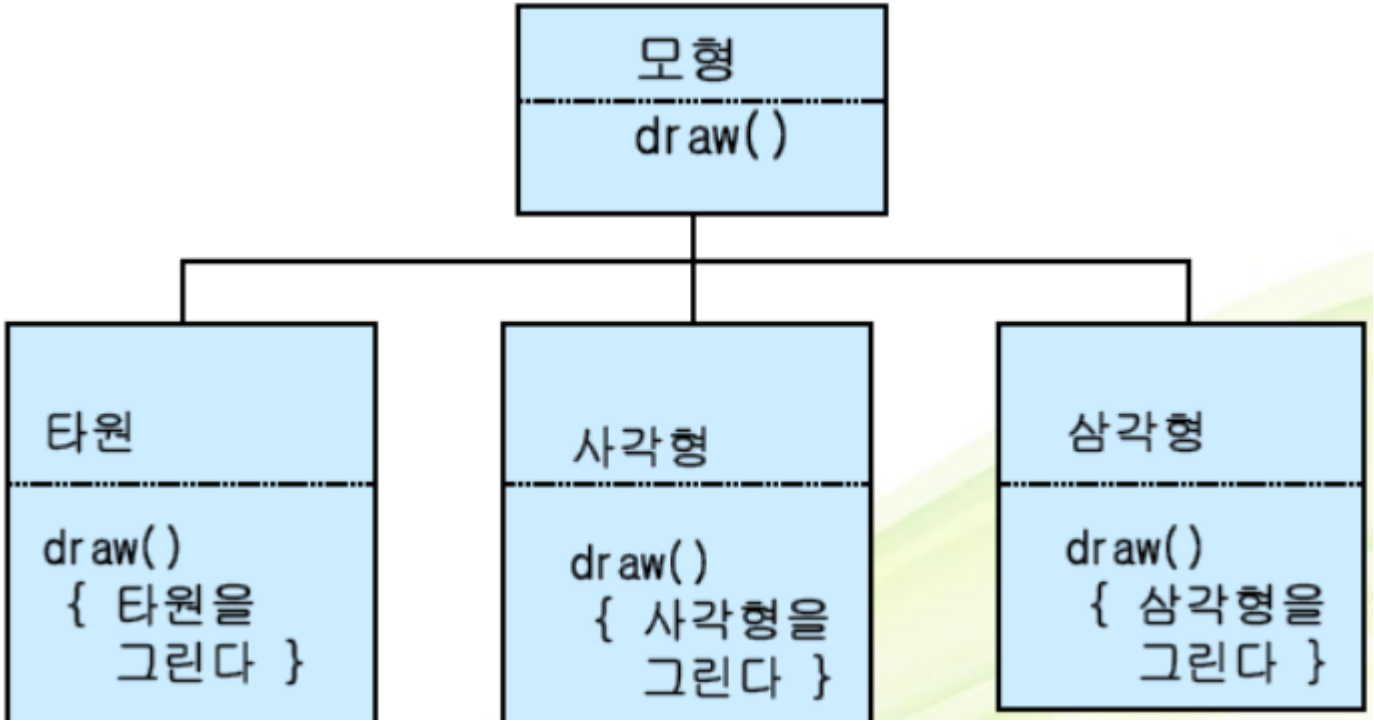
08. 다형성, 추상_Interface

강사 강태광

I] 다형성

[1] one interface, multiple implementation

- 하나의 인터페이스를 사용하여 다양한 구현 방법을 제공
- 하나의 클래스나 함수가 다양하게 동작



2]오버로딩(**overloading**) 과 오버라이딩(**overriding**)

1) 오버로딩(**overloading**)
 - 메소드 다중정의
2) 오버라이딩(**overriding**)
 - 메소드 재정의

1) 같은 클래스 내에서 만 오버로딩을 할 수 있다.
2) 같은 이름을 가진 메소드를 여러개 정의 하는 방법.
3) 규칙: * 메쏘드의 이름이 같아야한다.
  [1] 메소드 인자의 숫자가 다르거나
  [2] 메소드 인자의 타입이 달라야한다.
  [3] 메소드 리턴타입,접근지정자는 상관없다.

▶ 대표적인 예) 생성자

```
public ExOverLoading()
public ExOverLoading(int i)
public ExOverLoading(String str)
public ExOverLoading(int i, String str)
```

3] 오버로딩(**overloading**) – 예시문

```
class Test{
	Test(){
		System.out.println("인자가 없는 생성자 함수");
	}
	Test(int i){
		System.out.println("인자를 " +i+" 로 받은 생성자 함수");
	}
	Test(int i, int j) {
		System.out.println("인자를 " +i+" 와 "+j+" 로 받은 생성자 함수");
	}
	public static void main(String[] agrs){
		Test t = new Test(3,4);
	}
}
```

4] 오버라이딩(overriding)

- 조상클래스로부터 상속받은 메서드의 내용을 상속받는 클래스에 맞 게 변경하는 것

```
class Point {
      int x;
      int y;

      String getLocation() {
            return "x :" + x + ", y :"+ y;
      }
}

class Point3D extends Point {
      int z;
      String getLocation() {       // 오버라이딩
            return "x :" + x + ", y :"+ y + ", z :" + z;
      }
}
```

4] 오버라이딩(overriding) 조건

1. 선언부가 같아야 한다.(이름, 매개변수, 리턴타입)
2. 접근제어자를 좁은 범위로 변경할 수 없다.
   - 조상의 메서드가 protected라면, 범위가 같거나 넓은 protected나 public으로만 변경할 수 있다.
3. 조상클래스의 메서드보다 많은 수의 예외를 선언할 수 없다

```
class Parent {
    void parentMethod() throws IOException, SQLException {
        // ...
    }
}

class Child extends Parent {
    void parentMethod() throws IOException {
        //..
    }
}

class Child2 extends Parent {
    void parentMethod() throws Exception {
        //..
    }
}
```

4] 오버라이딩(overriding) 예시

OverridingTest1.java

```
class Da {
        void show(String str) {
       System.out.println("상위 클래스의 " + str);
        }
}
class Db extends Da {
       void show( ) {
           System.out.println("하위클래스의 메소드 내용");
       }
}
public class OverridingTest1 {
       public static void main(String args[]) {
             Db over = new Db();
             over.show("메소드 내용" );
             over.show();
       }
 }
```

5] 제한자(final)

1. 클래스 앞에 붙일경우
   상속금지
   ex> public final class Test{}
2. 멤버 메소드 앞에 붙일경우
   오버라이딩 금지
   ex> public final void print(){}
3. 멤버변수 앞에 붙일 경우--> 상수
   ex> public final int PORT_NUMBER=80;
   상수화된다. -> 변경금지

5] 추상메서드(abstract method)

1) 선언부만 있고 구현부(몸통, body)가 없는 메서드

```
/* 주석을 통해 어떤 기능을 수행할 목적으로 작성하였는지 설명한다. */
abstract 리턴타입 메서드이름();

Ex)
/* 지정된 위치(pos)에서 재생을 시작하는 기능이 수행되도록 작성한다.*/
abstract void play(int pos);
```

2) 꼭 필요하지만 자손마다 다르게 구현될 것으로 예상되는 경우에 사용
3) 추상클래스를 상속받는 자손클래스에서 추상메서드의 구현부를 완성

```
abstract class Player {
    ...
     abstract void play(int pos);      // 추상메서드
     abstract void stop();             // 추상메서드
    ...
}

class AudioPlayer extends Player {
    void play(int pos) { /* 내용 생략 */ }
    void stop() { /* 내용 생략 */ }
}

abstract class AbstractPlayer extends Player {
    void play(int pos) { /* 내용 생략 */ }
}
```

5] 추상클래스의 작성 예시

- 여러 클래스에 공통적으로 사용될 수 있는 추상클래스를 바로 작성하거나 기존클래스의 공통 부분을 뽑아서 추상클래스를 만든다

```
abstract class Unit {
    int x, y;
    abstract void move(int x, int y);
    void stop() {      /* 현재 위치에 정지 */ }
}

class Marine extends Unit {  // 보병
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank extends Unit {    // 탱크
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship extends Unit { // 수송선
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

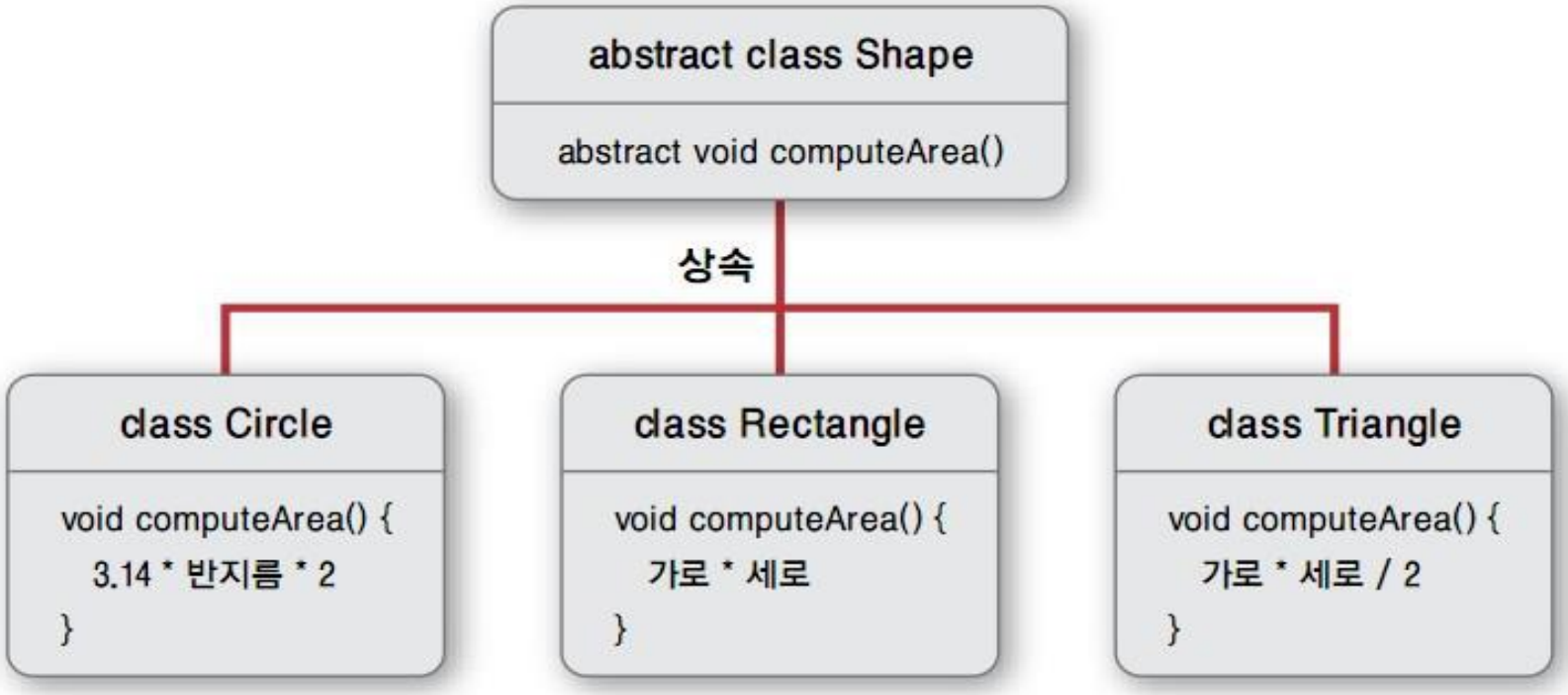
```
Unit[] group = new Unit[4];
group[0] = new Marine();
group[1] = new Tank();
group[2] = new Marine();
group[3] = new Dropship();

for(int i=0;i< group.length;i++) {
    group[i].move(100, 200);
}
```

추상메서드가 호출되는 것이 아니라 각 자손들에 실제로 구현된 move(int x, int y)가 호출된다.

5] 추상클래스(abstract class) I

[1] 추상클래스 개념
- **추상 메소드를 하나라도 가지고 있는 클래스를 추상 클래스**
- **추상 메소드 없이 변수들과 생성자만 선언된 추상클래스 생성가능**

[2] 추상클래스 특징
1) 추상 클래스를 상속받은 자식 클래스는 반드시 추상 메소드를 오버라이딩해야 함
2) 추상 클래스가 자식 클래스에 추상 메소드의 재정의를 **강요**
3) 추상 클래스는 객체로 생성할 수 없음
4) 추상 클래스에서 설계가 완료되면 자식 클래스에서 상속을 받아서 기능을 확장하는데 용이



6] 추상클래스(abstract class) 2

1) 추상 메소드
   - 추상 클래스 내에 정의되는 메소드로써 선언 부분만 있고 구현 부분이 없는 메소드
   - 하위 클래스는 상위 클래스에서 추상 메소드로 정의된 메소드를 재정의하여 사용
2) 추상 클래스와 추상 메소드
   - 내부 클래스에서 외부 클래스의 멤버들을 쉽게 접근할 수 있다.
   - 코드의 복잡성을 줄일 수 있다(캡슐화)

형식

```
abstract class 클래스이름 {
    ..........
    클래스에 기술할 수 있는 일반적인 멤버 변수와 메소드
    ..........
    abstract void 추상메소드이름();
}
```

추상 메소드는 선언 부분만 있다.
하위 클래스에서 오버라이딩하여 사용

6] 추상화 형식

1.정의: 하나이상의 추상 메소드가 정의되어있는 클래스

ex> public abstract class Test{
```
    // 추상 메소드();
    public abstarct int print(int i);
    // 일반 메소드();
    public void test(){
    }
}
```
⇒추상메소드: 메소드의 구현부분이 없고 원형(prototype) 만 존재하는 메소드

ex> public abstract int print(int i);

2. 추상클래스는 불완전한 추상 메소드를 가지므로 객체생성이 불가능하다. Test t=new Test();(X)

3. 추상클래스는 추상클래스를 상속받아서 추상 메소드를 구현(오버라이딩)하는 자식 클래스를 만들어 사용(객체생성)해야 한다

▶ class 를 추상화 시키는 형식

abstract class class-name{.....} ☞ abstract class Person{.....}

▶ method를 추상화 시키는 형식

abstract 접근 제어자 반환 자료형 method-name(인자);

☞ abstract public void setName(String name);

☞ abstract public String getName();

☞ abstract public void setAge(int age);

7] 인터페이스(interface) I

1) 일종의 추상클래스. 추상클래스(미완성 설계도)보다 추상화 정도가 높다.
2) 실제 구현된 것이 전혀 없는 기본 설계도.(알맹이 없는 껍데기)
3) 추상메서드와 상수만을 멤버로 가질 수 있다.
4) 인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용된다
5) 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 데 사용된다
6) class'대신 'interface'를 사용한다는 것 외에는 클래스 작성과 동일하다

```
interface 인터페이스이름 {
    public static final 타입 상수이름 = 값;
    public abstract 메서드이름(매개변수목록);
}
```

7] 인터페이스(interface) 2

구성요소(멤버)는 추상메서드와 상수만 가능하다.

- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있다.

```
interface PlayingCard {
     public static final int SPADE = 4;
     final int DIAMOND = 3;      // public static final int DIAMOND = 3;
     static int HEART = 2;       // public static final int HEART = 2;
     int CLOVER = 1;             // public static final int CLOVER = 1;

     public abstract String getCardNumber();
     String getCardKind();  // public abstract String getCardKind();
}
```

7] 인터페이스의 상속

-인터페이스도 클래스처럼 상속이 가능하다.(클래스와 달리 다중상속 허용)
-인터페이스는 Object클래스와 같은 최고 조상이 없다.

```
interface Movable {
	/** 지정된 위치(x, y)로 이동하는 기능의 메서드 */
	void move(int x, int y);
}

interface Attackable {
	/** 지정된 대상(u)을 공격하는 기능의 메서드 */
	void attack(Unit u);
}

interface Fightable extends Movable, Attackable { }
```

7] 인터페이스의 구현

1) 인터페이스를 구현하는 것은 클래스를 상속받는 것과 같다. 다만, 'extends' 대신 'implements'를 사용한다.

```
class 클래스이름 implements 인터페이스이름 {
     // 인터페이스에 정의된 추상메서드를 구현해야한다.
}
```

2) 인터페이스에 정의된 추상메서드를 완성해야 한다

```
class Fighter implements Fightable {
     public void move() { /* 내용 생략*/ }
     public void attack() { /* 내용 생략*/ }
}

interface Fightable {
     void move(int x, int y);
     void attack(Unit u);
}
```

3) 상속과 구현이 동시에 가능하다

```
class Fighter extends Unit implements Fightable {
     public void move(int x, int y) { /* 내용 생략 */}
     public void attack(Unit u) { /* 내용 생략 */}
}
```

7] 인터페이스의 특징

1. 클래스가가진 모든 메쏘드가 추상메쏘드 형식임
2. 다중상속의 효과를 냄 (클래스는 불가능, 인터페이스는가능)

-형식 * class keyword 대신에 interface 라는 keyword를사용

* 추상메쏘드앞에 abstract 를 붙히지않는다.

ex>public interface Test{

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

}

- 사용 : 상속(implements)받아서 재정의(구현)한후 사용한다.

ex> public class TestImpl implements Test{

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

}

interface : 연결, 양식

interface의 포함 멤버

public static final 멤버 필드

7] 인터페이스 예제 사용예시

```
interface AA {
            int aa();
}
class BB implements AA { //오버라이딩
      public int aa() {
      return 100;
      }
}
class CC implements AA { //오버라이딩
      public int aa() {
      return 200;
      }
}
class InterfaceTest2 {
      public static void main(String args[]) {
      System.out.println(new BB().aa());
      System.out.println(new CC().aa());
      }
}
```

7] 인터페이스의 장점

1. 개발시간을 단축시킬 수 있다.
   - 일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문.
   - 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행
2. 표준화가 가능하다.
   - 프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능.
3. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.
   - 서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 매핑.
4. 독립적인 프로그래밍이 가능하다.
   - 인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제 구현에 독립적인 프로그램을 작성하는 것이 가능
   - 클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능

8] 추상 Class와 비교

- 추상 클래스와 유사(상수와 추상 메소드만 구성)
- 추상 클래스보다 더 완벽한 추상화 제공
- 다중 상속(Multiple Inheritance) 지원

구분	추상클래스	인터페이스

선언	abstract class 클래스명{
	변수;
	메소드(){ .....}
	abstract 메소드( );
	}
		interface 인터페이스명{
		상수;
		메소드( ); // 추상 메소드
		}

상속	class Sub extends Super{
	메소드 재정의(Overriding)
	}
		class Sub implements Interface1, Interface2{
		메소드 재정의(Overriding)
		}

장점	프레임 제공	프레임 제공, 다중 상속

8.1 인터페이스의 이해

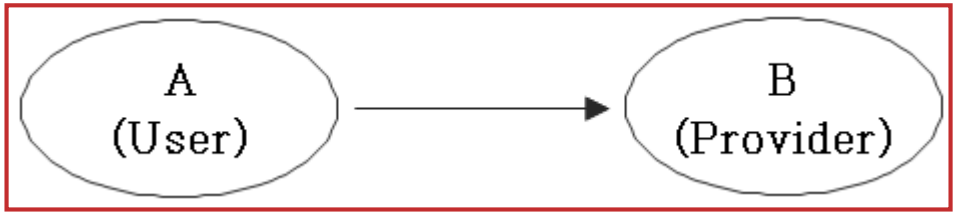
- 두 대상(객체) 간의 '연결, 대화, 소통'을 돕는 '중간 역할.
- 선언(설계)와 구현을 분리시키는 것을 가능하게 한다.

```
class B {
   public void method() {
      System.out.println("methodInB");
   }
}
```

→

```
interface I {
   public void method();
}

class B implements I {
   public void method() {
      System.out.println("methodInB");
   }
}
```

▶ 인터페이스를 이해하려면 먼저 두 가지를 기억하자.
- 클래스를 사용하는 쪽(User)과 클래스를 제공하는 쪽(Provider)이 있다.
- 메서드를 사용(호출)하는 쪽(User)에서는 사용하려는 메서드(Provider)의 선언부만 알면 된다.



9] 인터페이스의 Java8

1. Interface 에서 JAVA8 버전이후 추가 사항

▶ **default method 추가**
- 문제점 : 인터페이스에 추상메서드를 추가하게 되면 모든 구현체에 구현을 해야한다.
- 해결 방안 : 인터페이스에 default method를 사용하면 추가 변경을 막을 수 있다.

▶ **정적(static) Method.**
- 호환성 때문에 강제성 없음
- 필요시 적용 가능
- 사용시 일반적인 정적 method 호출처럼 사용 가능