COMP1006/1406 – Fall 2017

Submit a single file called `assignment3.zip` to cuLearn.
The assignment is out of 50 marks.

1: ♠♥♣♦ Cards ♦♣♥♠ [30 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 ♦).

You will create a class called `Card` that will simulate cards from a standard deck of cards. Your class will `extend` the `AbstractCard` class (provided).

The ordering of the cards in a standard deck (as defined for this assignment) 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

diamonds ♦ < clubs ♣ < hearts ♥ < spades ♠

ranks: The ranks will be ordered

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

A **Joker** card is a special card that is "greater" than any other card in the deck (any two jokers are equal to each other). A joker has no suit ("None" from `AbstractCard.SUITS`).

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 `Card` that `extends` the provided `Card` class. Your class must have two constructors:

```
public Card(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
  // Note: If the rank is AbstractCard.RANKS[15] then any
  //       valid suit from AbstractCard.SUITS can be given
  //       but the card's suit will be set to AbstractCard.SUITS[4]

public Card(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: as with the other constructor, if a joker is created, any valid suit can be passed
  //       but the card's suit will be set to AbstractCard.SUITS[4]
```

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

The specification for the three `abstract` methods in the `AbstractCard` 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 `AbstractCard` class. You can add any (non-static) attributes and helper methods that you need for your `Card` class.

Mark breakdown: 30 marks for correctness

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

Example:

`Card c = new Card("Queen", "Diamonds");`

`c.getRank();` $\xrightarrow{\text{returns}}$ `12`

`c.getRankString();` $\xrightarrow{\text{returns}}$ `"Queen"`

`c.getSuit();` $\xrightarrow{\text{returns}}$ `"Diamonds"`

`System.out.println(c);` $\xrightarrow{\text{displays}}$ `12D`

`Card d = new Card("4", "Spades");`

`c.compareTo(d);` $\xrightarrow{\text{evaluates to}}$ `some negative int`

`d.compareTo(c);` $\xrightarrow{\text{evaluates to}}$ `some positive int`

`Card e = new Card("Jack", "Spades");`

`d.compareTo(e);` $\xrightarrow{\text{evaluates to}}$ `some negative int`

`e.compareTo(e);` $\xrightarrow{\text{evaluates to}}$ `0`

`e.getRank();` $\xrightarrow{\text{evaluates to}}$ `11`

`e.getSuit();` $\xrightarrow{\text{evaluates to}}$ `"Spades"`

`Card j = new Card(1, "None");`

`System.out.println(j);` $\xrightarrow{\text{displays}}$ `"J"`

`j.getRankString();` $\xrightarrow{\text{returns}}$ `"Joker"`

`j.getRank();` $\xrightarrow{\text{returns}}$ `1`

`j.getSuit();` $\xrightarrow{\text{returns}}$ `"None"`

`e.compareTo(j);` $\xrightarrow{\text{evaluates to}}$ `some negative integer`

2: Class/Interface Design [20 marks]

Consider an adventure game in which players live in a two-dimensional grid. Each grid element can be considered to be a room in the world and each room can have things in it (treasure, food, tools, etc) as well as players. There can be different players in the world (user controlled, computer controlled, etc).

Outline the various classes and interfaces (if any) that you would design for this game. Specify which classes are abstract and which are concrete. Provide an API for each class. The API should consist of a list of public methods that the class has. You do not have to mention attributes or private/protected methods.

Draw simple UML class diagrams for the classes/interfaces in your game. That is, draw a box for each class (identifying if it is absbtact or an interface) and show any inheritance (extending a class or implementing an interface) with an arrow.

You must consider the world itself, players in the world, rooms in the world, and several things that can also be in the world that include food, treasure, and tools. For something like food, there should be food that is good for a player and food that is bad (poison). 80% of this problem will be for designing classes (using OOP principles) for the basics described above. The remaining 20% will be allocated for going for going beyond this.

Create a single PDF file called `Adventure.pdf` with your design in it. Your solution to this problem should be typeset (not hand-written and scanned).

Mark breakdown: 20 marks for design

Put your `Adventure.pdf` file in your `assignment3.zip` file.

Submission Recap

A complete assignment will consist of a single file (`assignment3.zip`) with the following two files: `Card.java`, and `Adventure.pdf`.