CSIT121

Object Oriented Design and Programming

Lesson 4

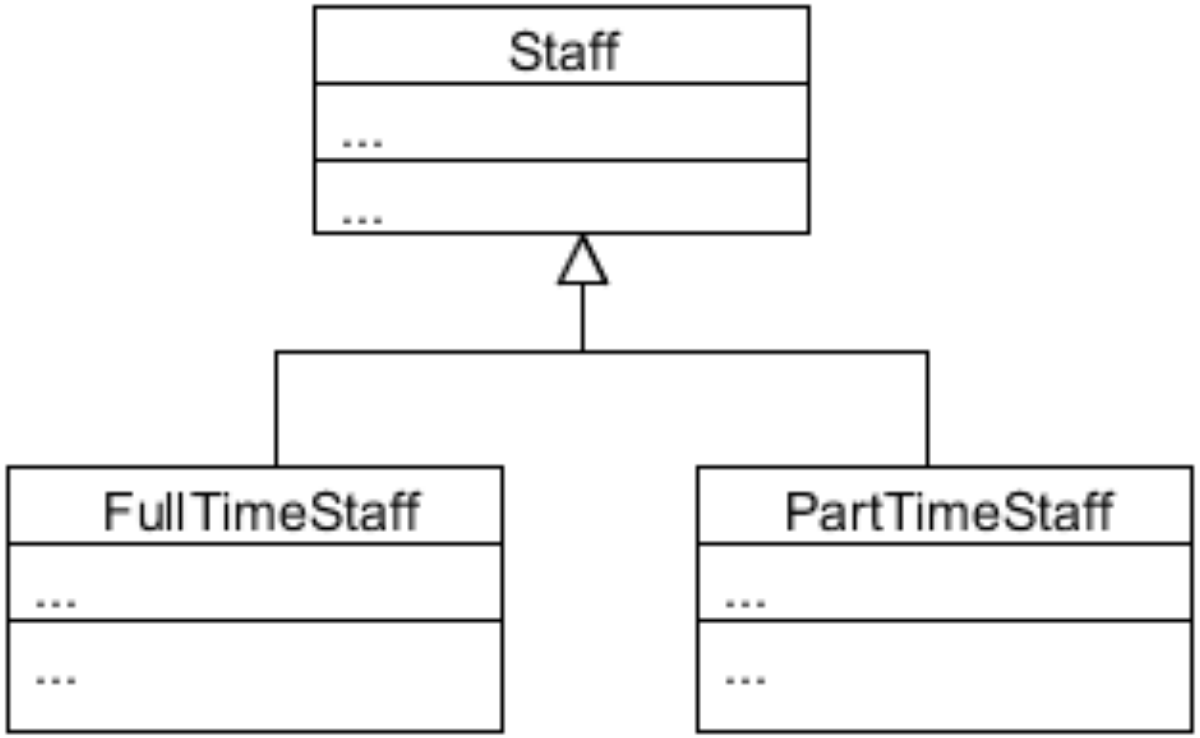
Polymorphism

- Poly
  - Many
  - Polygon (a figure having many sides)
  - Polytechnics (a school that teaches many different technical skills)
- Morph – change into something
- Polymorphism
  - Ability of an object to take on many forms

- Can be achieved by
  - Polymorphism by Inheritance
  - Polymorphism by Interface (later)

Polymorphism by Inheritance

- Consider this class design



Polymorphism by Inheritance

- The Staff class

```
public class Staff {

    private String employeeNumber;
    private String name;
    private String dept;
    private double monthlyPay;

    public Staff(String employeeNumber, String name,
                 String dept, double monthlyPay) {
        this.employeeNumber = employeeNumber;
        this.name = name;
        this.dept = dept;
        this.monthlyPay = monthlyPay;
    }

    public String getEmployeeNumber() {
        return this.employeeNumber;
    }

    public String getName() {
        return this.name;
    }

    public String getDept() {
        return this.dept;
    }

    public double getMonthlyPay() {
        return this.monthlyPay;
    }


    public double calculatePay() {
        return this.monthlyPay;
    }

    public String toString() {
        return this.employeeNumber+" " + this.name + " " + this.dept;
    }
}
```

Polymorphism by Inheritance

- The FullTimeStaff class

```
public class FullTimeStaff extends Staff{

    private double allowance;

    public FullTimeStaff(String employeeNumber, String name, String dept,
                         double monthlyPay, double allowance) {
        super(employeeNumber, name, dept, monthlyPay);
        this.allowance = allowance;
    }

    //override
    public double calculatePay() {
        return getMonthlyPay()+this.allowance;
    }
}
```

Polymorphism by Inheritance

- The PartTimeStaff class

```
public class PartTimeStaff extends Staff{

    private double hoursWorked;
    private double hourlyRate;

    public PartTimeStaff(String employeeNumber, String name, String dept,
                         double hoursWorked,
                         double hourlyRate) {
        super(employeeNumber, name, dept, 0);
        this.hoursWorked = hoursWorked;
        this.hourlyRate = hourlyRate;
    }

    public double calculatePay() {
        return this.hoursWorked*this.hourlyRate;
    }
}
```

Polymorphism by Inheritance

Can assign FullTimeStaff or PartTimeStaff data type to Staff.
s2 polymorph into FullTimeStaff, s3 polymorph into PartTimeStaff

```
Staff s1 = new Staff("001","Peter","IT",10000);
Staff s2 = new FullTimeStaff("002","John","Acc",5000,300);
Staff s3 = new PartTimeStaff("T01","Tim","Sales",10,50);

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);

System.out.println(s1.calculatePay());
System.out.println(s2.calculatePay());
System.out.println(s3.calculatePay());
```

```
001 Peter IT
002 John Acc
T01 Tim Sales
10000.0
5300.0
500.0
```

Polymorphism by Inheritance With ArrayList

staffs ArrayList can store any objects of type Staff and also all Staff's children.

```
ArrayList<Staff> staffs = new ArrayList<Staff>();
staffs.add(s1);
staffs.add(s2);
staffs.add(s3);
```

staff can polymorph into Staff, FullTimeStaff or PartTimeStaff

```
for(Staff staff: staffs) {
    System.out.println(staff);
    System.out.println(staff.calculatePay());
}
```

```
001 Peter IT
002 John Acc
T01 Tim Sales
10000.0
5300.0
500.0
```

Abstract class and abstract method

- Abstract class (recap)
  - Cannot create an object from Abstract class
  - Only for Inheritance
- Abstract method
  - A method declaration in an abstract class
  - The class that extends from the abstract class with abstract methods MUST implement all the abstract method.

Abstract class with abstract method

- Assume that in the HR system, there is no need for Staff object.
  - Only FullTimeStaff and PartTimeStaff object.
- Notice that the calculatePay() is not used in the sub class.
  - But we need to have it in super class for the sub class to override so that polymorphism can works.
- In this case, a better design to to make:
  - Staff class abstract and
  - calculatePay() method abstract
- Note: an abstract method must be in an abstract class

```
abstract public class Staff {

    private String employeeNumber;
    private String name;
    private String dept;
    private double monthlyPay;

    public Staff(String employeeNumber, String name,
                 String dept, double monthlyPay) {
        this.employeeNumber = employeeNumber;
        this.name = name;
        this.dept = dept;
        this.monthlyPay = monthlyPay;
    }

    public String getEmployeeNumber() {
        return this.employeeNumber;
    }

    public String getName() {
        return this.name;
    }

    public String getDept() {
        return this.dept;
    }

    public double getMonthlyPay() {
        return this.monthlyPay;
    }

    abstract public double calculatePay();

    public String toString() {
        return this.employeeNumber+" " + this.name + " " + this.dept;
    }
}
```

Interface

- In the previous example, the most important method is calculatePay()
- All FullTimeStafff and PartTimeStaff objects need to have a calculatePay() method.
- We can create an interface

```
public interface HRPay {

    public double calculatePay();

}
```

This is an “abstract” method

- Which ever class that implements the HRPay interface must have the calculatePay() method.
- The calculatePay() method is an “abstract” method.

Implement Interface

- Alternately, Staff can implements the HRPay interface

```
public class Staff implements HRPay{

    private String employeeNumber;
    private String name;
    private String dept;
    private double monthlyPay;
```

- The Staff class MUST have the calculatePay() method.

```
public double calculatePay() {
    return this.monthlyPay;
}
```

- The child class of Staff can choose to override calculatePay().
- The fact that the super class implements the HRPay Interface guaranteed that the super class and the child classes have the method calculatePay().

Polymorphism by Interface

HRPay is an interface, it can also be a datatype!

```
HRPay s1 = new Staff("001","Peter","IT","CTO",10000);
HRPay s2 = new FullTimeStaff("002","John","Acc","Manager",5000,300);
HRPay s3 = new PartTimeStaff("T01","Tim","Marketing",10,50);

System.out.println(s1.calculatePay());
System.out.println(s2.calculatePay());
System.out.println(s3.calculatePay());
```

```
10000.0
5300.0
500.0
```

s1, s2 and s3 are of the type “HRPay”
It does not care which “datatype” is attached to s1,s2 or s3
All it cares is that it has the method calculatePay().

This is also known as “Late-binding”
The actual “datatype” is only “binded” into the variable during “run time”.

Polymorphism by Interface with ArrayList

```
ArrayList<HRPay> staffs = new ArrayList<HRPay>();
staffs.add(s1);
staffs.add(s2);
staffs.add(s3);

for(HRPay staff:staffs){
    System.out.println(staff);
    System.out.println(staff.calculatePay());
}
```

The staffs ArrayList can store any objects that implements the HRPay interface.

Design by Contract

- An approach of designing software
- Software designer should define formal and precise “interface” specification for software components.
- The “Interface” class of Java can be used to “implement” this idea.

Design by Contract

- What if now we have a new class in the HR system; InternStaff class.
- The monthly pay depends on which department.
  - IT and Accounting 800
  - Others 500
- The System Analysts can give the above requirement and the “Contract” to the programmer.
- The “Contract” is the “HRPay” interface.

InternStaff Class

```
public class InternStaff implements HRPay {

    private String name;
    private String school;
    private String dept;

    public InternStaff(String name, String school, String dept) {
        this.name = name;
        this.school = school;
        this.dept = dept;
    }

    public double calculatePay(){
        double pay = 0;
        if(dept.equals("IT") || dept.equals("Acc")){
            pay = 800;
        }
        else{
            pay = 500;
        }
        return pay;
    }

}
```

Polymorphism using Interface

```
HRPay s1 = new Staff("001","Peter","IT","CTO",10000);
HRPay s2 = new FullTimeStaff("002","John","Acc","Manager",5000,300);
HRPay s3 = new PartTimeStaff("T01","Tim","Marketing",10,50);
HRPay s4 = new InternStaff("Ben","UoW","IT");

ArrayList<HRPay> staffs = new ArrayList<HRPay>();
staffs.add(s1);
staffs.add(s2);
staffs.add(s3);
staffs.add(s4);
```

```
for(HRPay staff:staffs){
    System.out.println(staff);
    System.out.println(staff.calculatePay());
}
```

Note:
InternStaff is not a child of Staff class.
We still can put InternStaff object into staffs ArrayList as it accepts any object that implements HRPay.

This part of the codes remains the same.