LAB

Thursday 7/31/2008

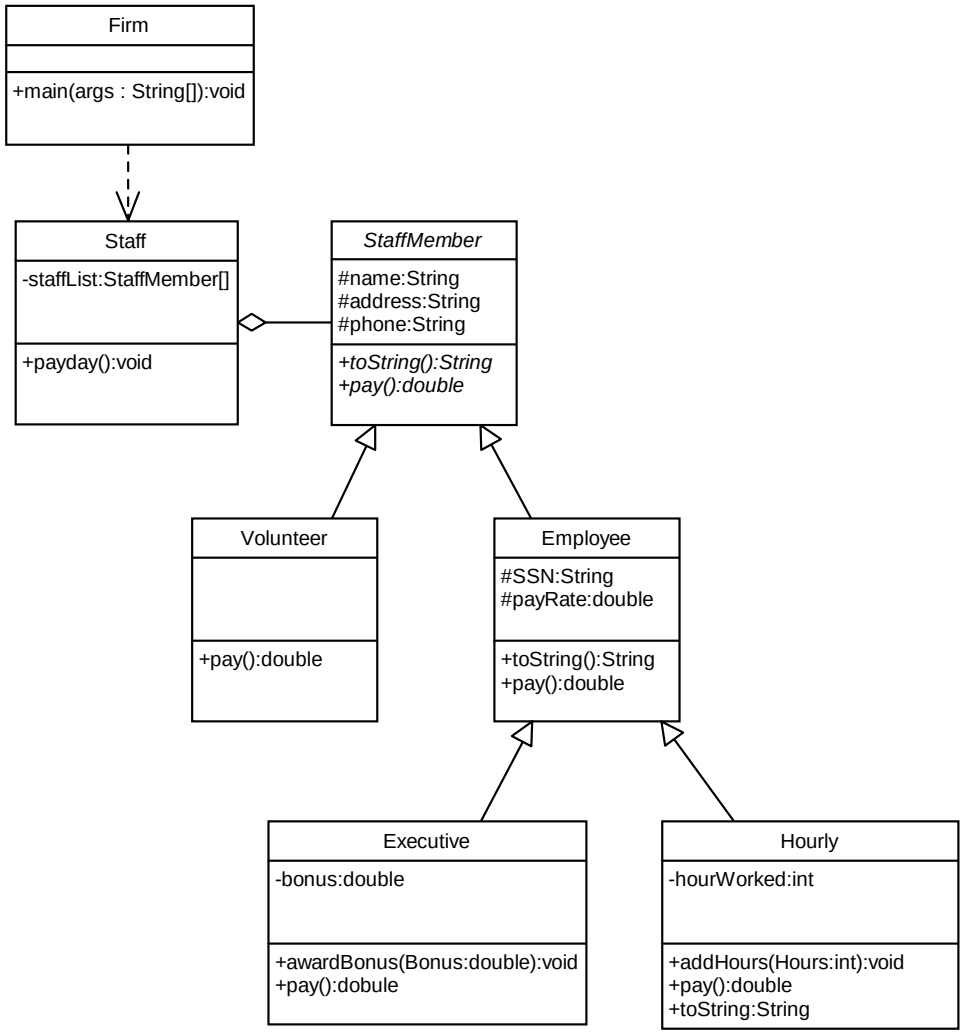
1. Now we use inheritance to perform payroll calculation based on the type of an employee. Consider the following problem:

*A company pays its employees on a weekly basis. The company has four types of employees: salaried employees, who are paid a fixed weekly salary regardless of the number of hours worked; hour employees, who are paid by the hour (for example, wage * hours) and receive overtime pay (for example, (hours – 40) * wage * 1.5) and the maximum hours worked is 168 hours; commission employees, who are paid a percentage of their sales; and salaried-commission employees, who receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their salaries. The company wants to implement a Java application that performs its payroll calculations.*

```
public class Main {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
      // create Employee array
      Employee[] employees = new Employee[4];

      // initialize array with Employess
      employees[0] = new SalariedEmployee("George", "Bush", "111-22-
3333",1000);
      employees[1] = new HourlyEmployee("Bill", "Clinton", "222-33-4444",
16.75, 40);
      employees[2] = new CommissionEmployee("John", "MaCain", "333-44-
5555", 10000.0, 0.06);
      employees[3] = new BasePlusCommissionEmployee("Barack", "Obama",
"444-55-6666", 5000.0, 0.04, 300);

      String output = "";

      // generically process each element in array employees

      for (int i = 0; i < employees.length; i++) {
```

```
            output += employees[i].toString();

            if (employees[i] instanceof BasePlusCommissionEmployee){
                                ......
               output += "\nnew base salary with 10% increase is: $" +
                     currentEmployee.getBaseSalary();
            }

            output += "\nearned: $" + employees[i].earnings() + "\n";
         }

      // get type name of each object in employees array
         for (int j = 0; j < employees.length; j++)
            output += "\nEmployee " + j + " is a " +
   employees[j].getClass().getName();

         System.out.println(output);
      }

   }
```

2. Below is the UML class diagram for another situation. The classes in it represent various types of employees that might be employed at a particular company. The Firm class contains a main driver that creates a `Staff` of employees and invokes the payday method to pay them all. The program output includes information about each employee and how much each is paid.



The `Staff` class maintains an array of objects that represent individual employee of various types. Note that the array is declared to hold `Staffmember` references, but it is actually filled with objects created from several other classes, such as `Executive`, `Employee`, etc. These classes are all descendents of the `StaffMember` class, so the assignments are valid. The `staffList` array is filled with polymorphic reference.

The `payday` method of the `Staff` class scans through the list of employees, printing their information and invoking their pay methods to determine how much each employee should be paid. The invocation of the `pay` method is polymorphic because each class has its own version of the `pay` method.

Note that this time, the executive is awarded $500 for bonus and the hourly employee worked 40 hours.

```
public class Firm {
   public static void main (String[] args){
   Staff personnel = new Staff();

   personnel.payday();
   }
}
```

```
public class Staff {
  private StaffMember[] staffList;

  public Staff(){
    staffList = new StaffMember[4];

    staffList[0] = new Executive ("George", "123 Main Street", "555-5555", "123-
45-6789", 10000.00);
    staffList[1] = new Employee ("Diane", "456 Grand Ave.", "666-6666", "012-
34-5678", 4000.00);
    staffList[2] = new Hourly ("Woody", "789 Fifth Ave.", "777-7777", "234-56-
7890", 10.55);
    staffList[3] = new Volunteer ("Jane", "012 South Street", "888-8888");

    ((Executive)staffList[0]).awardBonus(500.00);
    ((Hourly)staffList[2]).addHours(40);
  }

  public void payday(){
    double amount;

    for (int count = 0; count < staffList.length; count++){

      // get type name of each object staff array

      System.out.println(staffList[count].getClass().getName());
      System.out.println(staffList[count]);
      amount = staffList[count].pay();

      if (amount == 0.0)
        System.out.println("Thanks!");
      else
        System.out.println("Paid: " + amount);

      System.out.println ("-----------------------------------");
    }
  }
}
```

```
public abstract class StaffMember {
   protected String name;
   protected String address;
   protected String phone;

   public StaffMember (String iName, String iAddress, String iPhone){
                  ……
   }

   public String toString(){
                  ……
   }

   public abstract double pay();

}
```

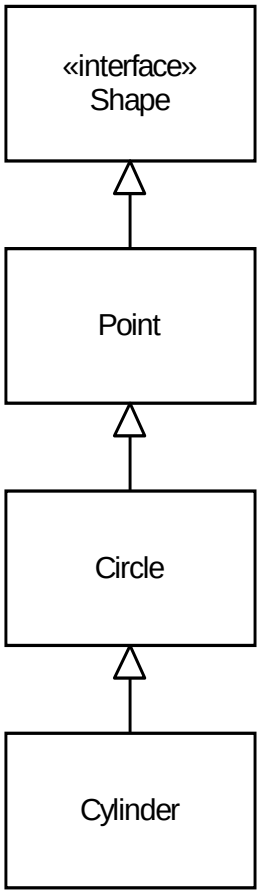
```
public class Employee extends StaffMember{
   protected String SSN;
   protected double payRate;

public Employee(String iName, String iAddress, String iPhone, String iSSN,
double rate){
                        ......
   }

   public String toString(){
                    ......
   }

   public double pay(){
                    ......
   }

}
```

```
public class Volunteer extends StaffMember{
   public Volunteer (String iName, String iAddress, String iPhone){
                        ……
   }

   public double pay(){
                ……
   }
}
```

```
public class Executive extends Employee{
   private double bonus;

   public Executive (String iName, String iAddress, String iPhone,
         String SSN, double rate){
                  ……
   }

   public void awardBonus(double execBonus){
                  ……
   }

   public double pay(){
                  ……
   }
}
```

```
public class Hourly extends Employee{
   private int hoursWorked;

   public Hourly (String iName, String iAddress, String iPhone, String iSSN,
double rate){
                        ……
   }

   public void addHours (int moreHours){
                        ……
   }

   public double pay(){
                        ……
   }

   public String toString(){
                        ……
   }
}
```

3. Create a program that calculates the areas and volumes of circle and cylinder using an interface class and inheritance. Below are the shape hierarchy class diagram and the table shows polymorphic interface for the Shape hierarchy classes.



	getArea	getVolume	getName	toString
Shape	0.0	0.0	abstract	Default Object Implementation
Point	0.0	0.0	"Point"	[x, y]
Circle	πr^2	0.0	"Circle"	center = [x, y] radius = r
Cylinder	$2\pi r^2 + 2\pi rh$	$\pi r^2 h$	"Cylinder"	center = [x, y] radius = r height = h

```
public interface Shape {
	public double getArea();
	public double getVolume();
	public String getName();
}
```

```
public class Point extends Object implements Shape {
        private int x;
        private int y;

        public Point (int xValue, int yValue){
           x = xValue;
           y = yValue;
        }

        public void setX (int xValue){
           x = xValue;
        }

        public int getX(){
           return x;
        }

        public void setY(int yValue){
           y = yValue;
        }

        public int getY(){
           return y;
        }

        public double getArea(){
           return 0.0;
        }

        public double getVolume(){
           return 0.0;
        }

        public String getName(){
           return "Point";
        }

        public String toString(){
           ......
        }
    }
```

```
public class Circle extends Point{
   private double radius;

   public Circle (int x, int y, double radiusValue){
                  ……
   }

   public void setRadius (double radiusValue){
                  ……
   }

   public double getRadius(){
      return radius;
   }

   public double getDiameter(){
                  ……
   }

   public double getCircumference(){
                  ……
   }

   public double getArea(){
                  ……
   }

   public String getName(){
      return "Circle";
   }

   public String toString(){
          return        ……

   }
}
```

```
public class Cylinder extends Circle{
   private double height;

   public Cylinder (int x, int y, double radius, double heightValue){
                  ……
   }

   public void setHeight(double heightValue){
      height = (heightValue < 0.0 ? 0.0 : heightValue);
   }

   public double getHeight(){
      return height;
   }

   public double getArea(){
                  ……
   }

   public double getVolume(){
                  ……
   }

   public String getName(){
      return "Cylinder";
   }

   public String toString(){
                  ……
   }
}
```

```
import java.text.DecimalFormat;

public class ShapeTest {
   public static void main(String[] args){
      DecimalFormat twoDigits = new DecimalFormat("0.00");
      Point point = new Point (7, 11);
      Circle circle = new Circle (22, 8, 3.5);
      Cylinder cylinder = new Cylinder (20, 30, 3.3, 10.75);

      String output = point.getName() + ": " + point + "\n" +
            circle.getName() + ": " + circle + "\n" +
            cylinder.getName() + ": " + cylinder + "\n";

      Shape[] arrayOfShapes = new Shape [3];

      arrayOfShapes[0] = point;
      arrayOfShapes[1] = circle;
      arrayOfShapes[2] = cylinder;

      for (int i = 0; i < arrayOfShapes.length; i++){
         output += "\n\n" + arrayOfShapes[i].getName() + ": " +
               arrayOfShapes[i].toString() + "\nArea = " +
               twoDigits.format(arrayOfShapes[i].getArea()) + "\nVolume = " +
               twoDigits.format(arrayOfShapes[i].getVolume());
      }

      System.out.println(output);
   }

}
```