CSAP - Inheritance Lab

Consider using the existing hierarchy that contains two incomplete classes: the `Person` super class and the `Student` subclass. Create two additional classes, `Teacher` and `CollegeStudent` using inheritance. A `Teacher` "is-a" `Person` but has additional data such as the `subject` and `yearsExperience`. A CollegeStudent "is-a" `Student` but has additional data such as `year` (freshman, sophomore, etc), `major` and `projectedYearOfGraduation`.

```
public class Person {
  private String pName;
  private int pAge;
  private char pGender;
  public Person (    ,     ,     )   {
      pName = n;
      pAge = a;
      pGender = g;
  }
  public String getName() { return pName; }
  public int getAge() { return         }
  public char getGender() { return      }
  public void setName (String n) {pName = n; }
  public void setAge (     ) {pAge = a; }
  public void setGender (char g) {           }
  public String toString () {
      return pName + ", age:" + pAge + ", gender:"+pGender; }
}
public class Student extends Person {
  private String StuID;
  private double StuGPA;
  public Student (String n, int age, char gender, String ID, double gpa){
      super (n, a, gender); //use parent class's constructor
      StuID = ID;
      StuGPA = gpa;
  }
  public String getIDNum() { return StuID;}
  public double getGPA() {                }
  public void setIDNum(String ID) {       }
  public void setGPA(double gpa) {        }
  public String toString() {
     return super.toString() + ", StudentID: "+ StuID + ", GPA: " + StuGPA; }
}
```

Part I: Extending Classes

1. Complete the missing parts necessary to declare the `Person` and `Student` class.

2. Write a `Teacher` class that includes additional data such as `subject` and `yearsExperience`. Write a `CollegeStudent` class that includes `year`, `major` and `projectedYearOfGraduation`.
    a. When writing both the `Teacher` and `CollegeStudent` class, use the `extends` keyword. Use `super` in the constructor to call on the parent constructor that initializes the inherited values.
    b. When writing constructors use `super` to initialize instance variables that are in the parent class
    c. When writing the `toString` method use `super` to do avoid duplicating code already in the parent class

3. Create a tester that constructs all the classes. For example:

```
Person jay = new Person (“Jay”, 35, ‘M’);
System.out.println(jay);

Student amy = new Student (“Amy”, 15, ‘F’, “123321”,3.4);
System.out.println(amy);

Teacher james = new Teacher (“Gosling”, 53, ‘M’, “Computer Science”,35);

CollegeStudent aaron = new CollegeStudent
   (“Aaron”,18, ‘M’, “99099”,3.9,“Senior”, “Computer Engineering”,2016);
```

Sample Output:

```
Jay , age: 35 , gender: M

Amy , age: 15 , gender: F , StudentID: 123321 , GPA: 3.4
```

Part II - Using the Comparable Interface

An interface is a collection of method headings without implementation. Interfaces are used to standardize behavior that is common among different classes. The Comparable Interface has one method compareTo that is used to compare two objects.

In Part II of this Lab, modify the CollegeStudent class so that it implements the Comparable interface. Once a class implements an interface, the class must supply the implementation for all methods of the interface. In Eclipse, when a class implements an interface, an error will occur until all methods of the interface are implemented. HINT: click on the red “X” and choose “Add unimplemented methods”

Comparing CollegeStudent objects is based alphabetical order of pname. The compareTo method should return a negative number, positive number or 0. Since pname is type String, the compareTo method for a CollegeStudent will call String’s compareTo method. Write the compareTo method for CollegeStudent and test it by creating and listing 3 college students in alphabetical order.

Sample Tester code:

```
CollegeStudent c1 = new CollegeStudent (
     (“Aaron”,18, ‘M’, “99099”,3.9,“Freshman”, “Computer Engineering”,2016);

CollegeStudent c2 = new CollegeStudent (
     (“Baron”,19, ‘M’, “19191”,3.9,“Sophomore”, “Computer Engineering”,2016);

CollegeStudent c3 = new CollegeStudent (
     (“Caren”,20, ‘F’, “54321”,3.9,“Junior”, “Computer Engineering”,2016);
```

Part III - Creating an Interface

In Part III of this Lab, create the interface Employable. Employable has only one method isEmployable. An interface can be made in Eclipse by using File-New-Interface and include the one method heading for isEmployable. It should look like the following:

```
public interface Employable {
    public boolean isEmployable();
}
```

Modify the heading of the CollegeStudent class and the Teacher class so that it implements the Employable interface. Once a class implements an interface, the class must supply the implementation for all methods of the interface.

Write the implementation for isEmployable in both the CollegeStudent class and the Teacher class. For the CollegeStudent class, isEmployable will return true if the CollegeStudent is between the ages of 22 and 18, inclusive and has a gpa > 2.5. For a Teacher, isEmployable will return true if age > 22 and yearsExperience > 5.

Part IV - Instantiating Objects with an Interface Reference

Create a helper method that returns the birth year for any `Employable` person based on their age. Since the helper method is called from main, it must be defined as a `static` method. Use the following method heading:

```
private static int getBirthYear (Employable e, int currYr)
```

Test the helper method by instantiating 2 different `Employable` objects and printing out their birth year.

Sample Tester code:

```
 Employable c = new CollegeStudent ( /* provide necessary parameters */ );
 Employable t = new Teacher (/* provide necessary parameters */ );

System.out.println (c.getName() + “, Birth year: “ + getBirthYear(c,2016));
System.out.println (t.getName() + “, Birth year: “ + getBirthYear(t,2016));
```