San José State University
Computer Science Department
CS151, Object Oriented Design and Programming, 05, Fall 2022

Homework #1

Objective:

This homework's objective is to review and understand the unit on classes and interfaces, when to use one or another, and various ways to implement them in Java.

Details:

Exercise 1:

Define and implement class **Student**. This class should contain the following fields: first name, last name, age, gpa, major, department. Age should be an integer value. GPA should be a floating point value. This class should contain getters and setters for all its attributes. This class also needs at least one constructor implemented. Within class Student implement a nested non-static inner class called **Course**. This class should declare and implement method *printSchedule()*. This method should print out the student's course schedule to command line (using *System.out.println()* method). The student schedule is just a string and its details are up to you. For example, it can print something like this:

CS151 Tue/Thur 6-7:15
Eng101 Mon/Wed 10-11:15
Hist100 Tue/Thur 1:30-2:45

You do not need to declare the schedule details as attributes in the Course class. You can simply print a hardcoded string in the *printSchedule()* method. Save this class and its definition into a file named **Student.java**.

Define and implement class **StudentTest**. This class should implement *main()* method. In the body of the *main()* method you should create an instance of Student with the following information: John Smith, 20 year old, 3.6 gpa, Computer Science major, School of Computer Science department. You should make an appropriate call and print this student's schedule. Save this class and its definition into a file named **StudentTest.java**.

Exercise 2:

List main differences between primitive data type **int** and class **Integer**. Be as specific as possible. What are advantages of using each one? List all specific functionalities that are available when using one and not the other. Consult Java API as a resource to find out this information. Save your answer to a text file named **exercise2.txt**.

Exercise 3:

Define and implement class **Person**. This class should contain the following fields: first name, last name, age, social security number, address, gender, and weight. Age should be an integer value, while weight should be a floating point value. This class should contain getters and setters for all its attributes. This class also needs at least one constructor implemented. For example, this constructor can accept values for all the attributes as input arguments. This class should also override implementation of *toString()* method and implement new method named *introduce()*, which will display/print this person's attribute values to the command line. Remember that *toString()* method returns a string representation of the object, while *introduce()* method prints the object information and has a return type void. Save this class and its definition into a file named **Person.java**.

Define and implement class **Employee**. An employee is a person, so has all the attributes and functionality of a person. In addition to being a person, an employee should have the following attributes: employee id, employee status (can be contractor, full time, part time), pay amount. The pay amount is meant to reflect the amount of pay an employee gets paid for a unit of time they work. In other words, pay amount is the employee's base pay. For part time employees and contractors pay units are per hour. For full time employees pay units are per year. Therefore, based on the employee status the employee pay units can be determined. E.g. if pay amount field is 40 for a part time employee, then we can say that this employee gets paid $40/hr. On the other hand if pay amount field is 100000 for a full time employee, then we can say that this employee gets paid $100000/yr.

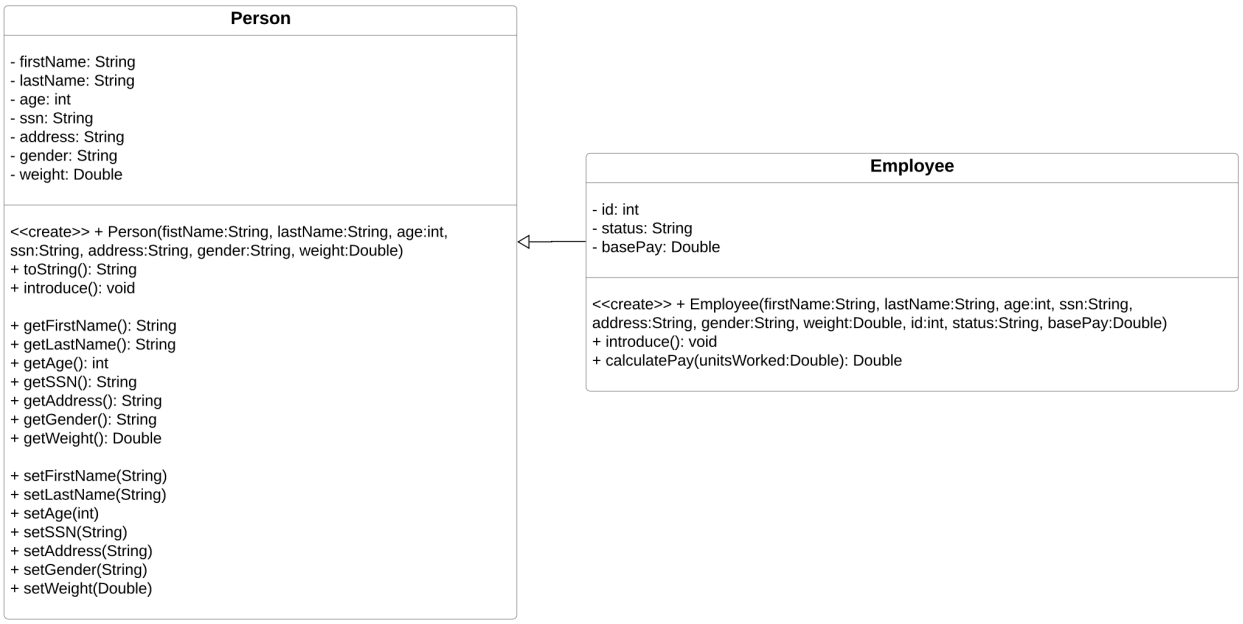
This class should contain getters and setters for all its attributes. In addition to all the functionality of a person, employee should be able to print their own introduction by overriding method *introduce()*. This means they need to be able to display to command line all their information as a person as well as additional employee information (employee id, status, and pay compensation).

Employees should also be able to compute pay owed to them for time they worked. In other words, calculate the amount of their paycheck. E.g. a part time employee makes $20/hr and worked 30 hours. Therefore, the money owed to this employee should be $600, computed as 20*30. As another example, let's say a full time employee gets paid $100,000/yr and worked for 2 weeks. This employee should get paid $3846.15, computed as 100000 * 1/52 * 2 (base pay of 100000 divided by 52 weeks to figure out pay per week, then multiplied by the number of weeks worked). Implement method *calculatePay()*, which accepts the number of units worked (number of hours for part time employees and contractors or the number of weeks for full time employees). This method should check the value of the employee status and use the proper logic to compute the paycheck amount based on the base pay and the number of units of work.

During an Employee object instantiation, the constructor of this class should pass on instantiation of superclass' attributes to the superclass constructor. Employee constructor could, for example, accept values for all the attributes of this class and the superclass, then invoke the superclass

constructor and pass on the appropriate attribute values. Save this class and its definition into a file named **Employee.java**.

Below is a UML class diagram for these two classes and their relationship:



Define and implement class **EmployeeTest**. This class should implement *main()* method. In the body of the *main()* method you should create the following Employee instances, call *introduce()* and *calculatePay()* on each one and display the results of *calculatePay()* method to command line:
Joe Smith, a contractor, pay is $60/hr, should get paid or working 30 hours, other details are up to you
Lisa Gray, a full time employee, pay is $110,000/yr, should get paid or working 2 weeks, other details are up to you
Timothy Briggs, a full time employee, pay is $80,000/yr, should get paid or working 4 weeks, other details are up to you
George Wallace, a part time employee, pay is $20/hr, should get paid or working 25 hours, other details are up to you
Amy Student, a contractor employee, pay is $45/hr, should get paid or working 45 hours, other details are up to you

For visual presentation make sure to include an empty line between each employee instance output. Save this class and its definition into a file named **EmployeeTest.java**.

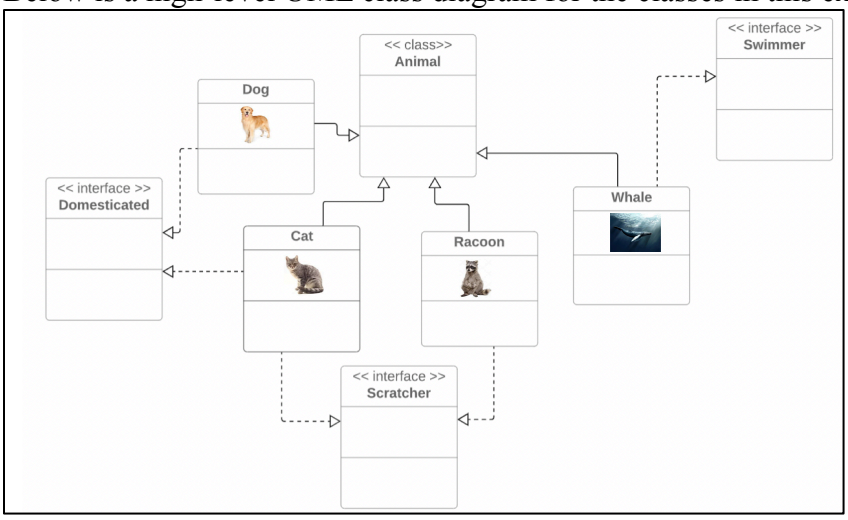
Exercise 4:

Define and implement an immutable class **Product**. This class should contain the following fields: product name, product description, product price, maximum quantity allowed to be ordered. This class should contain getters for all its attributes. This class also needs a fully parameterized constructor. This class should also override Object's implementation of *toString()* method. Save this class and its definition into a file named **Product.java**.

Exercise 5:

This exercise demonstrates how you can create an application with complex class and interface relationships and utilize interfaces to achieve multiple inheritance (remember that in Java you cannot inherit from multiple classes).

Below is a high-level UML class diagram for the classes in this exercise:



Define and implement the following classes **Dog**, **Cat**, **Racoon**, **Whale**. Save the class definitions into files named **Dog.java**, **Cat.java**, **Racoon.java**, **Whale.java**. Each of these classes should define or inherit the following attributes: type of the animal, name of the animal, age, male or female, which environment it lives in (home for dogs and cats, forest for racoons, ocean for whales), and how fast it moves (e.g. x miles/hr). Each of these classes should contain getters and setters for all their attributes. Each of these classes should have the following functionality where the action is simply printed to command line: *move()*, *sound()*, *eat()*, *sleep()*, *toString()*. All classes are a type of animal. Some of the classes are domesticated (for example, you could introduce Domesticated interface to your design). Being domesticated introduces additional functionality: *walk()*, *greetHuman()*. Some of the classes are swimmers, which adds functionality *swim()*. Dog has an additional functionality *bark()*. Both Racoon and Cat can *scratch()*.

Define and implement class **AnimalTest**. This class should implement *main()* method. In the body of the *main()* method you should create one instance of your classes Dog, Cat, Racoon, and Whale. On each instance call each one of the possible functionalities (methods representing those

functionalities). These calls should just output the appropriate action to command line. For example, calling method *walk()* should output something like "Walking". For visual presentation make sure to include an empty line between the outputs for each animal instance. Save this class and its definition into a file named **AnimalTest.java**.

Submission:

Submit all files created by you for the homework exercises: Student.java, StudentTest.java, exercise2.txt, Person.java, Employee.java, EmployeeTest.java, Product.java, Dog.java, Cat.java, Racoon.java, Whale.java, Animal.java, Domesticated.java, Swimmer.java, Scratcher.java, AnimalTest.java and any other files you completed for this assignment, if any.

Make sure to submit by 11:59pm on the due date listed in Canvas. Submit your solution via Canvas.

If you have any questions, message me or the grader or both:
Yulia.Newton@sjsu.edu
madhujitaranjit.ambaskar@sjsu.edu

Grading:

Your code must compile and execute successfully in order to get full credit for this assignment. For each exercise, except exercises # 2, I will compile and execute the files.

- Program with no compile errors
- Program executes
- Program outputs what is required by the exercise

Exercise #2 will be graded based on my reading the answer (5 pts for a correct and complete answer).

A total of 45 points are possible for this homework assignment.