

Computer Language



OOP 4: Abstraction



Agenda

- Abstract Class
- Interface

Abstract Class

Interface

Abstract Class: Goal

■ Abstraction

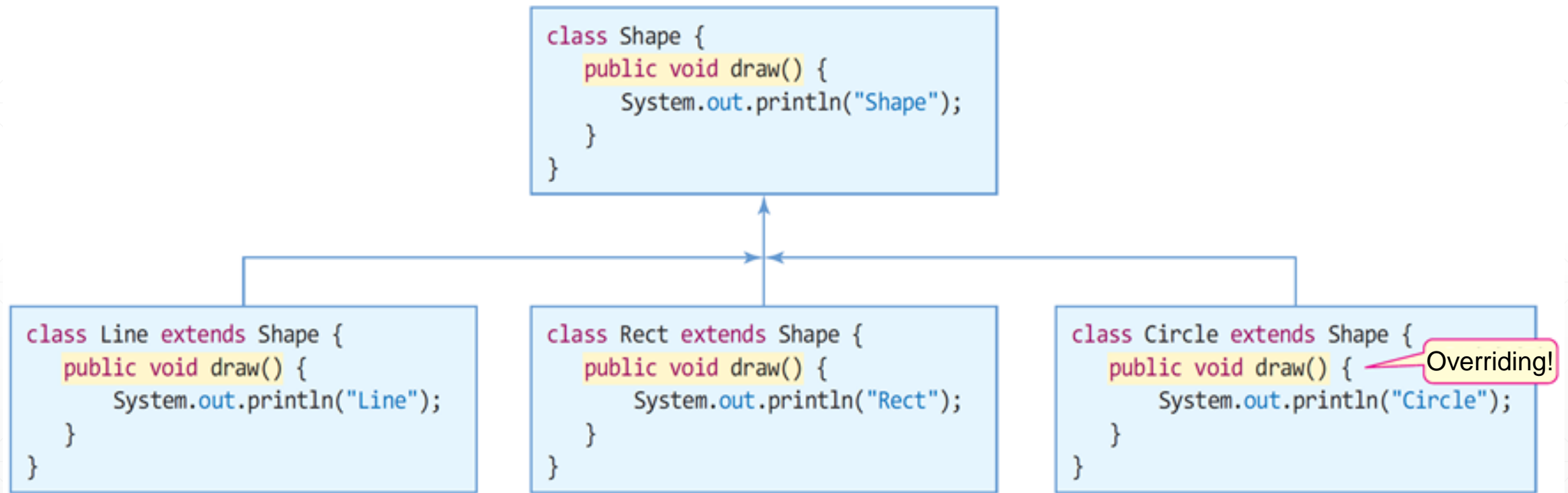
- Extraction of common features from a set of similar instances
 - Example 1) bird, insect, fish → animal (abstraction)
 - Example 2) Samsung, Hyundai, LG → company (abstraction)

■ Abstract class

- Class to define the common fields and methods of concrete classes
- Act as a parent (base) class for concrete classes

Abstract Class: Goal (cont'd)

- What's different? Compared to standard inheritance relationship?



Abstract Class: Goal (cont'd)

■ Goal of Abstract class

- Separate interface (design) from implementation!

■ Abstract class

- Define common concepts
- Declare 'abstract' methods that **MUST** be implemented in the subclasses
 - Abstract method does not have implementations!

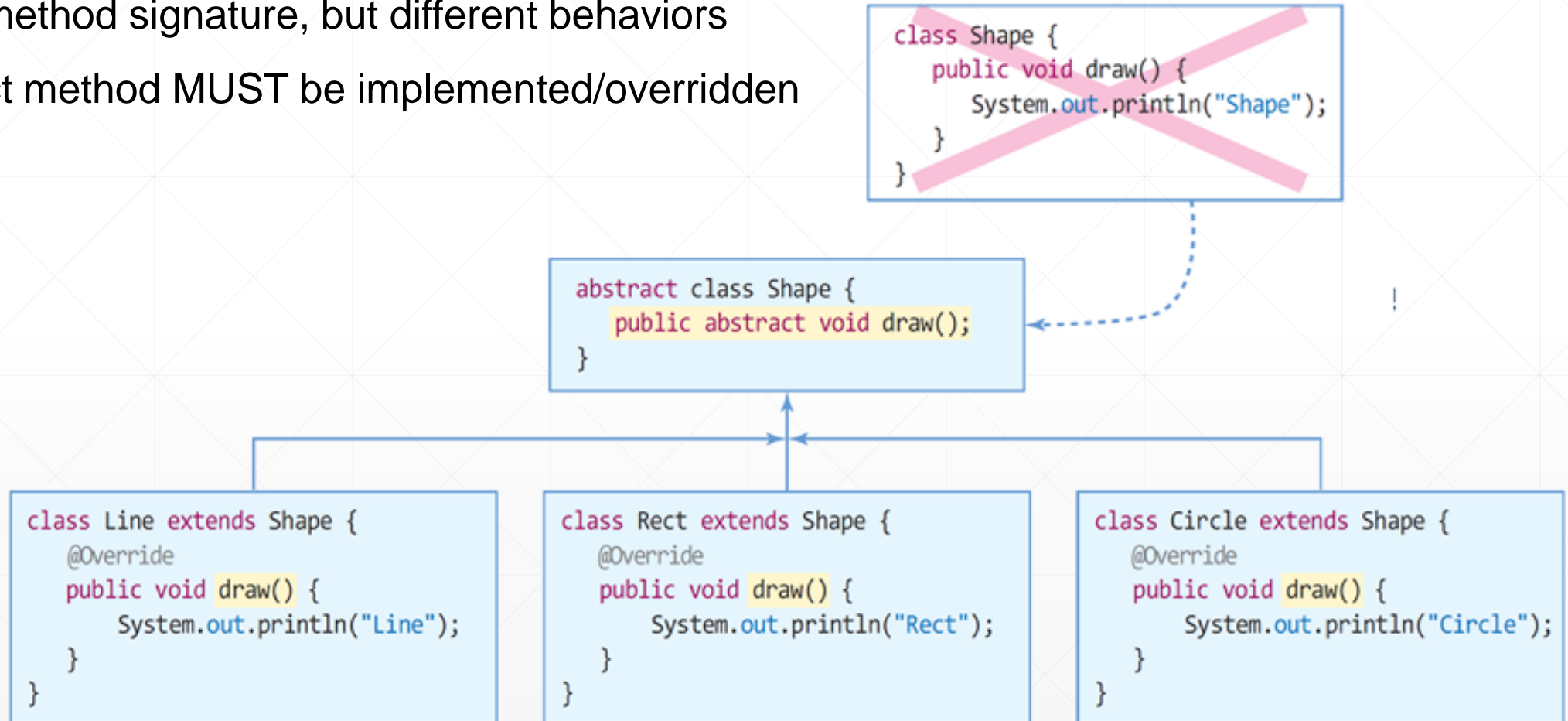
■ Concrete classes

- Implement class-specific behaviors

Abstract Class: Goal (cont'd)

■ Redefinition of superclass's method in the subclass

- Same method signature, but different behaviors
- Abstract method MUST be implemented/overridden



Abstract Class: Goal (cont'd)

■ Redefinition of superclass's method in the subclass

- Same method signature, but different behaviors
- Abstract method **MUST** be implemented/overridden

■ Achieves polymorphism with inheritance

- Same interface, but different behaviors
 - Line class draws a line using draw() interface
 - Circle class draws a circle using draw() interface
 - Rect class draws a rectangle using draw() interface

Abstract Class: Definition

■ Use abstract keyword!

- Abstract Class
- Abstract method
 - Defined but not implemented method
 - MUST be overridden by subclasses

■ Characteristics of abstract classes

- Cannot be instantiated by new() keyword
- May or may not include abstract methods
- If a class includes abstract methods, then the class MUST be declared abstract

Abstract Class: Definition (cont'd)

■ Characteristics of abstract classes

- May or may not include abstract methods

```
// 1. abstract class containing abstract methods
```

```
abstract class Shape { // declaration of abstract class
    public Shape() { }
    public void paint() { draw(); }
    abstract public void draw(); // declaration of abstract method
}
```

← No implementation for abstract methods

```
// 2. abstract class without abstract methods
```

```
abstract class MyComponent { // declaration of abstract class
    String name;
    public void load(String name) {
        this.name = name;
    }
}
```

Abstract Class: Definition (cont'd)

■ Inheritance of abstract classes

➤ Abstract class inheriting another abstract class

- Subclass cannot be instantiated

```
abstract class Shape { // abstract class
    public Shape() { }
    public void paint() { draw(); }
    abstract public void draw(); // abstract method
}
abstract class Line extends Shape { // abstract class, not implementing draw() method
    public String toString() { return "Line"; }
}
```

➤ Concrete class inheriting an abstract class

- All abstract methods MUST be implemented (overriding)
- Concrete subclass can be instantiated

Abstract Class: Usecases

■ Example)

- An array containing various payment methods
- Process a series of payments using abstraction and polymorphism

```
public class MethodOverridingEx {  
    static void purchase(Payment[] pay){  
        for (Payment s: pay){  
            s.pay(1000);  
        }  
    }  
  
    public static void main(String[] args) {  
        Payment[] myPayments = new Payment[3];  
        myPayments[0] = new Cash();  
        myPayments[1] = new Bitcoin();  
        myPayments[2] = new Credit();  
  
        purchase(myPayments);  
    }  
}
```

Abstract Class: Usecases (cont'd)

■ Example)

- An array containing various payment methods
- Process a series of payments using abstraction and polymorphism

```
abstract class Payment {  
    abstract void pay(int money);  
}  
  
class Cash extends Payment {  
    void pay(int money) { System.out.println("Success!" + money + " Won paid"); }  
}  
  
class Bitcoin extends Payment {  
    void pay(int money) { System.out.println("Fail! Coin destroyed!"); }  
}  
  
class Credit extends Payment {  
    void pay(int money) { System.out.println("Success! Payment made with your card!"); }  
}
```



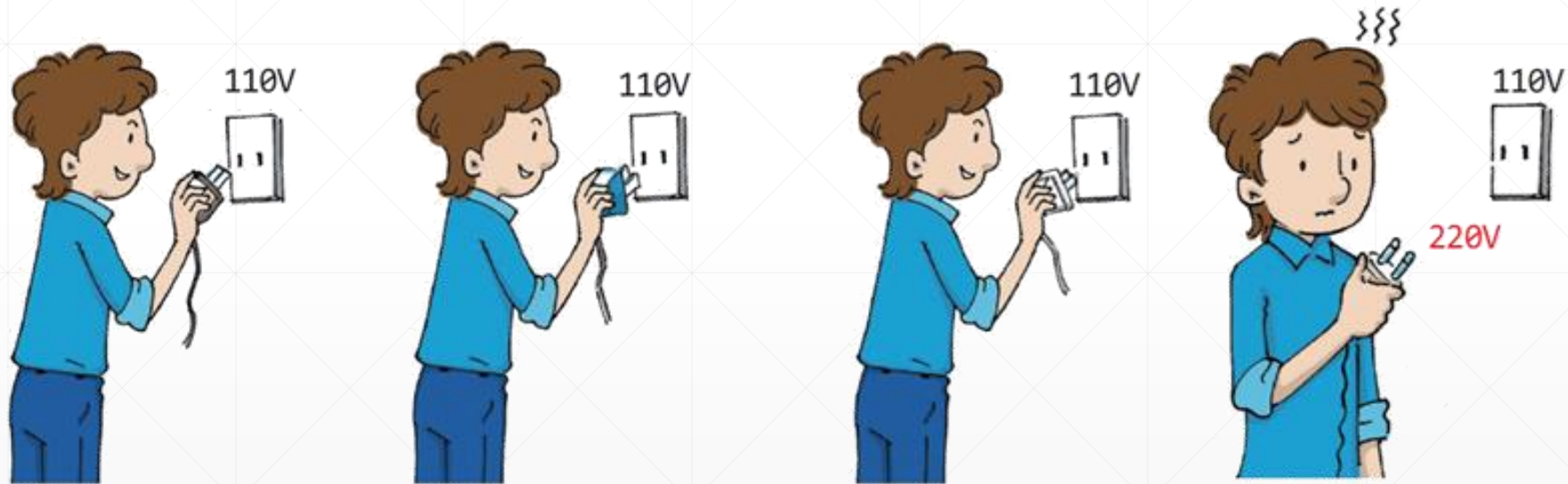
Abstract Class

Interface

Interface: Goal

■ Interface in real life

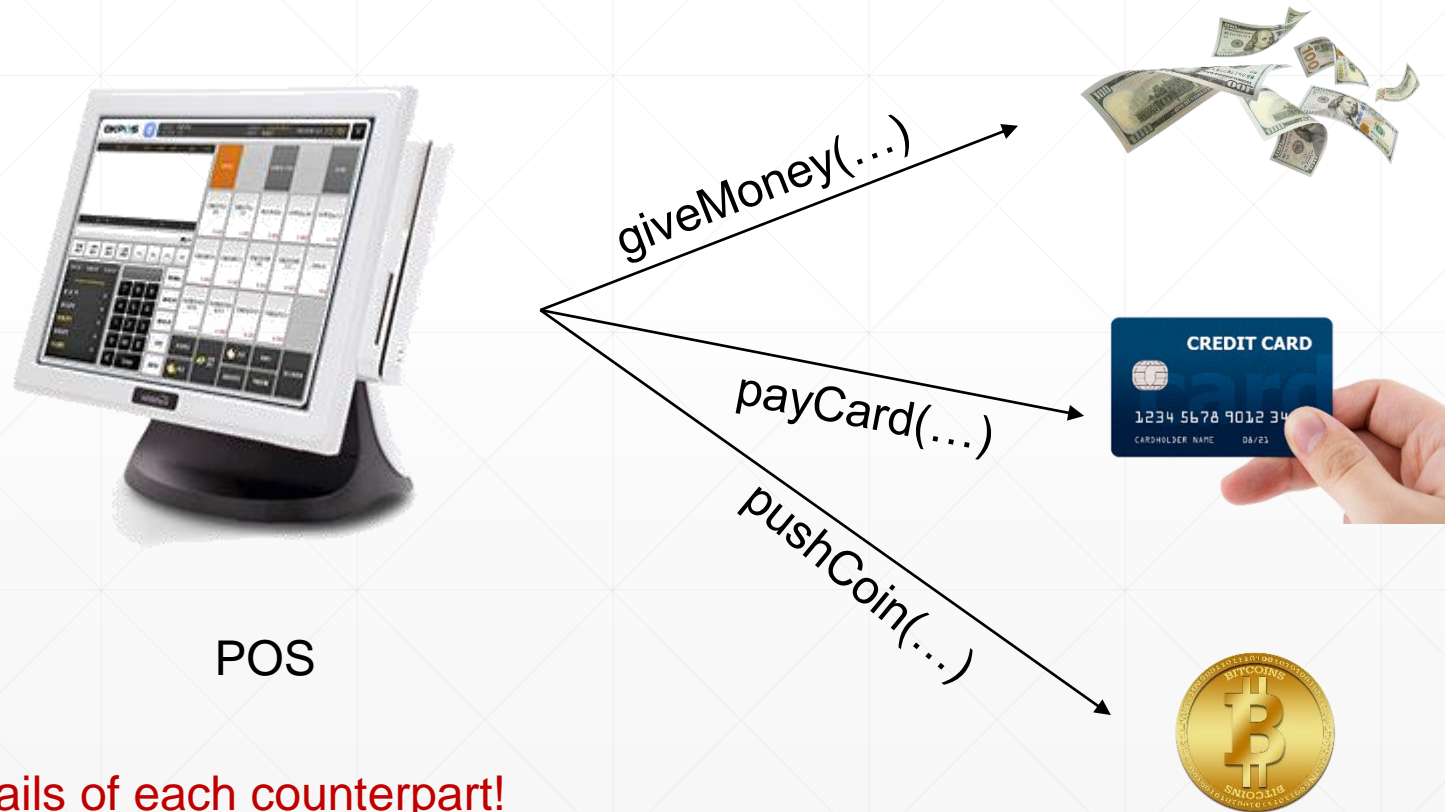
- Define a standard for interaction between devices



Interface: Goal (cont'd)

■ Interface in Java world

- Define a standard (contract) for interaction between class/objects

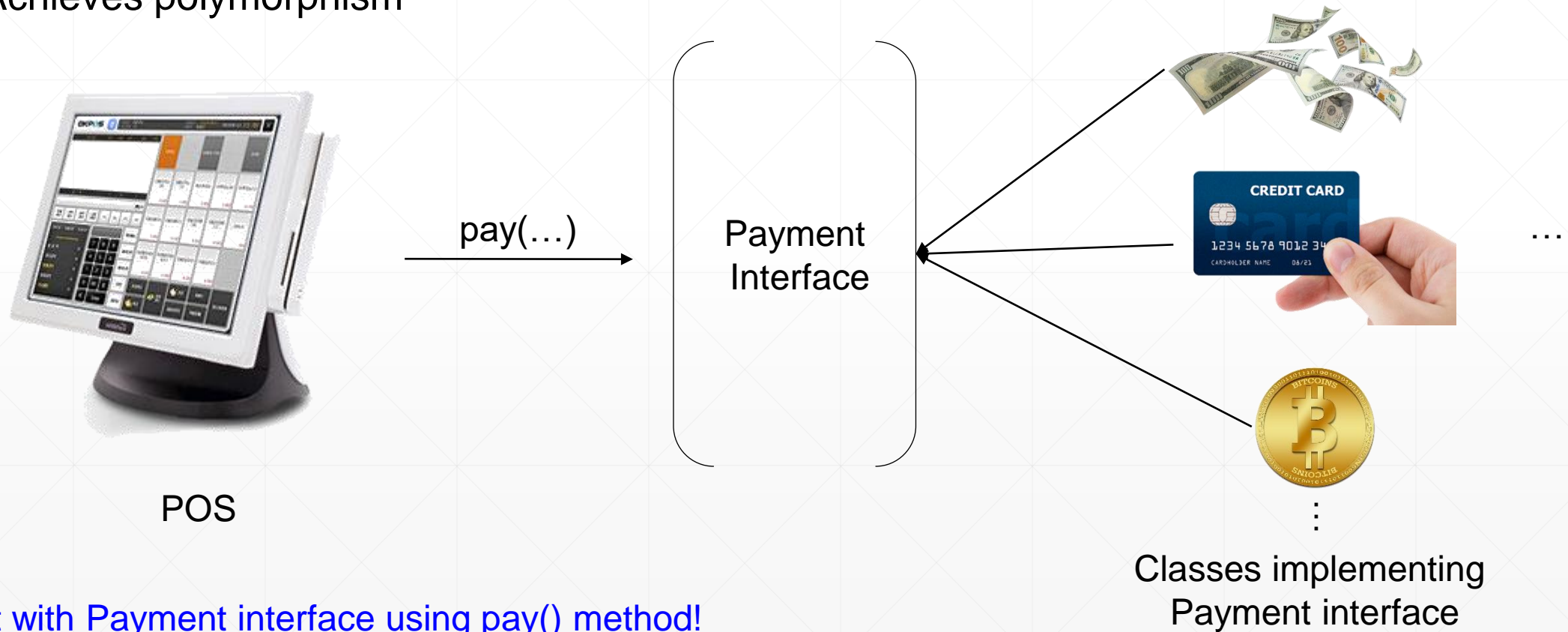


Need to know all the details of each counterpart!

Interface: Goal (cont'd)

■ Interface in Java world

- Define a standard (contract) for interaction between class/objects
- Achieves polymorphism



Only interact with Payment interface using `pay()` method!

Interface: Definition

■ Declaration of interface

- Use *interface* keyword!

■ Declaration of interface members

- Constants
- Abstract methods
- Default, private, static methods
- Variables are **NOT** allowed

Interface: Definition (cont'd)

■ Declaration of interface

- Use *interface* keyword!

```
interface PhoneInterface { // interface
    public static final int TIMEOUT = 10000; // constant
    public abstract void sendCall(); //
    public abstract void receiveCall(); // abstract methods
    public default void printLogo() { // default method
        System.out.println("** Phone **");
    };
}
```

■ Declaration of interface members

- Constants
- Abstract methods
- Default, private, static methods
- Variables are **NOT** allowed

Interface: Definition (cont'd)

■ Declaration of interface members

➤ Constants

- All fields defined in an interface are automatically declared as **public static final**
- Naming convention: USE CAPITAL LETTERS

➤ Abstract methods

- All methods defined in the interface are basically abstract without implementation
- 'public abstract' keywords can be omitted

```
interface PhoneInterface { //
    public static final int TIMEOUT = 10000; //
    public abstract void sendCall(); //
    public abstract void receiveCall();
    public default void printLogo() { //
        System.out.println("** Phone **");
    };
}
```

Interface: Definition (cont'd)

■ Declaration of interface members

➤ Default methods

- Methods with implementations
- MUST be declared as “default”
- Can be overridden by subclasses or interface realizations
- Access modifier: public

➤ Private methods

- Methods with implementations
- Access modifier: private
- Only accessible by the methods inside the same interface

➤ Static methods


- Can be either public or private (default: public)

```
interface PhoneInterface { //
    public static final int TIMEOUT = 10000; //
    public abstract void sendCall(); //
    public abstract void receiveCall();
    public default void printLogo() { //
        System.out.println("** Phone **");
    };
}
```

Interface: Characteristics

■ Instantiation

- Interfaces cannot be instantiated, like abstract classes
- Cannot use new() keyword for object instantiation

 `new PhoneInterface();`

■ Reference

- Reference variable of a certain interface type can be declared

`PhoneInterface galaxy;` **OK!**

■ Inheritance

- Can extend another interface
- Can extend multiple interfaces

Interface: Characteristics (cont'd)

■ Example of interface definition and inheritance

```
interface PhoneInterface {  
    final int TIMEOUT = 10000;  
    void sendCall();  
    void receiveCall();  
}
```

```
interface MP3Interface {  
    void play();  
    void stop();  
}
```

```
interface MobilePhoneInterface extends PhoneInterface {  
    void sendSMS();           // additional abstract methods  
    void receiveSMS();        // additional abstract methods  
}
```

Interface can extend another interface!

```
interface MusicPhoneInterface extends MobilePhoneInterface, MP3Interface {  
    void playMP3RingTone(); // additional abstract methods  
}
```

Interface can extend multiple interfaces!

Interface: Realization

- Use 'implements' keyword to realize a certain interface
 - Multiple realization is also allowed
 - All abstract methods defined in the interface MUST be implemented
- Example of interface realization

```
class SamsungPhone implements PhoneInterface {  
  
    public void sendCall() { System.out.println("RRRRiinnnnngg~~"); }  
    public void receiveCall() { System.out.println("Incoming call!!"); }  
  
    public void flash() { System.out.println("Mmmmmyyy Flash----!!"); }  
}
```

```
interface PhoneInterface {  
  
    final int TIMEOUT = 10000;  
  
    void sendCall();  
    void receiveCall();  
}
```



SamsungPhone's additional method

Interface: Realization (cont'd)

■ Example of interface realization (cont'd)

```
class LGPhone implements PhoneInterface {  
  
    public void sendCall() { System.out.println("Yap, my call!!!"); }  
    public void receiveCall() { System.out.println("Give me a call!!!"); }  
  
    public void knock() { System.out.println("knock, knock!"); }  
}
```

→ LGPhone's additional method

```
public class InterfaceEx{  
    public static void main(String[] args) {  
        SamsungPhone myPhone = new SamsungPhone();  
        LGPhone yourPhone = new LGPhone();  
  
        myPhone.sendCall();  
        myPhone.receiveCall();  
        myPhone.flash();  
  
        yourPhone.sendCall();  
        yourPhone.receiveCall();  
        yourPhone.knock();  
    }  
}
```

Interface: Realization (cont'd)

■ Example) default and private methods

- New requirement: Every phone should be able to print Phone logo!
 - 1) adding abstract printLogo() method? → existing interface broken
 - 2) adding default printLogo() method! → method with implementation in the interface

```
interface PhoneInterface {  
    final int TIMEOUT = 10000;  
  
    void sendCall();  
  
    void receiveCall();  
  
    default void printLogo() {  
        System.out.println("** Phone **");  
    }  
}
```

```
class IPhone implements PhoneInterface {  
  
    public void sendCall() { System.out.println("Yap, my call!!!"); }  
    public void receiveCall() { System.out.println("Give me a call!!!"); }  
  
    public void printLogo(){  
        System.out.println("IIIII PPPPHONE!");  
    }  
  
    public void watch() { System.out.println("Apple watch activated!"); }  
}
```

Overriding the default method

Interface: Realization (cont'd)

■ Example) default and private methods (cont'd)

- New requirement: Every phone should be able to print Phone logo!
- 1) adding abstract printLogo() method? → existing interface broken
- 2) adding default printLogo() method! → method with implementation in the interface

```
public class InterfaceEx{  
    public static void main(String[] args) {  
        SamsungPhone myPhone = new SamsungPhone();  
        LGPhone yourPhone = new LGPhone();  
        iPhone hisPhone = new iPhone();  
  
        myPhone.printLogo();  
        yourPhone.printLogo();  
        hisPhone.printLogo();  
    }  
}
```

} work!

Interface: Realization (cont'd)

■ Example) default and private methods (cont'd)

- New requirement: Every phone should be able to print Phone logo!
 - 1) adding abstract printLogo() method? → existing interface broken
 - 2) adding default printLogo() method! → method with implementation in the interface
 - Even we can define and invoke a private method in the interface!

```
interface PhoneInterface {  
    final int TIMEOUT = 10000;  
  
    void sendCall();  
  
    void receiveCall();  
  
    default void printLogo() {  
        System.out.println("*** Phone ***" + getPhoneID());  
    }  
  
    private int getPhoneID(){  
        return (int)(Math.random()*10);  
    }  
}
```

Interface: Realization (cont'd)

■ Example) static methods

➤ Utility method of an interface

- Static methods of an interface can be only accessed via Interface name

```
interface PhoneInterface {  
    final int TIMEOUT = 10000;  
  
    void sendCall();  
  
    void receiveCall();  
  
    default void printLogo() {  
        System.out.println("*** Phone ***"+getPhoneID());  
    }  
  
    private int getPhoneID(){  
        return (int)(Math.random()*10);  
    }  
  
    static int getTimeout(){  
        return TIMEOUT;  
    }  
}
```

```
public class InterfaceEx{  
    public static void main(String[] args) {  
        SamsungPhone myPhone = new SamsungPhone();  
        LGPhone yourPhone = new LGPhone();  
  
        System.out.println(PhoneInterface.getTimeout());  
        System.out.println(myPhone.getTimeout());  
    }  
}
```

Interface: Realization (cont'd)

```
interface MobilePhoneInterface extends PhoneInterface {  
    void sendSMS();  
    void receiveSMS();  
}
```

■ Realization of multiple interfaces

- A class can implement multiple interfaces

```
interface AllInterface {  
    void recognizeSpeech();  
    void synthesizeSpeech();  
}  
  
class AIPhone implements MobilePhoneInterface, AllInterface { // realization of multiple interfaces  
    // realize MobilePhoneInterface  
    public void sendCall() { ... }  
    public void receiveCall() { ... }  
    public void sendSMS() { ... }  
    public void receiveSMS() { ... }  
  
    // realize AllInterface  
    public void recognizeSpeech() { ... }  
    public void synthesizeSpeech() { ... }  
  
    // can add class-specific methods  
    public int touch() { ... }  
}
```

Interface: Realization (cont'd)

■ Example) Extending abstract class + implementing multiple interfaces

```
interface PhoneInterface {  
    final int TIMEOUT = 10000;  
    void sendCall();  
    void receiveCall();  
    default void printLogo() {  
        System.out.println("** Phone **");  
    }  
}
```

```
interface MobilePhoneInterface extends  
PhoneInterface {  
    void sendSMS();  
    void receiveSMS();  
}
```

```
interface AllInterface {  
    void recognizeSpeech();  
    void synthesizeSpeech();  
}
```

```
abstract class PDA {  
    public int calculate(int x, int y) {  
        return x + y;  
    }  
}
```

```
class AlPhone extends PDA implements MobilePhoneInterface, AllInterface { // realization of multiple interfaces
```

```
    public void sendCall() {  
        System.out.println("Al Sends call!");  
    }  
    public void receiveCall() {  
        System.out.println("Al receives call!");  
    }  
    public void sendSMS() {  
        System.out.println("Al sends sms!");  
    }  
    public void receiveSMS() {  
        System.out.println("Al receives sms!");  
    }  
}
```

realize MobilePhoneInterface

```
    public void recognizeSpeech() {  
        System.out.println("Al recognized your speech!");  
    }  
    public void synthesizeSpeech() {  
        System.out.println("Al synthesized your speech!");  
    }  
}
```

realize AllInterface

```
    // can add class-specific methods  
    public void touch() {  
        System.out.println("Don't touch me!");  
    }  
}
```

Interface: Realization (cont'd)

- Example) Extending abstract class + implementing multiple interfaces (cont'd)

```
public class InterfaceEx{  
    public static void main(String[] args) {  
        AIPhone myPhone = new AIPhone();  
        myPhone.touch();  
        myPhone.sendCall();  
        myPhone.receiveSMS();  
        myPhone.synthesizeSpeech();  
        System.out.println("My Phone can calculate: 10 + 10 = "+myPhone.calculate(10,10));  
    }  
}
```


Interface: Usecases

■ Example)

- An array containing various payment methods
- Process a series of payments using abstraction and polymorphism

```
interface Payment {  
    public void pay(int money);  
}  
  
class Cash implements Payment {  
    public void pay(int money) { System.out.println("Success!" + money + " Won paid"); }  
}  
  
class Bitcoin implements Payment {  
    public void pay(int money) { System.out.println("Fail! Coin destroyed!"); }  
}  
  
class Credit implements Payment {  
    public void pay(int money) { System.out.println("Success! Payment made with your card!"); }  
}
```

Interface: Usecases (cont'd)

■ Example)

- An array containing various payment methods
- Process a series of payments using abstraction and polymorphism

```
public class MethodOverridingEx {  
    static void purchase(Payment[] pay){  
        for (Payment s: pay){  
            s.pay(1000);  
        }  
    }  
  
    public static void main(String[] args) {  
        Payment[] myPayments = new Payment[3];  
        myPayments[0] = new Cash();  
        myPayments[1] = new Bitcoin();  
        myPayments[2] = new Credit();  
  
        purchase(myPayments);  
    }  
}
```

Abstract Class vs Interface

Abstract class	Interface
Abstract class does not support multiple inheritance	Interface supports multiple inheritance
Abstract class can have final, non-final, static and non-static variables	Interface has only static and final variables (constants)
<i>abstract</i> keyword is used to declare abstract class	<i>interface</i> keyword is used to declare interface
Abstract class can extend another class and implement multiple interfaces	Interface can extend another interface only
Abstract class can be extended using keyword " extends "	Interface can be implemented using keyword " implements "
Abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default
Example: <pre>public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre>public interface Drawable{ void draw(); }</pre>

Abstract Class vs Interface (cont'd)

■ Which should you use, abstract classes or interfaces?

➤ Consider using abstract classes if any of these statements apply to your situation:

- You want to share code among several **closely related classes**
- You expect that classes that extend your abstract class have **many common methods or fields**, or require access modifiers other than public (such as protected and private)
- You want to declare non-static or non-final fields

➤ Consider using interfaces if any of these statements apply to your situation:

- You expect that **unrelated classes** would implement your interface
- You want to **specify the behavior of a particular data type**, but **not concerned about who implements** its behavior
- You want to take advantage of multiple inheritance of type

Q&A

■ Next week (eClass video)

- Exception handling
- Java basic packages