06_3 Abstract Class

Object-Oriented Programming

Abstract Class

- Created by extracting the common characteristics (fields and methods) of concrete classes that can create objects.
- There can be one or more abstract methods in an abstract class.
- Abstract method has only the signature, no body.
- Abstract class cannot make an object.

Example) PaymentDemo (1/4)

```
abstract class Payment { // Abstract class: cannot make an object

    abstract void makePayment(double amount); // Abstract method (no body)

    void transactionDetails() { // Regular method
        System.out.println("Processing payment...");
    }
}

class CreditCardPayment extends Payment { // (Concrete) Subclass
    void makePayment(double amount) { // abstract method implementation
        System.out.println("Payment of $" + amount + " Credit Card.");
    }
}
```

Example) PaymentDemo (2/4)

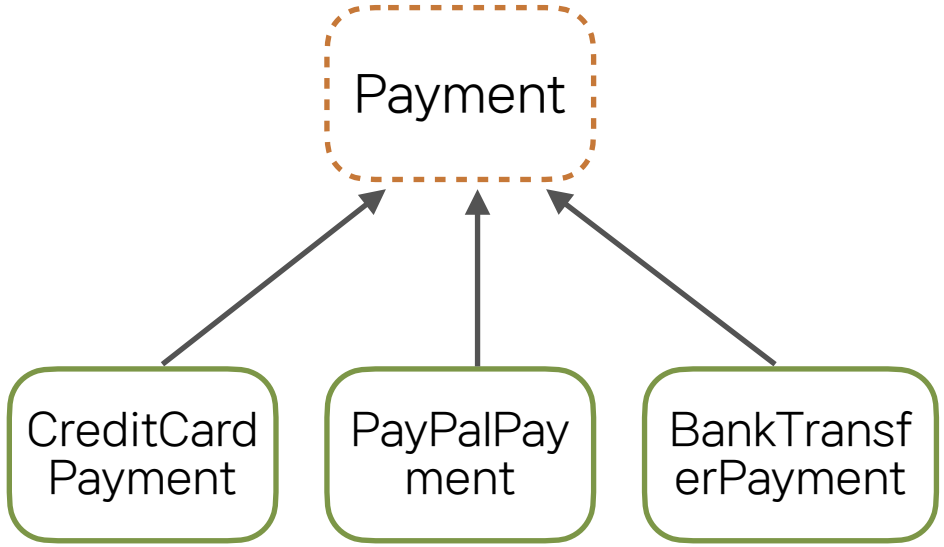
```
class PayPalPayment extends Payment { // Another (concrete) subclass
    // Providing implementation of abstract method
    void makePayment(double amount) {
        System.out.println("Payment of $" + amount + " PayPal.");
    }
}

class BankTransferPayment extends Payment { // concrete subclass
    // Providing implementation of abstract method
    void makePayment(double amount) {
        System.out.println("Payment of $" + amount+“ Bank Transfer.");
    }
}
```

Example) PaymentDemo (3/4)

```
public class PaymentDemo {
    public static void main(String[] args) {
        // Payment payment = new Payment(); //Error:no instance allowed

        Payment creditCardPayment = new CreditCardPayment();
        Payment payPalPayment = new PayPalPayment();
        Payment bankTransferPayment = new BankTransferPayment();

        creditCardPayment.transactionDetails();
        creditCardPayment.makePayment(100.50);

        payPalPayment.transactionDetails();
        payPalPayment.makePayment(250.75);

        bankTransferPayment.transactionDetails();
        bankTransferPayment.makePayment(400.00);
    }
}
```

Example) PaymentDemo (4/4)

```
OUTPUT:

Processing payment...
Payment of $100.5 Credit Card.
Processing payment...
Payment of $250.75 PayPal.
Processing payment...
Payment of $400.0 Bank Transfer.
```

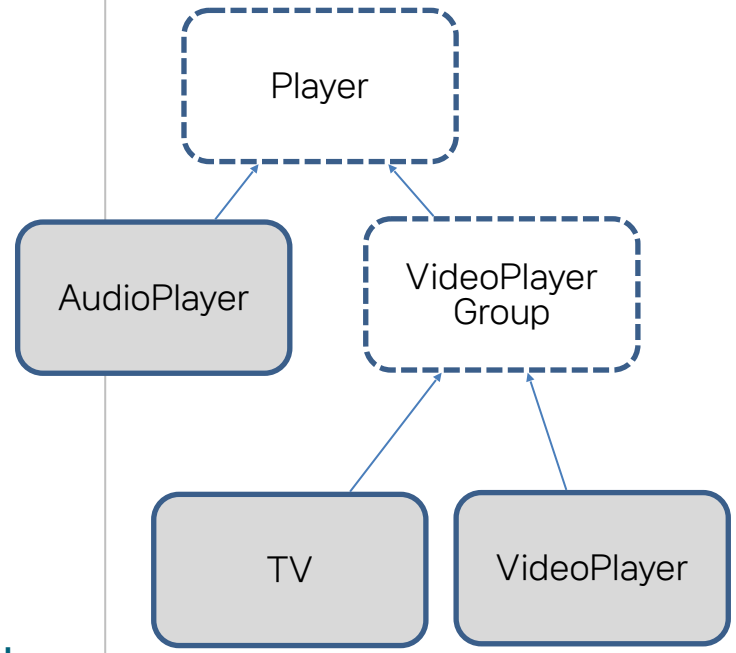


(Concrete) Subclass should implement abstract classes (1/2)

```
abstract class Player {
    int currentPos;

    Player() {
        currentPos = 0;
    }

    abstract void play(int pos);  // abstract method
    abstract void stop();         // abstract method

    void play() { // overloaded play(), not abstract
        play(currentPos);
    }
}
```

(Concrete) Subclass should implement abstract classes (2/2)

```
class AudioPlayer extends Player { // concrete class
    void play(int pos) {
        System.out.println("play audio from " + pos);
    }
    void stop() {
        System.out.println("stop audio");
    }
}

// another abstract class in the class hierarchy
abstract class VideoPlayerGroup extends Player {
    // doesn't implement play(int pos) and stop()
    abstract void diplayOn(); // another abstract method
}
```



TV and VideoPlayer should implement play(int pos), stop(), and displayOn()

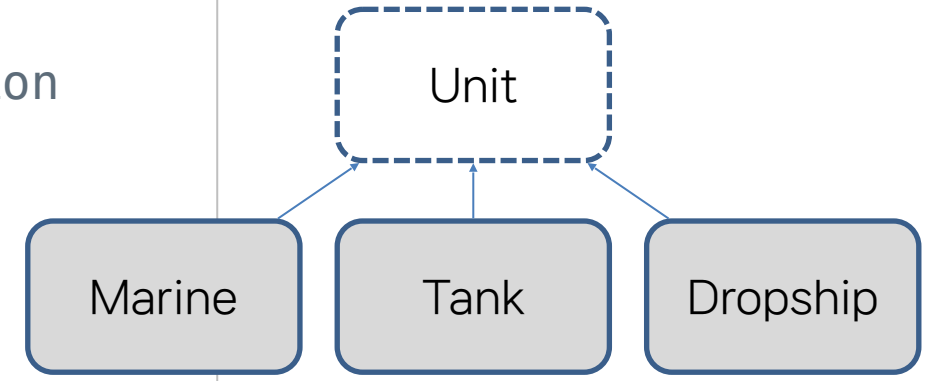
Designing Abstract Class (1/2)

```
class Marine {
    int x, y;
    void move(int x, int y) { ... } // move to (x,y)
    void stop() { ... } // stop at current position
    void stimPack() { ... } // using stimPack (무기 한 종류)
}
class Tank {
    int x, y;
    void move(int x, int y) { ... }
    void stop() {...}
    void changeMode() { } // change attack mode
}
class Dropship {
    int x, y;
    void move(int x, int y) { ... }
    void stop() { ... }
    void load() { ... }  // load the selected object
    void unload() { ... } // unload the selected object
}
```

- Extract common parts from existing classes to construct the abstract class

Designing Abstract Class (2/2)

```
abstract class Unit {
    int x, y;
    abstract void move(int x, int y);
    void stop() { ... } // stop at current position
}
class Marine extends Unit {
    void move(int x, int y) { ... }
    void stimPack() { ... } // using stimPack
}
class Tank extends Unit {
    void move(int x, int y) { ... }
    void changeMode() { } // change attack mode
}
class Dropship extends Unit {
    void move(int x, int y) { ... }
    void load() { ... }  // load the selected object
    void unload() { ... } // unload the selected object
}
```



Abstract Class Can be Used as Parameter (1/3)

```
abstract class Document { // Abstract class
    abstract void printContent(); // Abstract method
}

class PDFDocument extends Document { // Subclass
    void printContent() { // implementation of abstract method
        System.out.println("Printing PDF..");
    }
}

class WordDocument extends Document { // Subclass
    void printContent() { // implementation of abstract method
        System.out.println("Printing Word..");
    }
}
```

Abstract Class Can be Used as Parameter (2/3)

```
class Printer {
    void printDocument(Document doc) { // parameter: abstract class
        System.out.println("Starting..");
        doc.printContent();
        System.out.println(”Job completed");
    }
}
```

Abstract Class Can be Used as Parameter (3/3)

```
public class DocumentDemo {
    public static void main (String[] args) {
        // Create instances of subclasses
        Document pdf = new PDFDocument();
        Document word = new WordDocument();

        // Create Printer instance
        Printer printer = new Printer();

        // Use Printer to print documents
        printer.printDocument(pdf);
        // Output: Starting.. Printing PDF.. Job completed
        printer.printDocument(word);
        // Output: Starting.. Printing Word.. Job completed
    }
}
```