 2 or 4 is played, respectively, should be displayed.

The game must handle all special cards played (2's, 4's, 8's and jokers) and continuously check if the game has ended. When a player pays an 8, the game should then use the message method to ask which suit the player want the top of the pile to be. After this, instead of putting the discarded 8 on the top pile, the game will create a new 8 card with the chosen suit so that the next player can see the correct suit. When a player needs to pick up a card, the game will call their getCard method with the next card available from the deck.

The game can assume that no player (hand) cheats. Note that there are no human players in the game. All players are computer controlled.

Bonus: For up to 5 bonus marks, create a Hand subclass called `SmartHand` (to be included in your submission) that uses different logic for the game play and have your players play against each other in the game. Always ensure that at least one of each kind of player is playing any game.

Mark breakdown: 10 marks for Style/Desig/Logic and 20 marks for Correctness

Put your completed `Game.java` file in your `assignment3.zip` file.
If you do the bonus, also submit your `SmartHand.java` file.

Submission Recap

A complete assignment will consist of a single file (`assignment3.zip`) with the five files `Problem1.pdf`, `MyCard.java`, `Deck.java`, `Hand.java` and `Game.java`. If you complete the bonus problem then also include `SmartHand.java`.