COMP1406 – Winter 2019

Submit a single file called `assignment4.zip` to the submission server
http://134.117.31.149:9091/

Your zip file must contain a directory called `comp1406a4` and all of your `.java` files must be in this directory. Do not include your `.class` files.

This assignment has 50 marks.

1 ♠♥♣♦ Cards ♦♣♥♠ [25 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 `StandardCard` that will simulate cards from a standard deck of cards. Your class will `extend` the `Card` 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

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

ranks: The ranks will be ordered

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

A **Joker** card is 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. It uses "None" from `Card.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.

The `StandardCard` class must have the following two constructors:

```
public StandardCard(String rank, String suit)
  // purpose: creates a card with given rank and suit
  // preconditions: suit must be a string found in Card.SUITS
  //                rank must be a string found in Card.RANKS
  // Note: If the rank is Card.RANKS[1] then any
  //       valid suit from Card.SUITS can be given
  //       but the card's suit will be set to Card.SUITS[4]

public StandardCard(int rank, String suit)
  // purpose: creates a card with the given rank and suit
  // preconditions: suit must be a string found in Card.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 Card.SUITS[4]
```

The `Card` class has an abstract method called `getRankValue()`. This method should return a numerical value for the rank of the current card. The numerical values for the ranks is given in the specification for the second `StandardCard` constructor above.

You must override the abstract `compareTo()` method inherited from the `Comparable` interface. Use the description of the ordering of standard cards when writing this method.

When testing your class, we will be creating random decks/hands of cards and then sorting them using `java.util.Arrays.sort()`. You should test your class in the same way.

2 Taxes [25 marks]

In this problem, you will create two concrete subclasses of the provided `Taxes` class. These subclasses will compute taxes owed (in the `taxesOwed()` method) in different ways. Note that this is not how actual taxes are determined in Canada. You will create a `StudentTaxes` class and `SeniorTaxes` class. A `BasicTaxes` class is also provided but is not needed.

The `StudentTaxes` class must have the following constructor

```
public StudentTaxes(String name, int sin, double income, double tuition)
```

A student's taxes are determined by the following rules. The first rule that applies (based on the students income and tuition paid) is the rule that is used. Note that taxes owed may be negative.

- taxes owed is 10% of income less 100% of tuition costs if income is less than $10,000.
- taxes owed is 30% of income less 50% of tuition costs if income is less than $30,000.
- taxes owed is 50% of income less 25% of tuition costs.

The `SeniorTaxes` class must have the following constructor

```
public SeniorTaxes(String name, int sin, double income, double health_costs)
```

A senior's taxes are determined by the following rules. The first rule that applies (based on the seniors income and health costs) is the rule that is used. Note that taxes owed may be negative.

- taxes owed is 0% of income less 100% of health costs if income is less than $20,000.
- taxes owed is 10% of income less 75% of health costs if income is less than $50,000.
- taxes owed is 40% of income less 50% of health costs if income is less than $250,000.

Submission Recap

A complete assignment will consist of a single file (`assignment4.zip`) that has a single folder/directory called `comp1406a4`. The `comp1406a4` folder will have the following three files included:

`StandardCard.java`
`StudentTaxes.java`
`SeniorTaxes.java`

All classes must be in the `comp1406a4` package. That is, all files must have the `package comp1406a4;` directive as the first line. Your code will NOT compile if it does not have this and you will receive zero correctness marks if your code does not compile.