CSC 260.002 Programming Assignment #10
Due Date: Tuesday, December 6

This assignment will have you implement a class and extend it into subclasses, and implement a user class to create objects of the various classes. The user class will also use an ArrayList (import java.util). The topmost class is Employee which will have child classes of Technical and Manager and Technical will itself have subclasses of Developer and SysAdmin. In programs 8 and 9, you made your instance data private, here you will make them protected. There will also be a few methods that will be protected (or private) instead of public. The UML notation for the classes is given below. Note that – is used to indicate protected or private. If an item is to be inherited, it should be protected instead of private.

Employee
-first: String
-last: String
-job: String
-years: int
+Employee()
+Employee(first: String, last: String)
+Employee(first: String, last: String, years: int)
+getFirst(): String
+getLast(): String
+getJob(): String
-computeSalary(): int
+toString(): String

The job should be initialized to "unknown" in all constructors. If a name or years is not provided, initialize them to "unknown" and 0 respectively. The computeSalary method is private and is only called from the toString method. To compute the salary use the following logic:

if years worked > 2 then salary is 30000 + (int)((years-2) * .03 * 30000)
otherwise it is 30000

The toString method should return the Employee's first and last names, their job, and their salary as computed by computeSalary.

Technical extends Employee
-problemSolvingAbility: int
+Technical()
+Technical(first: String, last: String)
+Technical(first: String, last: String, years: int)
+Technical(first: String, last: String, years: int, problemSolvingAbility: int)
-computeSalary(): int

Each of the constructors should invoke its parent class constructor passing it whatever information it has (e.g., super(); for the no-arg constructor, super(first, last); for the 2-arg constructor). problemSolvingAbility is set to 0 for the first three constructors. The computeSalary method overrides that of its parent and works the same as in the Employee class except that the base number is 40000 instead of 30000 and instead of .03, use .04. Additionally, the Technical employee gets a bonus as follows.

If problemSolvingAbility is 1-2, bonus is 7500
If problemSolvingAbility is 3-4, bonus is 10000
If problemSolvingAbility is 5, bonus is 15000
If problemSolvingAbility is 6-7, bonus is 18000
If problemSolvingAbility is 8-9, bonus is 20000
If problemSolvingAbility is higher, bonus is 25000

Technical employees who have worked more than 2 years get a full bonus, those who have not get a ¼ bonus. Remember that this method returns an int.

Developer extends Technical
-numLanguages: int
+Developer()
+Developer(first: String, last: String)
+Developer(first: String, last: String, years: int)
+Developer(first: String, last: String, years: int, problemSolvingAbility: int)
+Developer(first: String, last: String, years: int, problemSolvingAbility: int, numLanguages: int)
+learnsNewLanguages(num: int): void
-computeSalary(): int
+toString(): String

learnsNewLanguages receives an int of the number of languages that this Developer has learned since the last update. Test to make sure the value > 0 before adding it to numLanguages. Both computeSalary and toString override those of the parent class. For computeSalary, call the parent class' computeSalary and return that value + 500 for each language known. For toString, return the value of the parent class's toString concatenated with "\n Number of languages " + numLanguages.

SysAdmin extends Technical
-knowsLinux: boolean
+SysAdmin()
+SysAdmin(first: String, last: String)
+SysAdmin(first: String, last: String, years: int)
+SysAdmin(first: String, last: String, problemSolvingAbility: int)
+SysAdmin(first: String, last: String, years: int, problemSolvingAbility: int)
+SysAdmin(first: String, last: String, years: int, problemSolvingAbility: int, knowsLinux: boolean)
+SysAdmin(first: String, last: String, problemSolvingAbility: int, knowsLinux: boolean)
+learnsLinux(): void
-computeSalary(): int

The method learnsLinux tests to see if knowsLinux is false and if so, changes it to true. The overridden computeSalary method obtains the value of Technical's computeSalary method and then multiplies it by 1.1 if this SysAdmin knows Linux (remember that this must be returned as an int).

Manager extends Employee
-highestDegree: String
+Manager()
+Manager(first: String, last: String)
+ Manager(first: String, last: String, years: int)
+ Manager(first: String, last: String, years: int, highestDegree: String)
-computeSalary(): int
+setHighestDegree(newDegree: String): void
-isBetterThan(degree: String): boolean
+toString(): String

This class extends the Employee class, not the Technical class. It adds the highest degree earned by this Manager, which will be one of "BA", "BS", "MA", "MS" or "MBA". Other degrees are possible but will be assumed to be worse than any of these. As with the other constructors, use super() or super(…) in these constructors so that you can use as much from the parent class as possible. For instance, the last constructor would do super(first, last, years); and then assign highestDegree based on the last parameter passed. The setHighestDegree method receives a new String and assigns highestDegree to this new degree if it is a better degree. In order to implement "better", implement a private (or protected) method called isBetterThan.

isBetterThan compares the new degree to the current highestDegree and returns true if the new degree is better, false otherwise. The order of the degrees is:

Anything not listed above < BA < BS < MA < MS< MBA

Override Employee's computeSalary and instead return the following:

A + (int)(years * B * A)

Where A and B are as follows

BA: 35000, .03
BS: 42000, .03
MA: 45000, .04
MS: 45000, .05
MBA: 50000, .07

If the degree is any other type, use 35000 + (int)(years * .02 * 30000).

The toString method overrides its parent class and returns what the parent class will return (e.g., return super.toString() + … plus on a new line the highest degree earned.

Once you have written and compiled these classes, implement a user class. This class will be short and work as follows.

1. import java.util.*;
2. Declare an ArrayList<Employee> variable as in
   ArrayList<Employee> employees = new ArrayList<Employee>();
3. Add to the ArrayList several new objects of various types of Employee classes (see below). To add to the ArrayList, use employees.add(…);
4. Use an iterator for loop to iterate through the ArrayList and print each object out using System.out.println.
5. Use an iterator for loop to iterate through the ArrayList and do the following
   a. if the object is a Developer, add to the Developer a random number of new languages between 0 and 5.
   b. if the object is a SysAdmin, change it so that the SysAdmin now knows Linux.
   c. if the object is a Manager, use setHighestDegree to "MS". Note that if this Manager has a higher degree, your setHighestDegree should prevent the degree from changing.
6. Use an iterator for loop to iterate through the ArrayList and print each object out using System.out.println.

The iterator for loop might look like this:

```
for(Employee e: employees)
    …
```

In order to test if a given object is a particular class such as Developer, use instance of as in

```
if(e instanceof Developer)
    …
```

In order to execute a method specific to a class (such as setHighestDegree), you have to cast the object. That is, since e would be an Employee, the compiler would provide an error if you tried to do e.setHighestDegree("MS"); So instead, do ((Manager)e).setHighestDegree("MS");

What follows are the initial Employee objects to create.

Technical, George, Duke, 15
Employee, Mike, Keneally
Developer, Gail, Zappa, 6, 5, 4
Manager, Thana, Harris, 3, BA

SysAdmin, Ruth, Underwood, 4, 10, false
Manager, Frank, Zappa, 16, MS
Technical, Ed, Mann, 1, 2
SysAdmin, Ian, Underwood, 4, 3, true
Manager, Lainey, Schooltree, 8, MBA

The following is the output that I received when running this program.

Before changes:

```
George Duke,   job: Technical
  Salary:  $60800

Mike Keneally,   job: unknown
  Salary:  $30000

Gail Zappa,   job: Developer
  Salary:  $63400
  Number of languages known: 4

Thana Harris,   job: Manager
  Salary:  $38150
  Degree: BA

Ruth Underwood,   job: System
administrator
  Salary:  $68200

Frank Zappa,   job: Manager
  Salary:  $81000
  Degree: MS

Ed Mann,   job: Technical
  Salary:  $41875

Ian Underwood,   job: System
administrator
  Salary:  $58520

Lainey Schooltree,   job:
Manager
  Salary:  $78000
  Degree: MBA
```

After changes:

```
George Duke,   job: Technical
  Salary:  $60800

Mike Keneally,   job: unknown
  Salary:  $30000

Gail Zappa,   job: Developer
  Salary:  $63900
  Number of languages known: 5

Thana Harris,   job: Manager
  Salary:  $51750
  Degree: MS

Ruth Underwood,   job: System
administrator
  Salary:  $75020

Frank Zappa,   job: Manager
  Salary:  $81000
  Degree: MS

Ed Mann,   job: Technical
  Salary:  $41875

Ian Underwood,   job: System
administrator
  Salary:  $58520

Lainey Schooltree,   job:
Manager
  Salary:  $78000
  Degree: MBA
```