Programming and Data Structures
Assignment 3: Polymorphism, Abstract Classes, and Interfaces

Objectives of the assignment

Students should demonstrate the following abilities:

1. Create abstract classes that model common behavior between related classes
2. Use interfaces that model common behavior between unrelated classes
3. Derive concrete classes from abstract classes
4. Use polymorphism to manipulate objects of the concrete classes as objects of the abstract class type

Assignment

1. Create an abstract class **`Person`** that implements the interface **`Comparable`** for the type **`Person`**. The class has two attributes name and age of type **`String`** and **`int`** respectively. Create two constructors, getters and setters, and **`toString()`** method.
2. Create an abstract class **`Employee`** that extends class **`Person`**. The class has two additional attributes for the employee ID and hire date (**`String`**). Create two constructors, getters and setters, an abstract method **`getSalary()`** that returns a double, and override **`toString()`** method. Define **`compareTo(Person p)`** as follows:
    a. Returns **0** if the salary of the first employee is equal to the salary of the second employee. Use the abstract method **`getSalary()`** for the value of the salary.
    b. Returns a positive integer if the salary of the first employee is greater than the salary of the second employee.
    c. Returns a negative integer if the salary of the first employee is less than the salary of the second employee.
3. Create a concrete class **`Student`** that extends the class **`Person`**. The class has three additional attributes for the student ID, major, and GPA. Create two constructors,

getters and setters, override **toString()** method and provide a definition for **compareTo(Peron p)** as follows:

   a. Returns 0 if the gpa of the first student is equal to the gpa of the second student.
   b. Returns a positive integer if the gpa of the first student is greater than the gpa of the second student.
   c. Returns a negative integer if the gpa of the first student is less than the gpa of the second student.

4. Create a concrete class **FullTimeEmployee** that extends the class **Employee** and has one additional attribute for the annual salary. Create two constructors, setter, override **toString()**, and provide a definition for **getSalary()** to return the annual salary.
5. Create a concrete class **PartTimeEmployee** that extends the class **Employee** and has two additional attributes for the hourly wage and the number of hours worked by the employee. Create two constructors, getters and setters, override **toString()**, and provide a definition for **getSalary()** to return the salary as the product of the hourly wage by the number of hours.
6. Create a test program named **Test** with a main method and three other methods **sort()**, **printSortedStudents()**, and **printSortedEmployees()** defined as follows:
   a. Method **void sort(Person[] list, int start, int end)** accepts an array of type **Person** and two integers as inputs. The method orders the elements in **list** from index **start** to **(end-1)**. Use any sorting algorithm you want and **compareTo()** to compare the elements in **list**.
   b. Method **printSortedStudents(Person[] list)** accepts an array of type **Person** as input, extracts the elements that are instances of the class **Student** in a separate array of type **Student**, passes the extracted array to method **sort**, and displays the elements of the extracted array after the sorting (students will be sorted based on their gpas).

c. Method **printSortedEmployees(Person[] list)** accepts an array of type **Person** as input, extracts the elements that are instances of class **Employee** in a separate array of type **Employee**, passes the extracted array to method **sort**, and displays the elements of the extracted array after the sorting (employees will be sorted based on their salaries).

d. The main method creates an array of 10 elements of type **Person** named **personList**. Initialize **personList** with the following mix of **Student**, **PartTimeEmployee**, and **FullTimeEmployee** objects as listed below.

```
personList[0] = new Student("Lucy Treston", 20,
                            12345, "CSE", 3.75);
personList [1] = new Student("Mark Brown", 18,
                             12344, "ISE", 3.50);
personList [2] = new FullTimeEmployee("Jerry Zurcker", 25,
                            3333333, "03/10/2017", 500000);
personList [3] = new PartTimeEmployee("Sharon Luft", 22,
                         6666666, "01/01/2010", 32.0, 100);
personList [4] = new Student("Emma Packard", 19, 12355,
                             "CSB", 3.0);
personList [5] = new Student("Felix Hirpara", 22, 55123,
                             "CSE", 2.75);
personList [6] = new PartTimeEmployee("Jade Farrar ", 29,
                          1111111, "07/22/2012", 22.0, 45);
personList [7] = new Student("Junita Stoltzman", 21,
                              44123, "ISE", 2.5);
personList [8] = new PartTimeEmployee("Brian Lin", 31,
                          7777777, "02/01/2014", 35.0, 31);
personList [9] = new FullTimeEmployee("Alicia Bubash", 35,
                            5555555, "08/01/2018", 125000);
```

Display the information of all the objects in **personList** using a for loop and invoking the method **toString()** on each element in the array.

Display the list of students sorted from the lowest to the highest gpa using the method **displaySortedStudents()**.

Display the list employees sorted from the lowest to the highest salary using the method **displaySortedEmployees()**.

7. Submit the following files on courseSite: **`Person.java`**, **`Student.java`**, **`Employee.java`**, **`PartTimeEmployee.java`**, **`FullTimeEmployee.java`**, and **`Test.java`**. Draw the UML diagram that shows the classes **`Person`**, **`Student`**, **`Employee`**, **`FullTimeEmployee`**, and **`PartTimeEmployee`**, the interface **`Comparable<Person>`**,and all the relationships between the interface, abstract classes and concrete classes. Submit the UML diagram on courseSite.
8. Your program should generate the same output as the sample run provided below.

```
List of people:
Name                    Age   ID          Major/Hire Date   GPA/Salary
Lucy Treston            20    12345       CSE               3.75
Mark Brown              18    12344       ISE               3.50
Jerry Zurcker           25    3333333     03/10/2017        $500000.00/year
Sharon Luft             22    6666666     01/01/2010        $3200.00
Emma Packard            19    12355       CSB               3.00
Felix Hirpara           22    55123       CSE               2.75
Jade Farrar             29    1111111     07/22/2012        $990.00
Junita Stoltzman        21    44123       ISE               2.50
Brian Lin               31    7777777     02/01/2016        $1085.00
Alicia Bubash           35    5555555     09/14/2018        $125000.00/year

List of students:
Name                    Age   ID          Major             GPA
Junita Stoltzman        21    44123       ISE               2.50
Felix Hirpara           22    55123       CSE               2.75
Emma Packard            19    12355       CSB               3.00
Mark Brown              18    12344       ISE               3.50
Lucy Treston            20    12345       CSE               3.75

List of employees:
Name                    Age   ID          Hire Date         Salary
Jade Farrar             29    1111111     07/22/2012        $990.00
Brian Lin               31    7777777     02/01/2016        $1085.00
Sharon Luft             22    6666666     01/01/2010        $3200.00
Alicia Bubash           35    5555555     09/14/2018        $125000.00/year
Jerry Zurcker           25    3333333     03/10/2017        $500000.00/year
```