Ch8
추상화 클래스와 인터페이스

객체지향프로그래밍(기본) 2019

경상대학교 항공우주및소프트웨어공학전공

8.1 추상화 메소드

추상화 메소드는 선언은 되어있지만 구현이 되지 않는 메소드를 의미한다.

추상 메소드를 선언하기 위해 abstract 키워드를 반환형 앞에 선언하며 추상 메소드를 사용하기 위해서는 반드시 오버라이드해서 구현해야 한다.

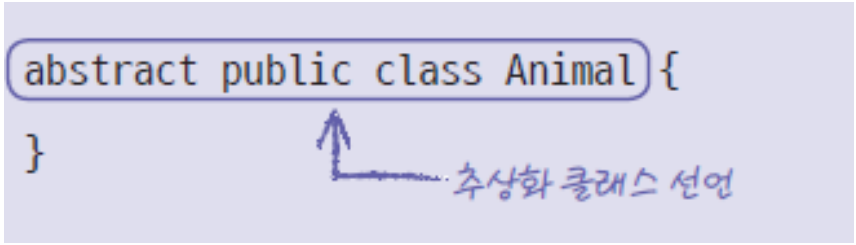
play 메소드를 추상화 클래스로 선언하는 경우 아래와 같이 abstract 키워드를 선언하고 메소드의 몸통은 구현하지 않고 세미콜론으로 끝낸다.

```
public abstract void play();
```

8.2 추상화 클래스

추상클래스(Abstract Class)는 개념적으로 정의되어 있고 상속을 통해 자식클래스의 형태를 잡아주는 역할을 한다.

추상 클래스를 만드는 방법은 class 키워드 앞에 abstract 키워드를 선언 한다.



추상 클래스는 객체를 만들 수 없다.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    Cannot instantiate the type Animal
```

추상화 메소드와 추상화 클래스

추상화 메소드를 가진 클래스는 추상화 클래스로 선언해야 한다. 일반 클래스에 추상화 메소드를 선언하면 문제가 발생한다.

```
public class Animal {
  String name;

  public void setName(String name){
    this.name = name;
  }

  public abstract void sound();
}
```

추상화 메소드 선언

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    The type Animal must be an abstract class to define abstract methods
    The abstract method sound in type Animal can only be defined by an
abstract class
```

추상화 클래스와 상속

추상화 클래스를 이용하여 객체를 만들려면 추상화 클래스를 상속 받는 자식 클래스를 이용하여 객체를 생성 할 수 있다.

추상 메소드가 있는 추상클래스를 일반클래스가 상속 받을 경우 추상 메소드를 오버라이드 하여 메소드를 구현해야 객체의 생성이 가능하다.

```
abstract public class Animal {
  String name;

  public void setName(String name){
    this.name = name;
  }

  public abstract void sound();
}
```

```
public class Dog extends Animal{

  public void sound() {  <-- 추상화 메소드를 오버라이드 함
    System.out.println("멍멍");
  }
}
```

```
public class Cat extends Animal{

  public void sound() {  <-- 추상화 메소드를 오버라이드 함
    System.out.println("야옹");
  }
}
```

추상화 클래스와 객체배열

추상 클래스는 객체 배열 선언을 통하여 자식 클래스의 다형성을 이용 할 수 있다.

```
Animal ani[] = new Animal[2];
```
← 추상화 클래스로 객체 배열 선언

```
ani[0] = new Dog();
```
← Animal 로 선언된 객체 배열의 레퍼런스 변수에 Dog 객체 생성하여 가르키도록 연결

```
ani[1] = new Cat();
```
← Animal 로 선언된 객체 배열의 레퍼런스 변수에 Cat 객체 생성하여 가르키도록 연결

```
for (int i = 0; i < 2; i++)
  ani[i].sound();
```

결과

```
멍멍
야옹
```

8.3 인터페이스

인터페이스(Interface)는 약속이다. 인터페이스는 인스턴스 변수는 가지지 않으며 순수 추상 메소드로 구성되어 있다.

인터페이스를 선언 할 때 interface 키워드로 선언 한다.

인터페이스의 약속을 준수하도록 하여 지정된 형태의 메소드를 생성하도록 한다.

```
interface 인터페이스명{
}
```

인터페이스 선언 시

```
class 클래스명  implements 인터페이스명{
}
```

클래스에서 인터페이스를 사용할 경우

인터페이스 구현

인터페이스를 구현하라고 선언된 클래스는 implements 키워드를 사용 하며 인터페이스를 구현하라고 정의된 클래스는 인터페이스에 정의된 추상 메소드를 구현해야 한다.

```
public interface Pet{
  public abstract void play();
}
```

```
public class Dog extends implements Pet {

 public void play() { <-- Pet 인터페이스의 추상화 메소드 구현
    System.out.println("공 물어오기");
 }
}
```

다중 인터페이스

자바는 다중 상속을 지하지 않는다. 하지만 인터페이스를 사용하면 다중 상속의 효과를 누릴 수 있다.

자바에서 자식 클래스는 하나의 부모 클래스만을 가질 수 있지만 여러 개의 인터페이스를 가지는 것은 허용한다.

```
public interface Pet{
  public abstract void play();
}
```

```
public interface HouseAnimal{
  public abstract void space();
}
```

```
public class Dog extends  implements Pet, HouseAnimal{

  public void play() {
    System.out.println("공 물어오기");
  }

  public void space() {
    System.out.println("거실");
  }
}
```

인터페이스의 상속

클래스와 인터페이스 간의 상속은 불가능 하지만 인터페이스와 인터페이스 간의 상속은 가능 하다.

```
public interface HouseAnimal{
  public abstract void space();
}
```

```
public interface Pet extends HouseAnimal{
  public abstract void play();
}
```