Unit 06

Polymorphism

CMPS 251, Fall 2020, Dr. Abdulaziz Al-Ali

Checkpoint (for Inheritance unit)

- Which statements is/are invalid according to our last code?

A. Glasses a = new MedicalGlasses(…)

B. Glasses[] a = new Glasses[3]
   a[0] = new MedicalGlasses(…)

C. SunGlasses a = new Glasses(….)

D. Glasses[] a = new Glasses[4];
   a[1] = new SunGlasses(…)

Checkpoint (from inheritance unit)

- Can we override private methods?

Objectives

- Introduce abstract classes and methods
- Introduce interfaces

Polymorphism

- Inheritance gives us the opportunity to program in general instead of in specifics
  - Write 1 piece of code that can process an entire group of objects that share the same superclass

```
public double calculateTotalPay(ArrayList<Employee> list) {
        double ret = 0;
        for (Employee e: list) {
                ret += e.getPayAmount();
        }
        return ret;
}
```

  - This code works for all types of Employees. (Employee, ComissionEmployee, etc.)

Abstract Classes

- An abstract class is one that is designed to function solely as a super class
  - You would **never** create one with `new`
- An abstract class can also contain abstract methods
  - A placeholder that specifies that all subclasses **must** define this method
- You can't declare constructors as abstract

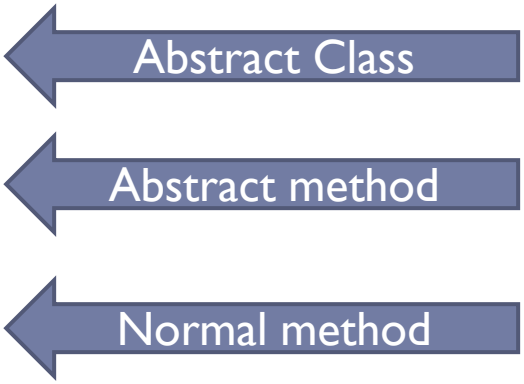
Abstract Class Example

Shape.java

```
public abstract class Shape {

    public abstract double getArea();

    public String getName() {
        return "Shape";
    }
}
```



Abstract Class Example

Shape.java

```
public abstract class Shape {

    public abstract double getArea();

    public String getName() {
        return "Shape";
    }
}
```

Rectangle.java

```
public class Rectangle extends Shape{
    private double width;
    private double height;

    public Rectangle(int w, int h) {
        this.width = w;
        this.height = h;
    }

    @Override
    public double getArea() {
        double area = width * height;
        return area;
    }

    @Override
    public String getName() {
        return "Rectangle";
    }
}
```

Abstract Class Example

Shape.java

```
public abstract class Shape {

    public abstract double getArea();

    public String getName() {
        return "Shape";
    }
}
```

Circle.java

```
public class Circle extends Shape {
    private double r;

    public Circle(double r) {
        this.r = r;
    }

    @Override
    public double getArea() {
        return Math.PI * r * r;
    }

    @Override
    public String getName() {
        return "Circle";
    }
}
```

Demo

- Practice todos 1-13 in this unit sample code.

Check point

- What are abstract classes?
- What can you NOT do with abstract classes?
- What can abstract classes have that regular classes cannot?
- What is the purpose of abstract methods?

Interfaces

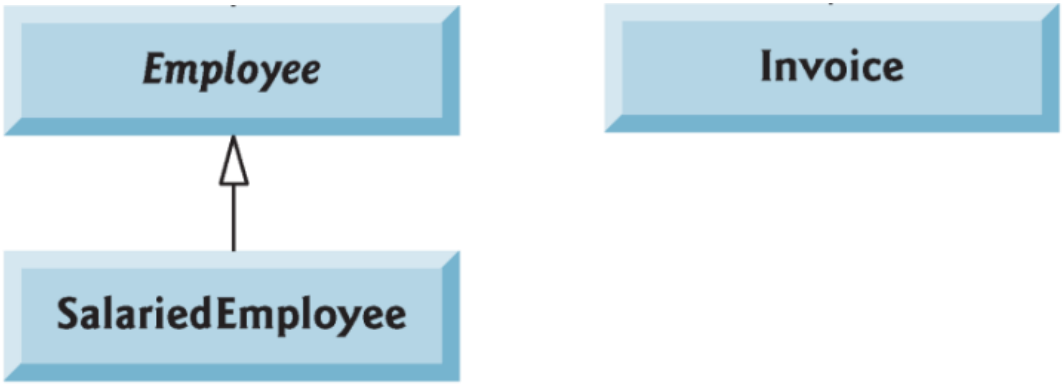
- An interface is like an abstract class, but it only contains constants and abstract methods
- Interfaces are used to specify specific methods to do something
- A class can implement more than one interface
  - This is the big difference between interfaces and inheritance
- A class that implements an interface is basically promising to implement all the methods provided by the interface

Why Use an Interface?

- When unrelated classes need to share common methods

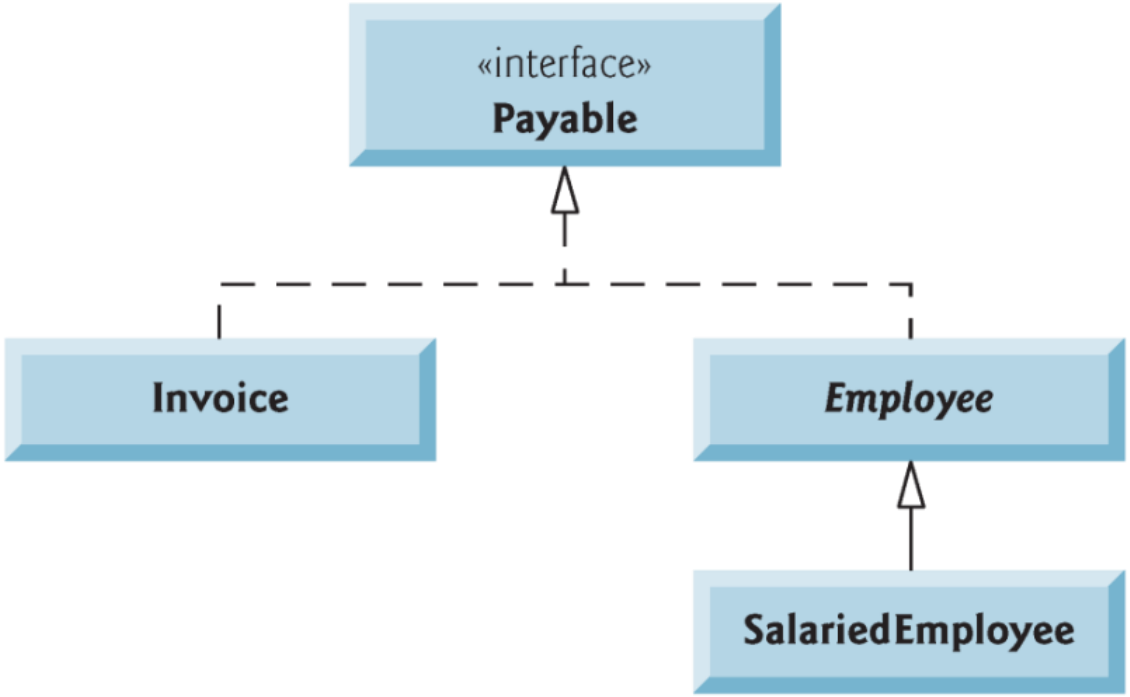
Interface Example

- Imagine a company system that has Employees and Invoices



- `Employee` and `Invoice` are not related by inheritance
- But to the company, they have a similar property: They are both *Payable*

Interface Example



Interface Example

Payable.java

```
public interface Payable {
    double getPaymentAmount();
}
```

Employee.java

```
public class Employee implements Payable{
    ...
    @Override
    public double getPaymentAmount() {
        return this.salary;
    }
    ...
}
```

Invoice.java

```
public class Invoice implements Payable {
    ...
    @Override
    public double getPaymentAmount() {
        return this.totalBill;
    }
    ...
}
```

Question

- Can we implement multiple interfaces in a class?
- How can we add multiple interfaces to a class?
  - Answer: we separate them with commas (,)
- Example:

```
public class Employee implements Payable, Insurable
```

- This means the Employee class implements both interfaces (Payable and Insurable)

Check point

- What is wrong with this code?

```
public abstract class Person
{
        String name;
        public abstract void displayPerson();
}

public class Student extends Person{
}
```

What is wrong with this code?

```
public abstract class Alpha{
        private int x;
        public Alpha(int x){
                this.x = x;
        }
}
```

```
public class Beta extends Alpha{
        private int y;
        public Beta(int y){
                this.y = y;
        }
}
```

```
public static void main(String args[])
{
        Beta b = new Beta(2);
        Alpha a = new Alpha(3);
}
```

Who can use Abstract and Non-Abstract Methods?

	Regular Not-Abstract methods	Abstract Methods
Normal Classes		
Abstract Classes		
Interfaces		

Who can use Abstract and Non-Abstract Methods?

	Regular Not-Abstract methods	Abstract Methods
Normal Classes	✓	
Abstract Classes	✓	✓
Interfaces		✓

Check point

- Suppose in your project you have the classes, *Animal*, *Cat*, and *Lion*. If *Animal* is the parent of *Cat*, and *Cat* is the parent of *Lion*, create an array that can store only *Cats* and *Lions* at the same time.

- If *Animal* was **abstract**, and you knew these are the only three classes in your project. Would your answer change? How and why?

Back to our demo

- See todos 14-28 in the sample code of this unit sample code.

Interfaces Summary

- Think of interfaces as a contract that can be signed by any class (both abstract and normal classes).
  - *Implementing* an interface is like *signing* the contract.
- By *implementing* an interface, a class promises to provide the body for all the abstract methods defined in the interface.
- If the class that implements the interface is *abstract*, it does not have to (but could if you want to make it) define these methods inside of it. This is because *abstract classes* are allowed to contain *abstract methods*.
- If an abstract *parent class* implements an interface and does NOT do the abstract methods inside of it, then all the not abstract *child classes* of this abstract parent must implement the abstract methods of the interface.