SOFTWARE DESIGN & ANALYSIS (Week-11)

USAMA MUSHARAF

MS-CS (Software Engineering)

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

WEEK # 11

- Polymorphism
- Abstract Class VS Interface

POLYMORPHISM

- ***Polymorphism in java*** *is a concept by which we can perform a single action by different ways.*

- *Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.*

- *There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism.*

- *We can perform polymorphism in java by method overloading and method overriding.*

POLYMORPHISM

- ***Runtime polymorphism*** *is a process in which a call to an overridden method is resolved at runtime rather than compile-time.*
- *In this process, an overridden method is called through the reference variable of a super class.*
- *The determination of the method to be called is based on the object being referred to by the reference variable.*

```
class Animal{
void eat() {System.out.println("eating...");}
}
class Dog extends Animal{
void eat() {System.out.println("eating bread...");}
}
class Cat extends Animal{
void eat() {System.out.println("eating rat...");}
}
class Lion extends Animal{
void eat() {System.out.println("eating meat...");}
}
```

```
class TestPolymorphism {
public static void main(String[]
args)
    {
      Animal a;
      a=new Dog();
      a.eat();
      a=new Cat();
      a.eat();
      a=new Lion();
      a.eat();
       }
      }
```

Output:

```
eating bread... eating rat... eating meat...
```

STATIC AND DYNAMIC BINDING

Association of method definition to the method call is known as binding.

There are two types of binding:

Static binding and dynamic binding.

The binding which can be resolved at compile time by compiler is known as static or early binding.

All the static, private methods have always been bonded at ***compile time*** *.*

Compiler knows that all such methods cannot be overridden and will always be accessed by object of local class.

Hence compiler doesn’t have any difficulty to determine object of class (local class for sure).

That’s the reason binding for such methods is static.

Example:

```
class Human {
     ....
     }

class Boy extends Human {
public void walk() {
System.out.println("Boy walks");
     }

public static void main( String args[]) {
Boy obj1 = new Boy();
obj1.walk();
     }
}
```

Note:
Here we have created an object of Boy class and calling the method walk() of the same class. Since nothing is ambiguous here, compiler would be able to resolve this binding during compile-time, such kind of binding is known as static binding.

- *When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late Binding.*

- *Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method.*

- *Thus while calling the overridden method, the compiler gets confused between parent and child class method(since both the methods have same name).*

CASE STUDY: PAYROLL SYSTEM USING POLYMORPHISM

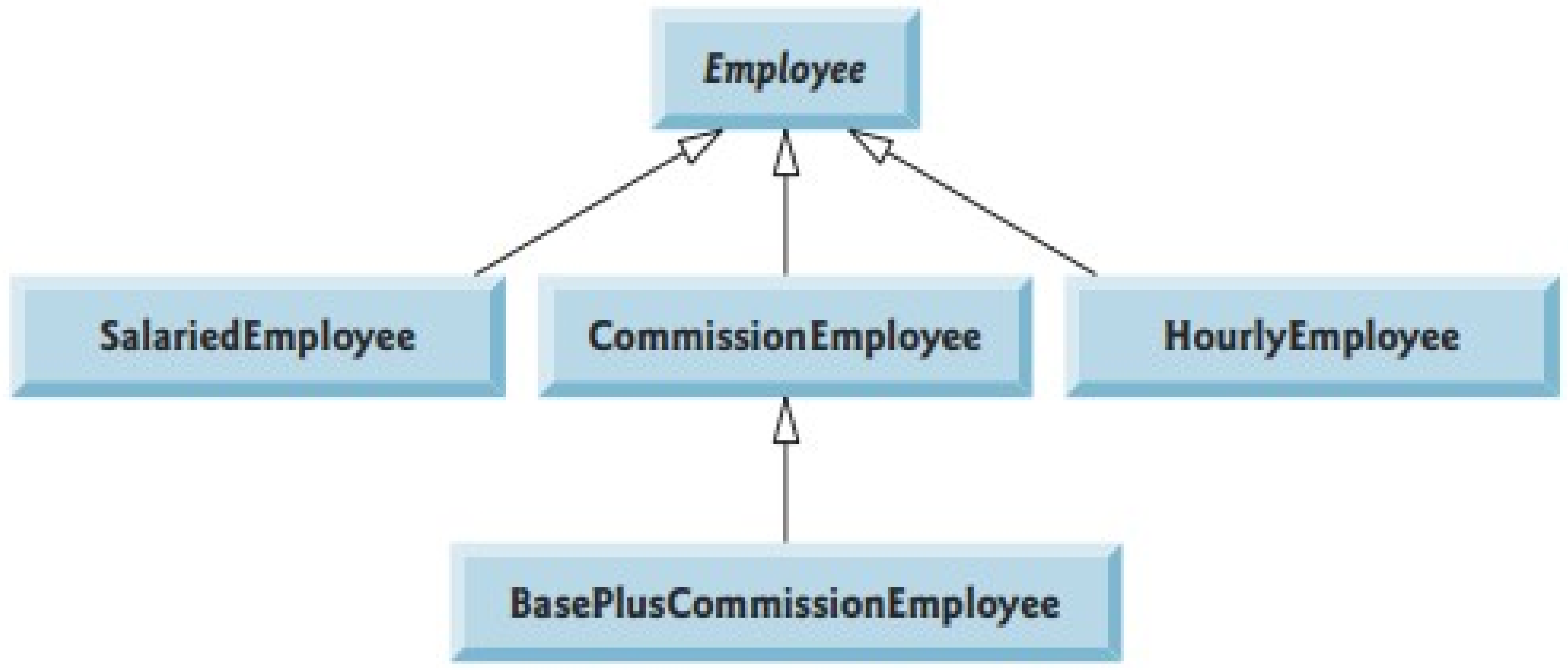
A company pays its employees on a weekly basis. The employees are of four types:

- **Salaried employees** are paid a fixed weekly salary regardless of the number of hours worked,
- **Hourly employees** are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) for all hours worked in excess of 40 hours,
- **Commission employees** are paid a percentage of their sales and **base-salaried commission employees** 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 base salaries.

The company wants to write an application that performs its payroll calculations polymorphically.

CASE STUDY: PAYROLL SYSTEM USING POLYMORPHISM



Abstract Class

A class that is declared using “abstract” keyword is known as abstract class.

Abstract classes may or may not contain abstract methods ie., methods with out body (public void get();)

which means in abstract class you can have concrete methods (methods with body) as well along with abstract methods (without body).

Abstract class can extend only one class or one abstract class at a time.

A class can extend only one abstract class.

An abstract class can not be **instantiated** (you are not allowed to create **object** of Abstract class).

Since abstract class allows concrete methods as well, it does not provide 100% abstraction.

You can say that it provides partial abstraction.

Interfaces are used for 100% abstraction.

Key Points to Remember

An abstract class has no use until unless it is extended by some other class.

If you declare an **abstract method** in a class then you must declare the class abstract as well. you can’t have abstract method in a **non-abstract class**.

Abstract class can have non-abstract method (concrete) as well.

Abstract method has no body.

Always end the declaration with a semi colon.

Why Need an Abstract Class?

Suppose there is a class Animal and there are few other classes like Cat, Dog and Horse.

These classes extends Animal class so basically they are having few common habits(methods in technically) which they are inheriting from Animal class.

Now, if you have understood the above example then you would have been able to figure out that **creating object of Animal class has no significance** as you can’t judge that the **new** object of Animal class will represent which animal.

Hence for such kind of scenarios we generally creates an **abstract classes.**

```
    abstract class Demo1{
    public void disp1() {
    System.out.println("Concrete method of abstract class");
    }
    abstract public void disp2();
    }
    class Demo2 extends Demo1{
    public void disp2() {
    System.out.println("I'm overriding abstract method");
    }
    public static void main(String args[]) {
    Demo2 obj = new Demo2();
    obj.disp2();
    }
}
```

OUTPUT:

I'm overriding abstract method

Abstract Vs Concrete:

A class which is not abstract is referred as ***Concrete class****.*

In the given example, animal is a abstract class and Cat , Dog and Horse are concrete classes .

Interface

Interface are special java type which contains only a set of method prototypes, but does not provide the implementation for these prototypes.

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method names, no body).

Interface is tantamount to a pure abstract class.

Interfaces are used for 100% abstraction.

Why Need an Interface?

As mentioned before they are used for pure abstraction.

Methods in interfaces do not have body, they have to be implemented by the class before you can access them.

The class that implements interface must implement all the methods of that interface.

Java programming language does not support multiple inheritance.

Using interfaces we can achieve this as a class can implement more than one interfaces.

```
public class Abc implements MyInterface {
    public void method1()
    {
    System.out.println ("implementation of method1");
     }
   public void method2()
   {
  System.out.println("implementation of method2");
   }

   public static void main(String args []){
    Abc obj = new Abc();
     obj. method1();
    }
}
```

```
public interface
MyInterface {

 public void method1();
 public void method2();
}
```

Multiple Inheritance with Interfaces

```
interface A {
public void aaa();
 }
interface B {
public void bbb();
}
class Central implements A,B {
public void aaa() {
 //Any Code here
}
public void bbb() {
 //Any Code here
}
```

```
public static void
main(String args[]) {
 //Statements
     }
}
```

Key points about Interface

1-Interface provides complete abstraction as none of its methods can have body. On the other hand, abstract class provides partial abstraction as it can have abstract and concrete methods both.

2-Implements keyword is used by classes to implement an interface.

3- Class implementing any interface must implement all the methods, otherwise the class should be declared as “abstract”.

4- Variable names conflicts can be resolved by interface name e.g:

```
interface A { int x=10; }
interface B { int x=100; }
class Hello implements A,B {
public static void Main(String args[]) {
System.out.println(x);          // reference to x is ambiguous both variables are x.
System.out.println(A.x);
System.out.println(B.x);
}
}
```

Difference b/w abstract class vs Interface

1-Abstract class can extend only one class or one abstract class at a time.

Interface can extend any number of interfaces at a time.

2-Abstract class can extend from a class or from an abstract class.

Interface can extend only from an interface.

3-Abstract class can have both abstract and concrete methods.

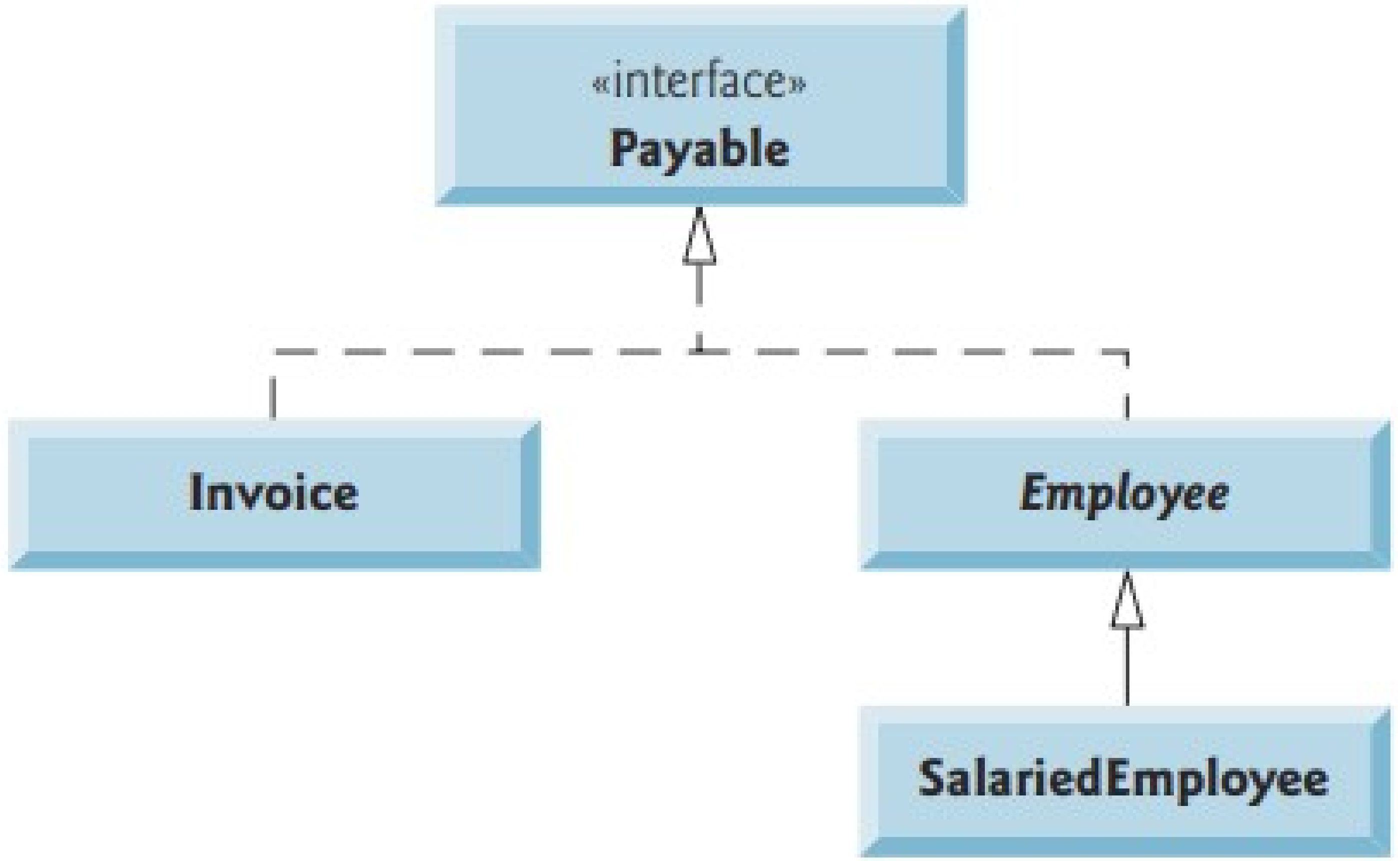
Interface can have only abstract methods.

4- A class can extend only one abstract class.

A class can implement any number of interfaces.

5- In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract

In an interface keyword ‘abstract’ is optional to declare a method as an abstract.


«interface»
Payable
Invoice
Employee
SalariedEmployee

Lets Code!

HAVE A GOOD DAY!