TCSS143 Fundamentals of Object-Oriented Programming Theory and Application Programming Assignment #9 20 Points + 5 (Extra Credit)

This assignment will make use of inheritance and polymorphism discussed in class.

Design a class named **BankAccount** to hold the following data for a bank account:

Balance
Number of deposits this month
Number of withdrawals
Annual interest rate
Monthly service charges

The class should have the following methods:

BankAccount(double balance, double rate)	The constructor should accept arguments for the balance and annual interest rate. It should throw an exception if either one of them is less than 0.
void deposit(double amount)	A method that accepts an argument for the amount of the deposit. The method should add the argument to the account balance. It should also increment the variable holding the number of deposits. It should throw an exception if amount is negative.
void withdraw(double amount)	A method that accepts an argument for the amount of the withdrawal. The method should subtract the argument from the balance. It should also increment the variable holding the number of withdrawals. It should throw an exception if amount is negative.
void calcInterest()	A method that updates the balance by calculating the monthly interest earned by the account, and adding this interest to the balance. This is performed by the following formulas: Monthly Interest Rate = (Annual Interest Rate / 1200) Monthly Interest = Balance * Monthly Interest Rate Balance = Balance + Monthly Interest
void monthlyProcess()	A method that subtracts the monthly service charges from the balance, calls the calcInterest method, and then sets the variables that hold the number of withdrawals, number of deposits, and monthly service charges to zero.

Next, design a **SavingsAccount** class that extends the BankAccount class.

The **SavingsAccount** class should have a status field to represent an active or inactive account. If the balance of a savings account falls below $25, it becomes inactive. (The status field could be a boolean variable.) No more withdrawals may be made until the balance is raised above $25, at which time the account becomes active again.

The savings account class should have the following methods:

withdraw	A method that determines whether the account is inactive before a withdrawal is made. (No withdrawal will be allowed if the account is not active.) A withdrawal is then made by calling the superclass version of the method.
deposit	A method that determines whether the account is inactive before a deposit is made. If the account is inactive and the deposit brings the balance above $25, the account becomes active again. The deposit is then made by calling the superclass version of the method.
monthlyProcess	Before the superclass method is called, this method checks the number of withdrawals. If the number of withdrawals for the month is more than 4, a service charge of $1 for each withdrawal above 4 is added to the superclass field that holds the monthly service charges. (Don't forget to check the account balance after the service charge is taken. If the balance falls below $25, the account becomes inactive.)

You may have to create some setters and getters to be able to access some of the superclass's fields. Your test classes should have tests for each method that you write in the corresponding classes.

Challenge (5 points): Define a class `RadioStation` that can be used to store information about radio stations. For each station, you must keep track of its name (a string), its broadcast band (a string) and its station number (a real number). For example, there is a local station called KUOW that is an FM station broadcast on 94.9. Your class must have the following public methods:

`RadioStation(name, band, number)`	constructs a radio station with given name, band and station number
`getName()`	returns the name
`getBand()`	returns the band
`getNumber()`	returns the number
`toString()`	returns a String representation of the radio station
`simulcast(other)`	records the fact that this station and the other are simulcasts

The broadcast band can be any arbitrary string. For example, it might be "AM" versus "FM" or might include subdivisions like "FM 1" and "FM 2" or might include other text like "XM" for satellite radio.

The toString method should return a String composed of the station name followed by the band followed by the station number. For example, if you were to construct the following station:

```
RadioStation station = new RadioStation("KKNW", "AM", 1150.0);
```

then a call on station.toString() should return "KKNW AM 1150.0".

The simulcast method is used to record a relationship between two stations that are broadcasting the same material, as in:

```
RadioStation station1 = new RadioStation("KUOW", "FM", 94.9);
RadioStation station2 = new RadioStation("KUOW", "AM", 1340);
station1.simulcast(station2);
```

The call on simulcast indicates a link between the two stations. The link goes in both directions, so it shouldn't matter whether you make the call above or if you instead make the following call:

```
station2.simulcast(station1);
```

A given station will have at most one simulcast relationship. When such a relationship exists, it should be included in the result of toString. Given the calls above, the call station1.toString() should return:

```
KUOW FM 94.9 (simulcast on AM 1340.0)
```

and the call station2.toString() should return:

```
KUOW AM 1340.0 (simulcast on FM 94.9)
```

Notice that the simulcast notation appears in parentheses and that we include just the band and station number. You are to exactly reproduce this format.

Your class should implement the Comparable<E> interface. Radio stations should be grouped together by band (e.g., AM stations grouped together and FM stations grouped together). Within a given band, the stations should be sorted by station number (e.g., FM 94.9 less than FM 96.5).

Submit RadioStation.java, RadioStationTest.java on Canvas.

Submission and Grading:

There will be points taken off for not following the conventions listed in this document regarding submissions, outputs and naming conventions.

You are required to properly indent your code and will lose points if you make significant indentation mistakes. See the coding conventions document on the course web page for an explanation and examples of proper indentation.

Give meaningful names to methods and variables in your code. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

Include a comment at the beginning of your program with basic information and a description of the program **and include a comment at the start of each method**. Your comments should be written in your own words and not taken directly from this document.

You should include a comment at the beginning of your program (for each class) with some basic information and a description of the program, as in:

```
// Menaka Abraham
// 1/28/16
// TCSS 143
// Assignment #8
//
// This program will...
```

Your files **BankAccount.java, BankAccountTest.java, SavingsAccount.java, SavingsAccountTest.java (if attempting extra credit - RadioStation.java, RadioStationTest.java)** must be submitted on the course web page.