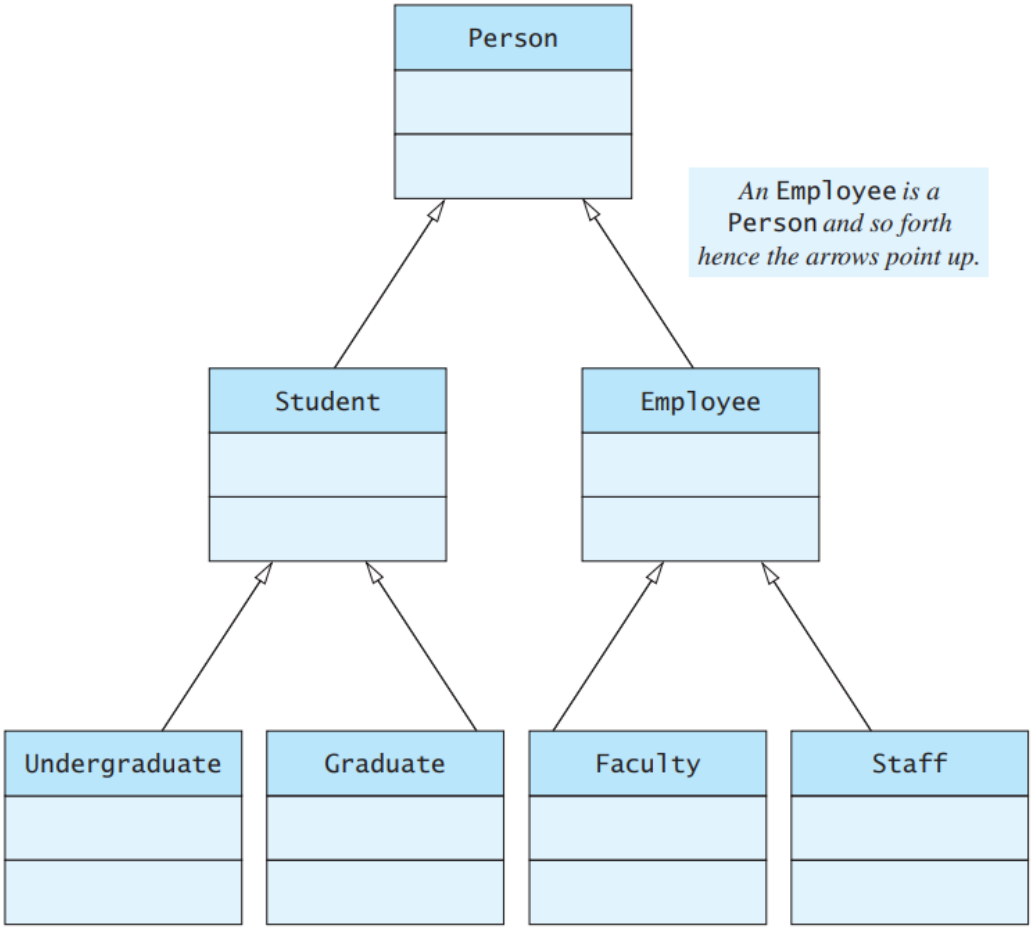
CSE 2102: Introduction to Software Engineering

Lab #5: October 19, 2022

Inheritance & Polymorphism

In this lab, we will learn about how to use Java to implement inheritance and polymorphism. Consider the inheritance hierarchy shown in the following figure (we saw a similar inheritance hierarchy in Lecture #7).



The following code snippet in the file `Person.java` defines the Person class, with just one instance variable name. We can make the following observations from this snippet. The instance variable is declared private. There is only one constructor, which accepts the name of the person as input, and creates an object of the type person with that name. `getName` and `setName` are accessor and mutator methods. The method `writeOutput()` prints the name of the person. The method `hasSameName` returns true if the two persons being compared have the same name.

```
public class Person
{
      private String name;
      public Person(String initialName)
      {
            name = initialName;
      }
      public void setName(String newName)
      {
            name = newName;
      }
      public String getName()
      {
            return name;
      }
      public void writeOutput()
      {
            System.out.println("Name: " + name);
      }
      public boolean hasSameName(Person otherPerson)
      {
            return this.name.equalsIgnoreCase(otherPerson.name);
      }
}
```

Next, we define the class student in the file `Student.java`. The `Student` class is derived from the `Person` class, and hence, it inherits all the public instance variables and methods from the person class. The following observations can be made from the `Student` class. The keyword extends indicates the inheritance relationship between the Student and Person class. The `Student` class adds another private instance variable `studentNumber`. The `Student` class has only one constructor, which takes both the name and student number as input, and creates a student object. This constructor calls the constructor in the `Person` class using `super()`. The `reset()` method is needed to change the name of the student, because the name variable being private cannot be directly accessed. Thus, private variables cannot be accessed, they can only be changed through the public methods provided in the base class. Finally, the `equals` method returns true only if the two students being compared have identical names and student numbers. The `Student` class overrides the methods `writeOutput()` from the `Person` class, and prints both the instance variables. The method `equals` is not overridden, because although it has the same return type, it accepts a different parameter in the `Student` class than `Person` class.

```
public class Student extends Person
{
    private int studentNumber;
    public Student(String initialName, int initialStudentNumber)
    {
        super(initialName);
        studentNumber = initialStudentNumber;
    }
    public void reset(String newName, int newStudentNumber)
    {
        setName(newName);
        studentNumber = newStudentNumber;

    }
    public int getStudentNumber()
    {
        return studentNumber;
    }
    public void setStudentNumber(int newStudentNumber)
    {
        studentNumber = newStudentNumber;
    }
    public void writeOutput()
    {
        System.out.println("Name: " + getName( ));
        System.out.println("Student Number: " + studentNumber);
    }
    public boolean equals(Student otherStudent)
    {
        return this.hasSameName(otherStudent) &&
               (this.studentNumber == otherStudent.studentNumber);
    }
}
```

Finally, the test driver `InheritanceDemo.java`, in its own file, demonstrates how the `Student` and `Person` classes work together. Warren Peace is the student `s1` and has the number 1234. Their name is then changed to Warren Buffet, holding the student number the same. Although the type of object `s2` is `Person`, at runtime, because object `s2` is instantiated from the `Student` class, the `writeOutput()` method from the Student class is called dynamically. `s2` can be substituted as a parameter in the equals method as well. Finally, students `s1` and `s2` are not the same because although they have the same student number they have different names.

```
public class InheritanceDemo
{
    public static void main(String[] args)
    {
        Student s1 = new Student("Warren Peace",1234);
        s1.writeOutput();
        System.out.println();
        s1.reset("Warren Buffet",1234);
        s1.writeOutput();
        System.out.println();
        Person s2 = new Student("Bill Gates",1234);
        s2.writeOutput();
        System.out.println();
        if (s1.equals(s2))
           System.out.println("Two students are the same");
        else
           System.out.println("Two students are not the same");
    }
}
```

`InheritanceDemo.java` produces the following output.

```
Name: Warren Peace
Student Number: 1234

Name: Warren Buffet
Student Number: 1234

Name: Bill Gates
Student Number: 1234

Two students are not the same
```