7장. 상속의 다형성과 추상 클래스

Contents

7절. 타입변환과 다형성(polymorphism)

❖ 다형성 (多形性, Polymorphism)

- 같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질
  - 부모 타입에는 모든 자식 객체가 대입 가능
    - 자식 타입은 부모 타입으로 자동 타입 변환
  - 효과: 객체 부품화 가능

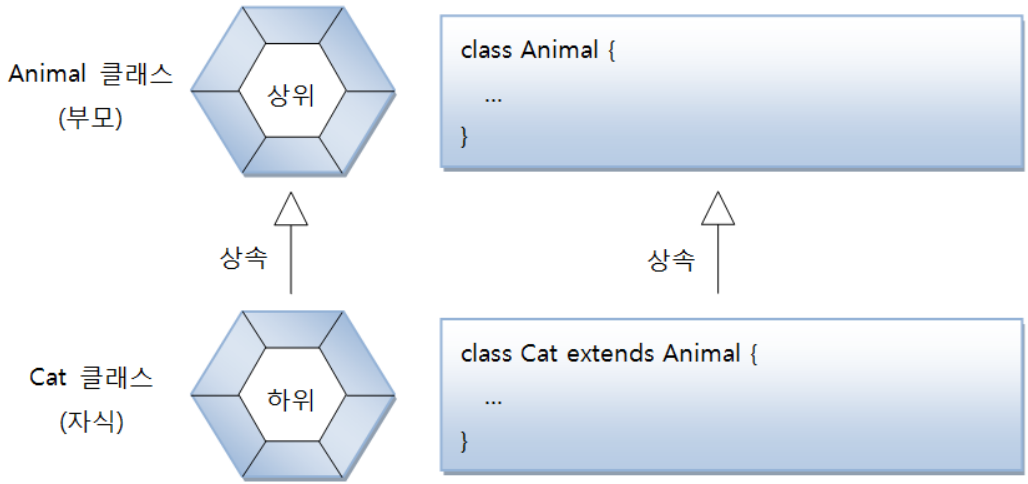
7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것

자동 타입 변환

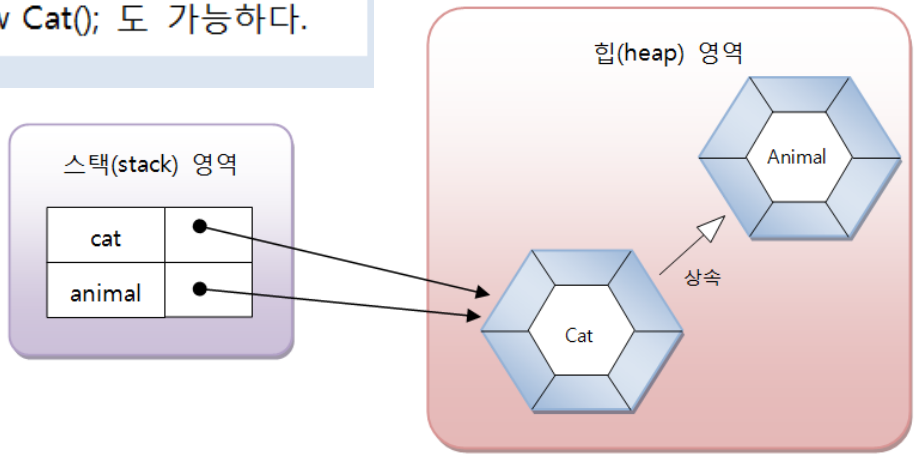
부모클래스 변수 = 자식클래스타입;




```
Cat cat = new Cat();
Animal animal = cat;
```

} Animal animal = new Cat(); 도 가능하다.

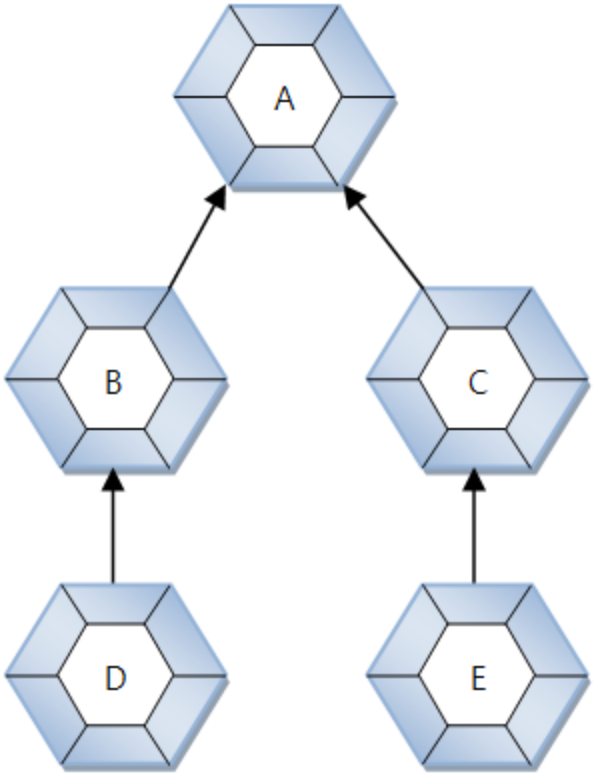
```
cat == animal    //true
```



7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 바로 위의 부모가 아니더라도 상속 계층의 상위면 자동 타입 변환 가능
  - 변환 후에는 부모 클래스 멤버만 접근 가능




```
B b = new B();
C c = new C();
D d = new D();
E e = new E();
```

```
A a1 = b;   (가능)
A a2 = c;   (가능)
A a3 = d;   (가능)
A a4 = e;   (가능)

B b1 = d;   (가능)
C c1 = e;   (가능)

B b3 = e;   (불가능)
C c2 = d;   (불가능)
```

7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것

Parent.java

```
1  package week10;
2
3  public class Parent {
4
5      public void method1() {
6          System.out.println("Parent-method1");
7      }
8
9      public void method2() {
10         System.out.println("Parent-method2");
11     }
12 }
```

Child.java

```
1  package week10;
2
3  public class Child extends Parent {
4      // Parent 클래스에 정의된 method2()를 재정의한다
5      @Override
6      public void method2() {
7          System.out.println("Child-method2");
8      }
9
10     // method3()는 Child 클래스에만 정의된 메소드이다
11     public void method3() {
12         System.out.println("Child-method3");
13     }
14 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것

Parent.java | Child.java | ChildEx.java

```java
package week10;

public class ChildEx {
    public static void main(String[] args) {
        // Parent 클래스를 상속받은 Child 클래스를 이용하여 객체를 생성한다
        Child  child = new Child();

        // 자식 클래스로 생성된 객체는 부모 클래스로 생성된 객체에 대입할 수 있다
        // 이 때, 자동 타입 변환된다
        Parent parent = child;

        parent.method1();

        // 자동 타입 변환이 적용된 경우 부모 클래스 객체라도 자식 클래스의 재정의된 메소드가 호출된다
        parent.method2();

        // 자식 객체가 부모 객체로 자동 타입 변환되면 자식 객체에만 정의된 메소드는 호출할 수 없다
        //parent.method3();
    }
}
```

7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것

Parent.java | Child.java | ChildEx.java

```java
package week10;

public class ChildEx {
    public static void main(String[] args) {
        // Parent 클래스를 상속받은 Child 클래스를 이용하여 객체를 생성한다
        Child  child = new Child();

        // 자식 클래스로 생성된 객체는 부모 클래스로 생성된 객체에 대입할 수 있다
        // 이 때, 자동 타입 변환된다
        Parent parent = child;

        parent.method1();

        // 자동 타입 변환이 적용된 경우 부모 클래스 객체라도 자식 클래스의 재정의된 메소드가 호출된다
        parent.method2();

        // 자식 객체가 부모 객체로 자동 타입 변환되면 자식 객체에만 정의된 메소드는 호출할 수 없다
        //parent.method3();
    }
}
```

Console

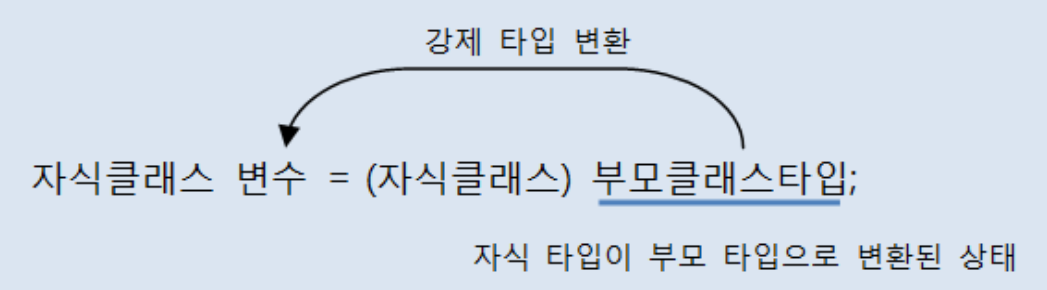
<terminated> ChildEx [Java Application]

```
Parent-method1
Child-method2
```

7절. 타입변환과 다형성(polymorphism)

❖ 강제 타입 변환(Casting)

- 부모 타입을 자식 타입으로 변환하는 것



- 조건
  - 자식 타입을 부모 타입으로 **자동 변환** 후, **다시 자식 타입으로 변환할 때**

- 강제 타입 변환이 필요한 경우
  - 자식 타입이 부모 타입으로 자동 변환
    - 부모 타입에 선언된 필드와 메소드만 사용 가능
  - 자식 타입에 선언된 필드와 메소드를 다시 사용해야 할 경우

7절. 타입변환과 다형성(polymorphism)

❖ 강제 타입 변환(Casting)

```java
public class Parent {
    String field;
    public void method1() {
        System.out.println("Parent-method1");
    }
    public void method2() {
        System.out.println("Parent-method2");
    }
}
```

```java
public class Child extends Parent {
    String field2;
    @Override
    public void method2() {
        System.out.println("Child-method2()");
    }
    public void method3() {
        System.out.println("Child-method3()");
    }
}
```

7절. 타입변환과 다형성(polymorphism)

❖ 강제 타입 변환(Casting)

```
public class ChildEx {
    public static void main(String[] args) {
        Child child = new Child();
        Parent parent = child;
        parent.field = "aaa";
        parent.method1();
        parent.method2();

        //자식 객체가 부모 타입으로 자동 변환=>부모 타입에 선언된 필드와 메소드만 사용
        //parent.field2 = "bbb";
        //parent.method3();

        //자식 객체로 강제 타입 변환 => 자식 타입에 선언된 필드와 메소드 사용 가능
        //자식 객체로 자동 타입 변환된 부모 객체만 자식 객체로 강제 타입 변환이 가능
        child = (Child)parent;
        child.field2 = "bbb";
        child.method3();
    }
}
```

7절. 타입변환과 다형성(polymorphism)

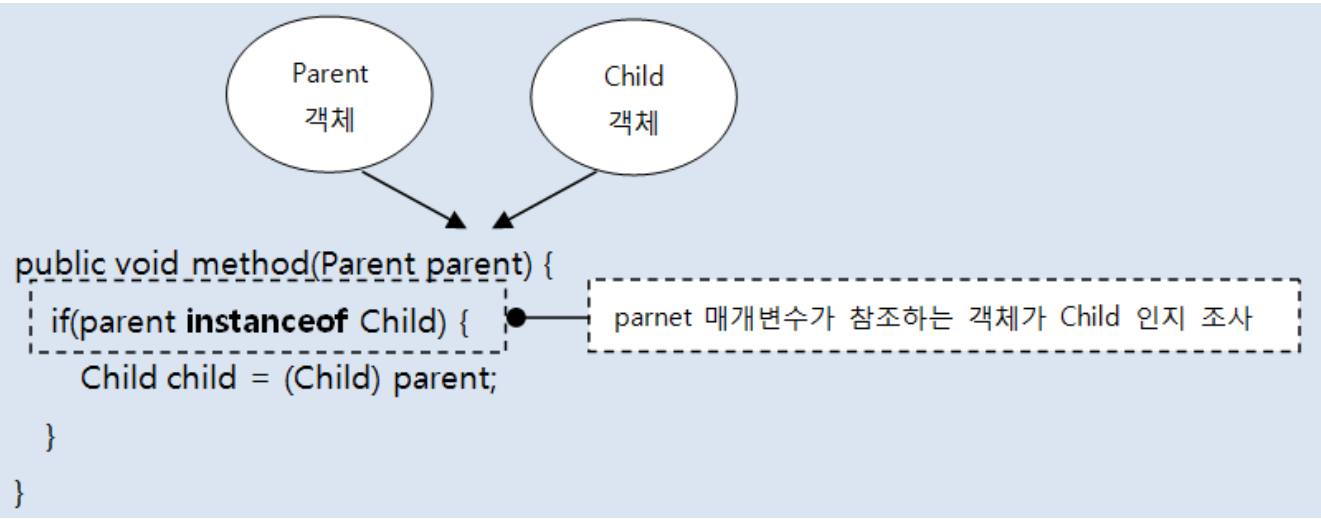
❖ 객체 타입 확인(instanceof)

- 부모 타입이면 모두 자식 타입으로 강제 타입 변환할 수 있는 것 아님
  - ClassCastException 예외 발생 가능

```
Parent parent = new Parent();
Child child = (Child) parent;      //강제 타입 변환을 할 수 없다.
```

- 먼저 자식 타입인지 확인 후 강제 타입 실행해야 함

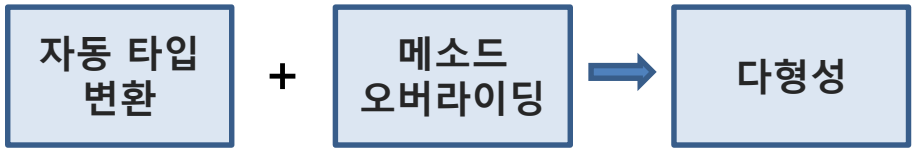
```
boolean result = 좌항(객체) instanceof 우항(타입)
```



7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

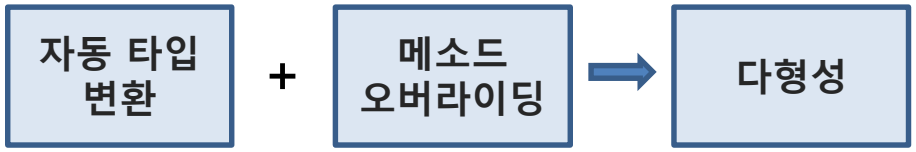
- 다형성 : 동일한 타입을 사용하지만 다양한 결과가 나오는 성질
- 다형성을 구현하는 기술적 방법
  - 부모 클래스 상속
  - 메소드 재정의(오버라이딩)
  - 부모 타입으로 자동 타입 변환

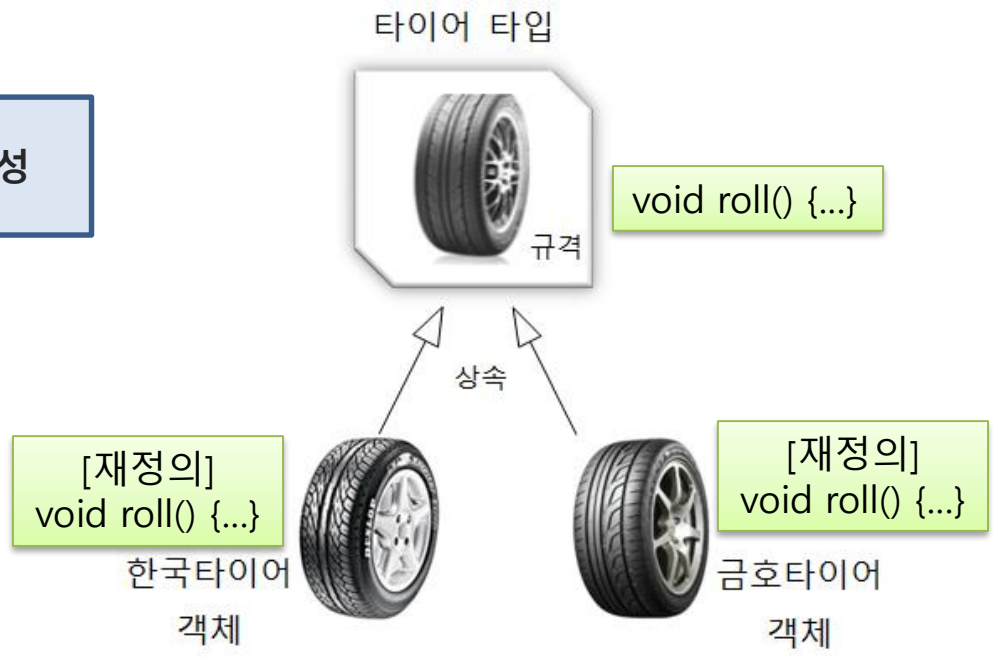


7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

- 다형성 : 동일한 타입을 사용하지만 다양한 결과가 나오는 성질
- 다형성을 구현하는 기술적 방법
  - 부모 클래스 상속
  - 메소드 재정의(오버라이딩)
  - 부모 타입으로 자동 타입 변환





7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

- 필드가 클래스 타입인 경우

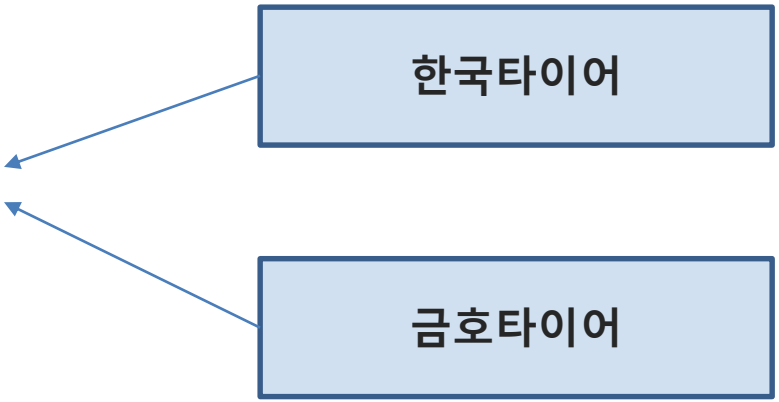
- Car 클래스
  - 속성 : Tire
  - 동작 : run()

- Tire 클래스
  - 속성 : 최대회전수
  - 동작 : roll()

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

- Car 클래스
  - 속성 : Tire
  - 동작 : run()
- Tire 클래스
  - 속성 : 최대회전수
  - 동작 : roll()



7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
public class Tire {
    private int maxRotation;

    public void roll() {
        System.out.println("타이어 회전");
    }
}
```

```
public class Car {
    public Tire tire;

    public void run() {
        tire.roll();
    }
}
```

```
public class HankookTire extends Tire {
    @Override
    public void roll() {
        System.out.println("한국 타이어 회전");
    }
}
```

```
public class KumhoTire extends Tire {
    @Override
    public void roll() {
        System.out.println("금호 타이어 회전");
    }
}
```

7절. 타입변환과 다형성(polymorphism)

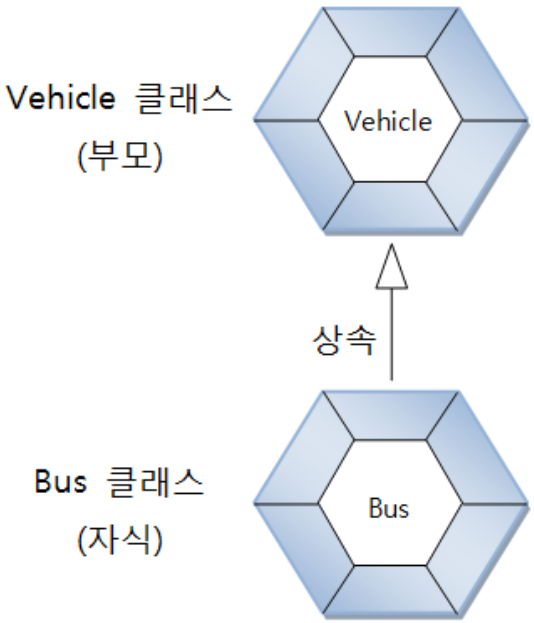
❖ 필드의 다형성

```
public class CarEx {
    public static void main(String[] args) {
        Car car = new Car();
        car.tire = new Tire();
        car.run();

        car.tire = new HankookTire();
        car.run();

        car.tire = new KumhoTire();
        car.run();
    }
}
```

7절. 타입변환과 다형성(polymorphism)

❖ 매개변수의 다형성

- 매개변수가 클래스 타입일 경우
  - 해당 클래스의 객체 대입이 원칙이나 자식 객체 대입하는 것도 허용
    - 자동 타입 변환
    - 매개변수의 다형성



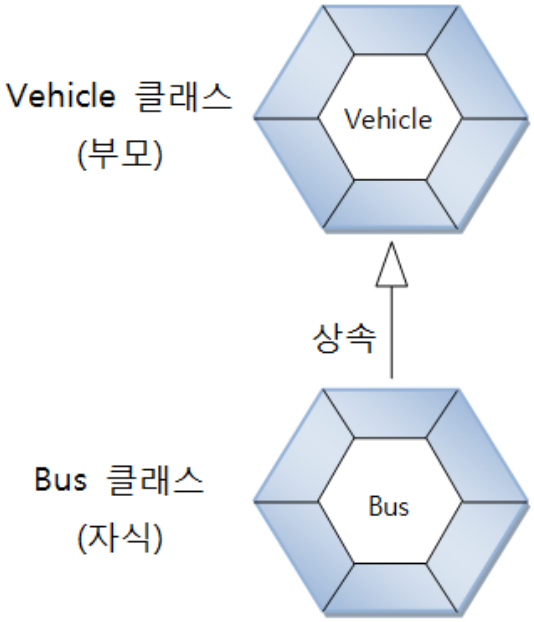

```
class Vehicle {
    public void run() {

    }
}
```

```
class Bus extends Vehicle {
    @Override
    public void run() {

    }
}
```

7절. 타입변환과 다형성(polymorphism)

❖ 매개변수의 다형성

- 매개변수가 클래스 타입일 경우
  - 해당 클래스의 객체 대입이 원칙이나 자식 객체 대입하는 것도 허용
    - 자동 타입 변환
    - 매개변수의 다형성




```
class Driver {
    void drive(Vehicle vehicle) {
        vehicle.run();
    }
}
```

```
Driver driver = new Dirver();

Bus bus = new Bus();

driver.drive( bus );
```

자동 타입 변환 발생
Vehicle vehicle = bus;

```
void drive(Vehicle vehicle) {
    vehicle.run();
}
```

자식 객체

자식 객체가 재정의한 run() 메소드 실행

7절. 타입변환과 다형성(polymorphism)

❖ 매개변수의 다형성

```java
public class Vehicle {
    public void run() {
        System.out.println("차량이 달립니다.");
    }
}
```

```java
public class Bus extends Vehicle {
    @Override
    public void run() {
        System.out.println("버스가 달립니다.");
    }
}
```

```java
public class Taxi extends Vehicle {
    @Override
    public void run() {
        System.out.println("택시가 달립니다.");
    }
}
```

```java
public class Driver {
    public void drive(Vehicle v) {
        v.run();
    }
}
```

```java
public class DriverEx {
    public static void main(String[] args) {
        Driver d = new Driver();
        Vehicle v = new Vehicle();
        d.drive(v);

        d.drive(new Bus());
        d.drive(new Taxi());
    }
}
```

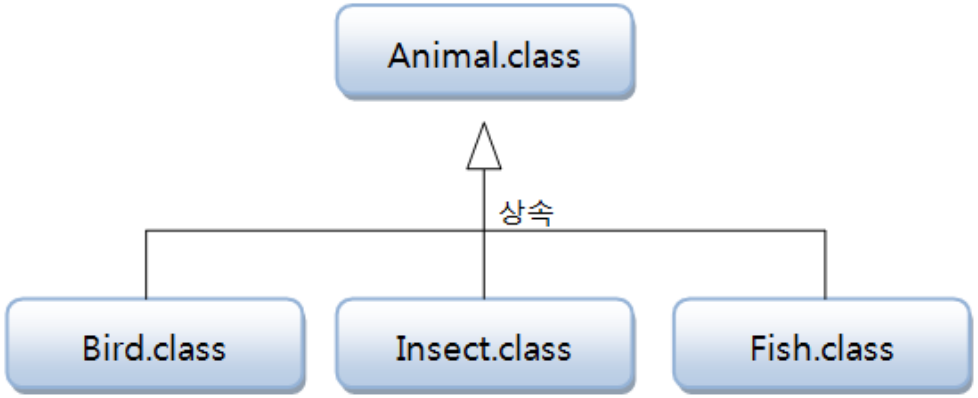
8절. 추상 클래스(Abstract Class)

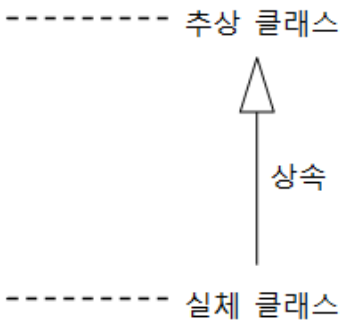
❖ 추상 클래스 개념

- 추상(abstract)
  - 실체들 간에 공통되는 특성을 추출한 것
    - 예1: 새, 곤충, 물고기→ 동물 (추상)
    - 예2: 삼성, 현대, LG → 회사 (추상)

- 추상 클래스(abstract class)
  - 실체 클래스들의 공통되는 필드와 메소드를 정의한 클래스
  - 추상 클래스는 실체 클래스의 부모 클래스 역할 (상속 관계, 단독 객체 X)

*실체 클래스: 객체를 만들어 사용할 수 있는 클래스





8절. 추상 클래스(Abstract Class)

❖ 추상 클래스의 용도

- 실체 클래스의 공통된 필드와 메소드의 이름을 통일할 목적
  - 실체 클래스를 설계하는 설계자가 여러 사람일 경우,
  - 실체 클래스마다 필드와 메소드가 제각기 다른 이름을 가질 수 있음
- 실체 클래스를 작성할 때 시간 절약
  - 실체 클래스는 추가적인 필드와 메소드만 선언
- 실체 클래스 설계 규격을 만들고자 할 때
  - 실체 클래스가 가져야 할 필드와 메소드를 추상 클래스에 미리 정의
  - 실체 클래스는 추상 클래스를 무조건 상속 받아 작성

8절. 추상 클래스(Abstract Class)

❖ 추상 클래스 선언

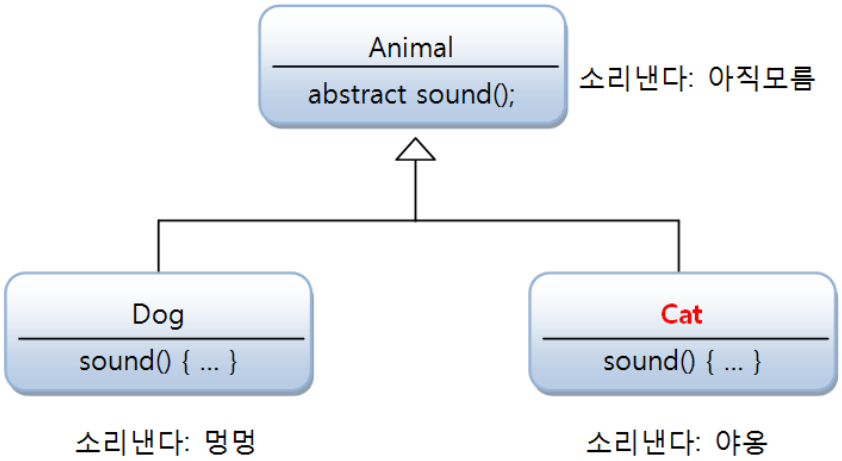
- 클래스 선언에 abstract 키워드 사용
  - New 연산자로 객체 생성하지 못하고 상속 통해 자식 클래스만 생성 가능

```
public abstract class 클래스 {
   //필드
   //생성자
   //메소드
}
```

8절. 추상 클래스(Abstract Class)

❖ 추상 메소드와 오버라이딩(재정의)

- 메소드 이름 동일하지만, 실행 내용이 실체 클래스마다 다른 메소드
- 예: 동물은 소리를 낸다. 하지만 실체 동물들의 소리는 제각기 다르다.
- 구현 방법
  - 추상 클래스에는 메소드의 선언부만 작성 (추상 메소드)
  - 실체 클래스에서 메소드의 실행 내용 작성(오버라이딩(Overriding))




```
public abstract class Animal {
    public abstract void sound();
}
```

```
public class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("멍멍");
    }
}
```

```
public class Cat extends Animal {
    @Override
    public void sound() {
        System.out.println("야옹");
    }
}
```

8절. 추상 클래스(Abstract Class)

Animal 클래스

Animal.java

```java
package week10;

public abstract class Animal {
    public String kind;

    public void breathe() {
        System.out.println("숨을 쉽니다.");
    }
    public abstract void sound();
}
```

Dog 클래스

AnimalEx.java

```java
package week10;

public class Dog extends Animal {

    public Dog() {
        this.kind = "포유류";
    }

    @Override
    public void sound() {
        System.out.println("멍멍");
    }

    public void dogLife() {
        System.out.println("Dog 평균 수명은 약 15~20년입니다.");
    }
}
```

8절. 추상 클래스(Abstract Class)

Animal.java | Dog.java | Cat.java | AnimalEx.java

```
1  package week10;
2
3  public class Cat extends Animal{
4
5      public Cat() {
6          this.kind = "포유류";
7      }
8      @Override
9      public void sound() {
10         System.out.println("야옹");
11     }
12
13     public void catLife() {
14         System.out.println("Cat 평균 수명은 약 12~15년입니다.");
15     }
16 }
```

Cat 클래스

8절. 추상 클래스(Abstract Class)

Animal.java | Dog.java | Cat.java | AnimalEx.java

```
1  package week10;
2
3  public class AnimalEx {
4      public static void main(String[] args) {
5          Dog dog = new Dog();
6          Cat cat = new Cat();
7          dog.sound();
8          cat.sound();
9          System.out.println("--------------------");
10
11         Animal animal = null;
12         animal = new Dog();
13         animal.sound();
14         animal = new Cat();
15         animal.sound();
16         System.out.println("--------------------");
17
18     }
19 }
```

AnimalEx 클래스

8절. 추상 클래스(Abstract Class)

Animal.java | Dog.java | Cat.java | AnimalEx.java

```
1  package week10;
2
3  public class AnimalEx {
4      public static void main(String[] args) {
5          Dog dog = new Dog();
6          Cat cat = new Cat();
7          dog.sound();
8          cat.sound();
9          System.out.println("--------------------");
10
11         Animal animal = null;
12         animal = new Dog();
13         animal.sound();
14         animal = new Cat();
15         animal.sound();
16         System.out.println("--------------------");
17
18     }
19 }
```

AnimalEx 클래스

Console

<terminated> AnimalEx [Java Application] C:

```
멍멍
야옹
--------------------
멍멍
야옹
--------------------
```

8절. 추상 클래스(Abstract Class)

Animal.java | Dog.java | Cat.java | AnimalEx.java | AnimalEx2.java

```
14          animal.sound();
15          System.out.println("--------------------");
16
17          animalSound(new Dog());
18          animalSound(new Cat());
19      }
20
21      private static void animalSound(Animal animal) {
22          animal.sound();
23          animal.breathe();
24
25          if (animal instanceof Dog) {
26              System.out.println("Dog 객체로 변환 가능");
27              Dog dog = (Dog)animal;
28              dog.dogLife();
29          }
30          else {
31              System.out.println("Cat 객체로 변환 가능");
32              Cat cat = (Cat)animal;
33              cat.catLife();
34          }
35          System.out.println("--------------------");
36      }
37  }
```

AnimalEx 클래스

8절. 추상 클래스(Abstract Class)

Animal.java | Dog.java | Cat.java | AnimalEx.java

```
14          animal.sound();
15          System.out.println("------------------");
16
17          animalSound(new Dog());
18          animalSound(new Cat());
19      }
20
21      private static void animalSound(Animal animal)
22          animal.sound();
23          animal.breathe();
24
25          if (animal instanceof Dog) {
26              System.out.println("Dog 객체로 변환 가능");
27              Dog dog = (Dog)animal;
28              dog.dogLife();
29          }
30          else {
31              System.out.println("Cat 객체로 변환 가능");
32              Cat cat = (Cat)animal;
33              cat.catLife();
34          }
35          System.out.println("------------------");
36      }
37  }
```

AnimalEx 클래스

Console

<terminated> AnimalEx2 [Java Application] C:₩

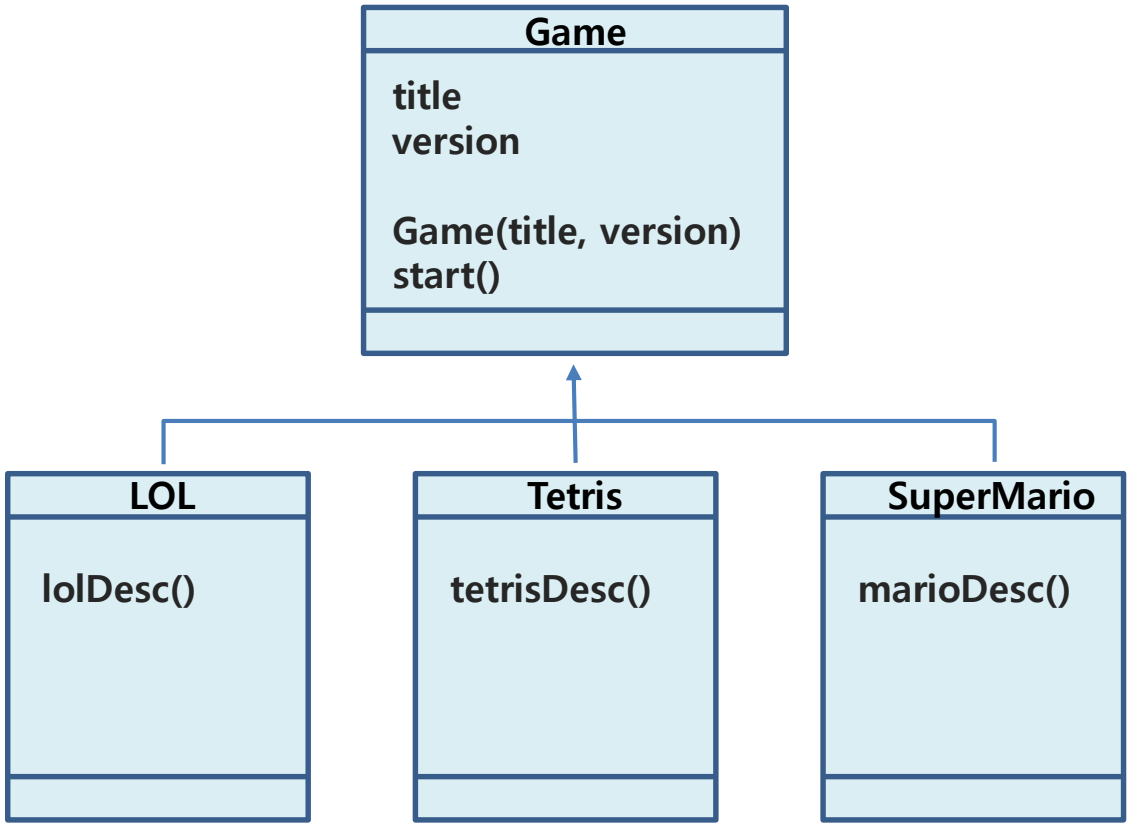
```
멍멍
야옹
------------------
멍멍
야옹
------------------
멍멍
숨을 쉽니다.
Dog 객체로 변환 가능
Dog 평균 수명은 약 15~20년입니다.
------------------
야옹
숨을 쉽니다.
Cat 객체로 변환 가능
Cat 평균 수명은 약 12~15년입니다.
------------------
```

게임 클래스

추상클래스를 이용한 상속으로 구현

❖ 게임 클래스 만들기

- 다음 3개의 게임 클래스를 만들어 보자.
- 공통된 특징을 부모 클래스(Game)로 만들어서 상속받도록 한다.
- Game 클래스는 매개변수가 있는 생성자를 가진다.
- <결과화면>을 참조해서 나머지 코드를 완성하시오 (GameEx.java).



게임 클래스

❖ 게임 클래스 만들기

```
public abstract class Game {
    private String title;
    private String version;

    public Game(String title, String version) {
        this.title = title;
        this.version = version;
    }

    public String getTitle() {
        return title;
    }
    public String getVersion() {
        return version;
    }

    public abstract void start();
    public abstract void gameDesc();
}
```

게임 클래스

❖ 게임 클래스 만들기

```
public class LOL extends Game{
    public LOL(String title, String version) {
        super(title, version);
    }

    @Override
    public void start() {
        System.out.println("제목 : " + getTitle());
        System.out.println("버전 : " + getVersion());
        System.out.println(getTitle() + "게임을 시작합니다.");
    }

    @Override
    public void gameDesc() {
        System.out.println("리그 오브 레전드는 세계 최고의 "
                + "MOBA(Multiplayer Online Battle Arena) "
                + "게임입니다.");
    }
}
```

게임 클래스

❖ 게임 클래스 만들기

```
public class Tetris extends Game{
    public Tetris(String title, String version) {
        super(title, version);
    }

    @Override
    public void start() {
        System.out.println("제목 : " + getTitle());
        System.out.println("버전 : " + getVersion());
        System.out.println(getTitle() + "게임을 시작합니다.");
    }

    @Override
    public void gameDesc() {
        System.out.println("테트리스(Tetris)는 퍼즐 게임으로, "
                + "소련의 프로그래머 알렉세이 파지트노프가 "
                + "처음 디자인하고 프로그래밍 한 게임이다.");
    }
}
```

게임 클래스

❖ 게임 클래스 만들기

```
public class Supermario extends Game{
    public Supermario(String title, String version) {
        super(title, version);
    }

    @Override
    public void start() {
        System.out.println("제목 : " + getTitle());
        System.out.println("버전 : " + getVersion());
        System.out.println(getTitle() + "게임을 시작합니다.");
    }

    @Override
    public void gameDesc() {
        System.out.println("닌텐도의 대표 비디오 게임 시리즈인 "
                + "마리오 시리즈의 핵심이 되는 본가 시리즈.");
    }
}
```

게임 클래스

❖ 게임 클래스 만들기

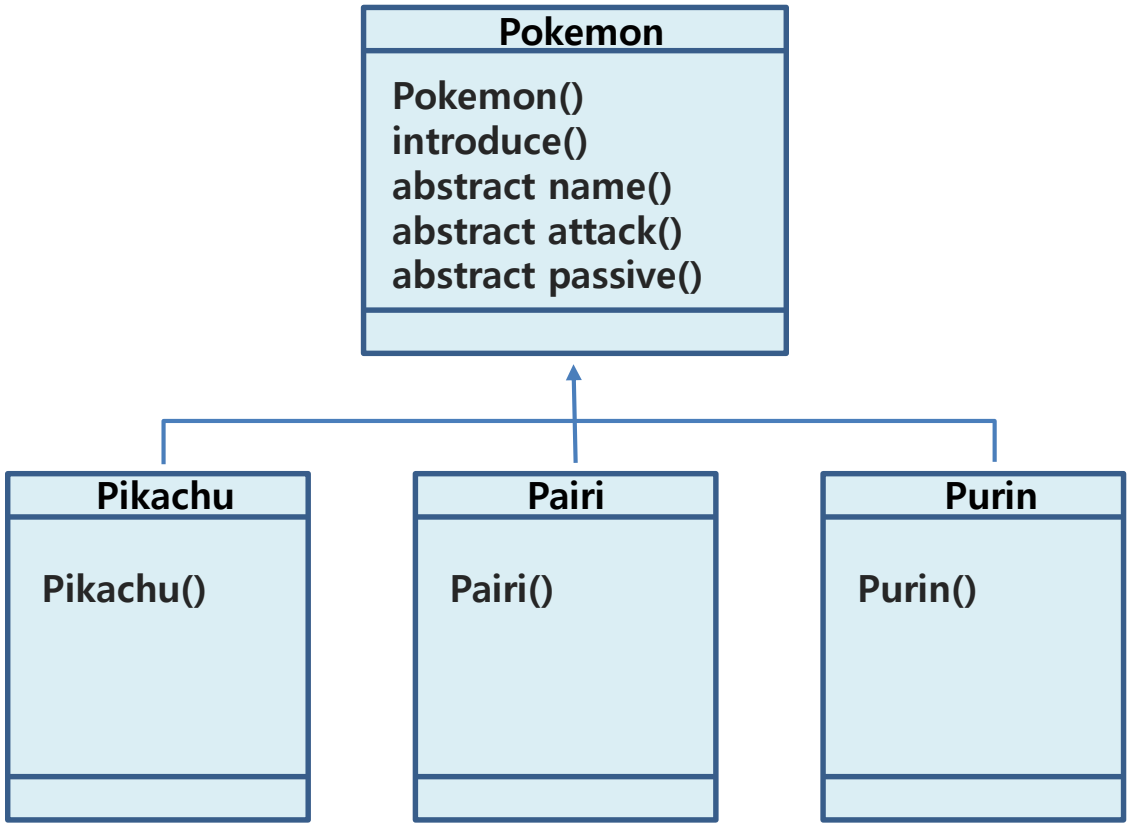
```
public class GameEx {
    public static void main(String[] args) {
        Game[] game = new Game[3];

        game[0] = new LOL("롤", "13.0");
        game[1] = new Tetris("테트리스", "12.5");
        game[2] = new Supermario("슈퍼마리오", "15.1");

        for (Game g : game) {
            g.start();
            g.gameDesc();
            System.out.println("--------------");
        }

    }
}
```

포켓몬 게임 클래스

추상클래스를 이용한 상속으로 구현

❖ 포켓몬 게임 클래스 만들기

- 다음 3개의 게임 클래스를 만들어 보자.
- 공통된 특징을 추상 클래스(Pokemon)로 만들어서 상속받도록 한다.
- Game 클래스는 매개변수가 있는 생성자를 가진다.
- <결과화면>을 참조해서 나머지 코드를 완성하시오 (GameEx.java).



포켓몬 게임 클래스

❖ 포켓몬 게임 클래스 만들기

코드 7-1 **Pokemon 클래스**

```
public abstract class Pokemon
{
  public void introduce() {
    name();
    attack();
    passive();
  }
  abstract void name();
  abstract void attack();
  abstract void passive();
}
```

코드 7-4 **Pikachu 클래스**

```
public class Pikachu extends Pokemon {
  public void passive() {
    System.out.println("패시브 스킬: 스피드 ₩n");
    // 스피드: 한 번에 두 번 공격함
  }
  public void attack() {
    System.out.println("공격 스킬: 백만 볼트");
    // 백만 볼트: 백만 볼트의 강력한 전압으로 공격함
  }
  void name() {
    System.out.println("이름: 피카츄, 속성: 번개");
  }
}
```

포켓몬 게임 클래스

❖ 포켓몬 게임 클래스 만들기

코드 7-3 **Pairi 클래스**

```
public class Pairi extends Pokemon {
  public void passive() {
    System.out.println("패시브 스킬: 방어₩n");
    // 방어: 받는 피해를 40% 감소시킴
  }
  public void attack() {
    System.out.println("공격 스킬: 불꽃");
    // 불꽃: 뜨거운 불꽃을 상대한테 쏨
  }
  void name() {
    System.out.println("이름: 파이리, 속성: 불");
  }
}
```

코드 7-2 **Purin 클래스**

```
public class Purin extends Pokemon {
  public void passive() {
    System.out.println("패시브 스킬: 회피 ₩n");
    // 회피: 30% 확률로 공격 회피
  }
  public void attack() {
    System.out.println("공격 스킬: 노래하기");
    // 노래하기: 노래를 불러 상대를 잠재움
  }
  public void name() {
    System.out.println("이름: 푸린, 속성: 노멀");
  }
}
```

포켓몬 게임 클래스

❖ 포켓몬 게임 클래스 만들기

코드 7-5 **Main 클래스**

```
public class Main {
  public static void main(String args[]) {
    Pokemon pikachu = new Pikachu();
    pikachu.introduce();

    Pokemon purin = new Purin();
    purin.introduce();

    Pokemon pairi = new Pairi();
    pairi.introduce();
  }
}
```

실행 결과

```
이름: 피카츄, 속성: 번개
공격 스킬: 백만 볼트
패시브 스킬: 스피드

이름: 푸린, 속성: 노멀
공격 스킬: 노래하기
패시브 스킬: 회피

이름: 파이리, 속성: 불
공격 스킬: 불꽃
패시브 스킬: 방어
```