



7장. 상속

Contents

- ❖ 7절. 타입변환과 다형성(polymorphism)
- ❖ 8절. 추상 클래스(Abstract Class)

7절. 타입변환과 다형성(polymorphism)

❖ 다형성 (多形性, Polymorphism)

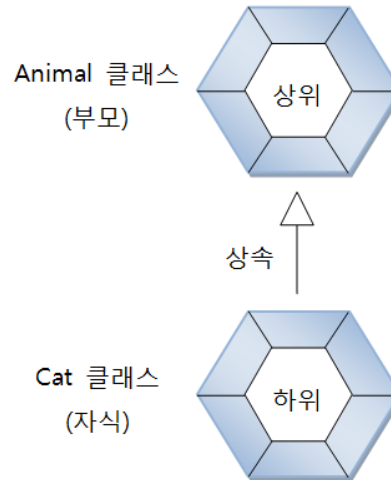
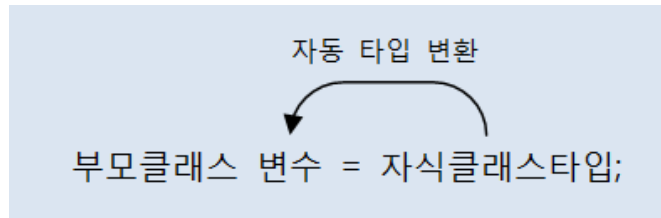
- 같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질
 - 부모 타입에는 모든 자식 객체가 대입 가능
 - 자식 타입은 부모 타입으로 자동 타입 변환
- 효과: 객체 부품화 가능



7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것



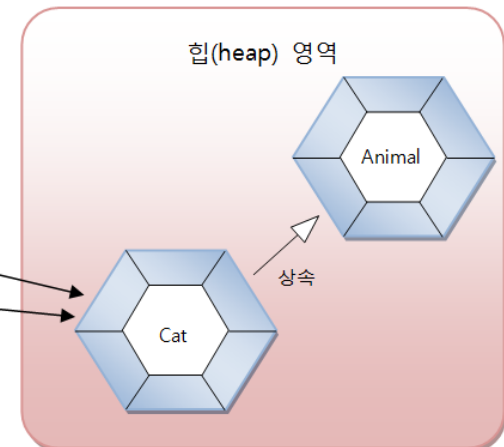
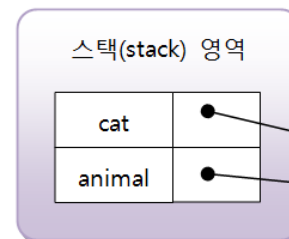
```
class Animal {  
    ...  
}
```

```
class Cat extends Animal {  
    ...  
}
```

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

Animal animal = new Cat(); 도 가능하다.

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



7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것

```
Parent.java Child.java ChildEx.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 }
```

```
Parent.java Child.java ChildEx.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 ✕
1  package week10;
2
3  public class ChildEx {
4      public static void main(String[] args) {
5          // Parent 클래스를 상속받은 Child 클래스를 이용하여 객체를 생성한다
6          Child child = new Child();
7
8          // 자식 클래스로 생성된 객체는 부모 클래스로 생성된 객체에 대입할 수 있다
9          // 이 때, 자동 타입 변환된다
10         Parent parent = child;
11
12         parent.method1();
13
14         // 자동 타입 변환이 적용된 경우 부모 클래스 객체라도 자식 클래스의 재정의된 메소드가 호출된다
15         parent.method2();
16
17         // 자식 객체가 부모 객체로 자동 타입 변환되면 자식 객체에만 정의된 메소드는 호출할 수 없다
18         //parent.method3();
19     }
20 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것

The screenshot shows an IDE with three tabs: Parent.java, Child.java, and ChildEx.java. The ChildEx.java tab is active, displaying the following code:

```
1 package week10;
2
3 public class ChildEx {
4     public static void main(String[] args) {
5         // Parent 클래스를 상속받은 Child 클래스를 이용하여 객체를 생성한다
6         Child child = new Child();
7
8         // 자식 클래스로 생성된 객체는 부모 클래스로 생성된 객체에 대입할 수 있다
9         // 이 때, 자동 타입 변환된다
10        Parent parent = child;
11
12        parent.method1();
13
14        // 자동 타입 변환이 적용된 경우 부모 클래스 객체라도 자식 클래스의 재정의된 메소드가 호출된다
15        parent.method2();
16
17        // 자식 객체가 부모 객체로 자동 타입 변환되면 자식 객체에만 정의된 메소드는 호출할 수 없다
18        //parent.method3();
19    }
20 }
```

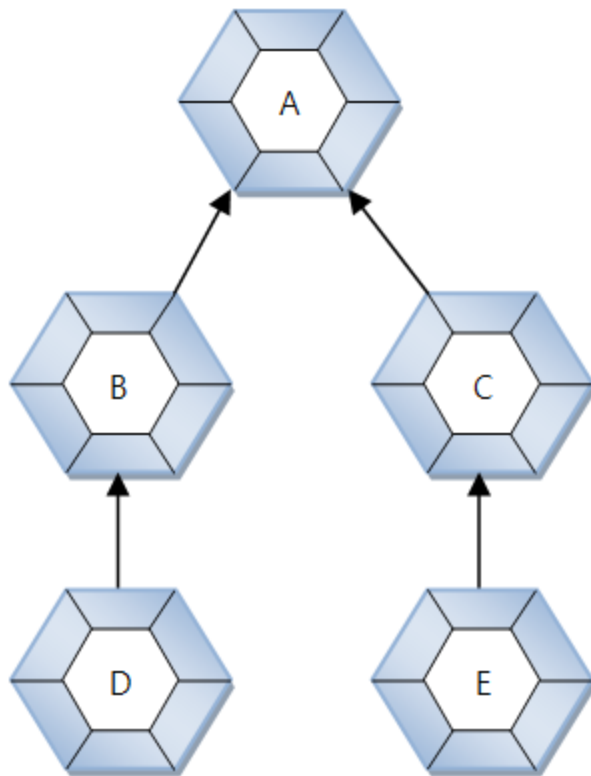
On the right, the Console window shows the output of the program:

```
<terminated> ChildEx [Java Application]
Parent-method1
Child-method2
```

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)
```

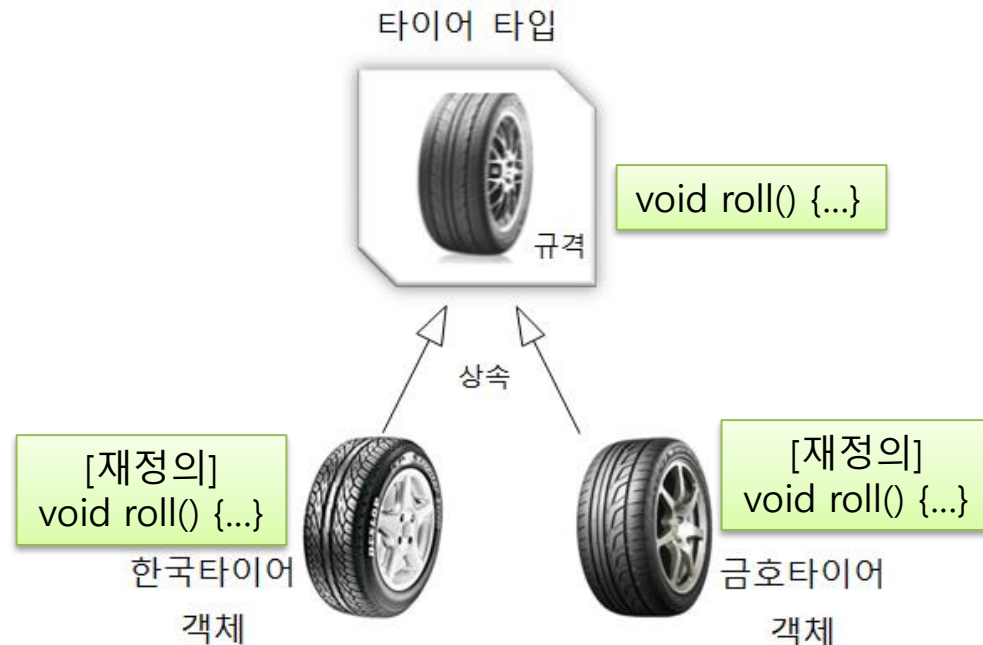
```
B b1 = d; (가능)  
C c1 = e; (가능)
```

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


7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

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



7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

■ Tire 클래스

- 속성
 - 타이어 위치, 최대 회전수(타이어 수명), 누적 회전수
- 생성자
 - 타이어 위치, 최대 회전수로 초기화
- 메소드 : roll()
 - 회전수 누적
 - 누적 회전수와 최대 회전수 비교 처리

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
Tire.java ✖
1 package week10;
2 public class Tire {
3     public String location;    //타이어 위치
4     public int    maxRotation; //최대 회전수(타이어 수명)
5     public int    accRotation; //누적 회전수
6     //생성자
7     public Tire(String location, int maxRotation) {
8         this.location    = location;
9         this.maxRotation = maxRotation;
10    }
11    public boolean roll() {
12        ++accRotation;
13
14        if (accRotation < maxRotation) {
15            //타이어 수명이 남아있는 경우
16            System.out.println(location+" 남은수명: "+(maxRotation-accRotation)+"회");
17            return true;
18        } else {
19            //타이어 수명이 다 되고 펑크난 경우
20            System.out.println("*** "+location+" 타이어 펑크 ***");
21            return false;
22        }
23    }
24 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

■ Car 클래스

- 속성
 - 타이어 (4개)
- 메소드 : run(), stop()
 - 타이어의 roll() 메소드 실행 (run())
 - 주행 정지 메시지 (stop())

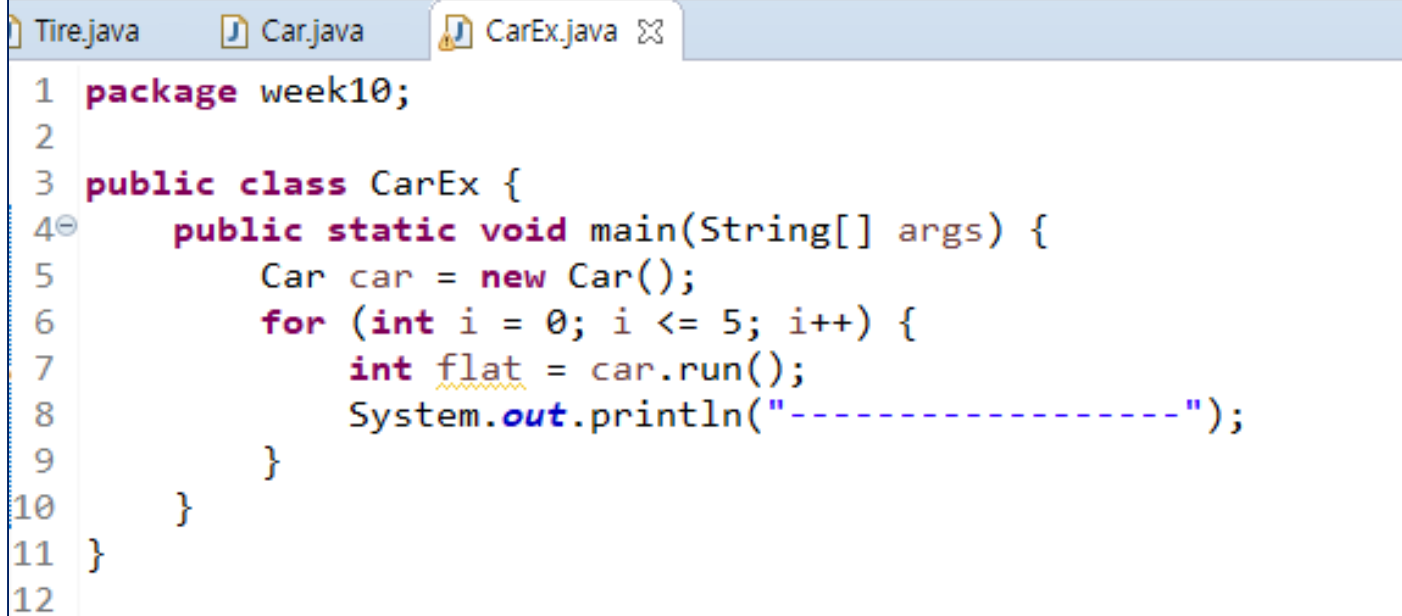
7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
Tire.java Car.java ✕
1 package week10;
2
3 public class Car {
4     //필드
5     Tire frontLeft = new Tire("앞왼쪽", 6); //1번 타이어
6     Tire frontRight = new Tire("앞오른쪽", 2); //2번 타이어
7     Tire backLeft = new Tire("뒤왼쪽", 3); //3번 타이어
8     Tire backRight = new Tire("뒤오른쪽", 4); //4번 타이어
9     //주행 메소드
10    int run() {
11        System.out.println("자동차가 달립니다.");
12        if (!frontLeft.roll()) {
13            stop();
14            return 1; //1번 타이어 리턴
15        }
16        return 0;
17    }
18    //정지 메소드
19    void stop() {
20        System.out.println("자동차가 정지합니다.");
21    }
22 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

A screenshot of a Java IDE with three tabs: Tire.java, Car.java, and CarEx.java. The CarEx.java tab is active, showing a Java class with a main method. The code is as follows:

```
1 package week10;
2
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 5; i++) {
7             int flat = car.run();
8             System.out.println("-----");
9         }
10    }
11 }
12
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
Tire.java Car.java CarEx.java
1 package week10;
2
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 5; i++) {
7             int flat = car.run();
8             System.out.println("-----");
9         }
10    }
11 }
12
```

```
Console
<terminated> CarEx (2) [Java Application] C:\#Pro
자동차가 달립니다.
앞왼쪽 남은수명: 5회
-----
자동차가 달립니다.
앞왼쪽 남은수명: 4회
-----
자동차가 달립니다.
앞왼쪽 남은수명: 3회
-----
자동차가 달립니다.
앞왼쪽 남은수명: 2회
-----
자동차가 달립니다.
앞왼쪽 남은수명: 1회
-----
자동차가 달립니다.
*** 앞왼쪽 타이어 펑크 ***
자동차가 정지합니다.
-----
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
Tire.java  Car.java  CarEx.java  Hankook.java ✕
1 package week10;
2
3 public class Hankook extends Tire {
4     public Hankook(String location, int maxRotation) {
5         super(location, maxRotation);
6     }
7     @Override
8     public boolean roll() {
9         ++accRotation;
10        if (accRotation < maxRotation) {
11            //타이어 수명이 남아있는 경우
12            System.out.println(location+" 한국타이어 남은수명: "+(maxRotation-accRotation)+"회");
13            return true;
14        } else {
15            //타이어 수명이 다 되고 펑크난 경우
16            System.out.println("*** "+location+" 한국타이어 펑크 ***");
17            return false;
18        }
19    }
20 }
```


7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
Tire.java  Car.java  CarEx.java  Hankook.java
1 package week10;
2
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 10; i++) {
7             int flat = car.run();
8             if (flat == 1) {
9                 System.out.println("한국타이어로 교체");
10                car.frontLeft = new Hankook("앞왼쪽", 15);
11            }
12            System.out.println("-----");
13        }
14    }
15 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

```
Tire.java  Car.java  CarEx.java  Hankook.java
1 package week10;
2
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 10; i++) {
7             int flat = car.run();
8             if (flat == 1) {
9                 System.out.println("한국타이어로 교체");
10                car.frontLeft = new Hankook("앞왼쪽", 15);
11            }
12            System.out.println("-----");
13        }
14    }
15 }
```

```
Console
<terminated> CarEx (2) [Java Application] C:\Program
-----
자동차가 달립니다.
앞왼쪽 남은수명: 1회
-----
자동차가 달립니다.
*** 앞왼쪽 타이어 펑크 ***
자동차가 정지합니다.
한국타이어로 교체
-----
자동차가 달립니다.
앞왼쪽 한국타이어 남은수명: 14회
-----
자동차가 달립니다.
앞왼쪽 한국타이어 남은수명: 13회
-----
자동차가 달립니다.
앞왼쪽 한국타이어 남은수명: 12회
-----
자동차가 달립니다.
앞왼쪽 한국타이어 남은수명: 11회
-----
자동차가 달립니다.
앞왼쪽 한국타이어 남은수명: 10회
-----
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 추가

- 앞오른쪽 타이어에 대해서도 수명을 체크한다
- 수명이 다 되면 한국타이어로 교체한다

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 추가

```
Tire.java  Car.java  CarEx.java  Hankook.java
1 package week10;
2
3 public class Car {
4     //필드
5     Tire frontLeft = new Tire("앞왼쪽", 6);    //1번 타이어
6     Tire frontRight = new Tire("앞오른쪽", 2); //2번 타이어
7     Tire backLeft = new Tire("뒤왼쪽", 3);     //3번 타이어
8     Tire backRight = new Tire("뒤오른쪽", 4);  //4번 타이어
9     //주행 메소드
10    int run() {
11        System.out.println("자동차가 달립니다.");
12        if (!frontLeft.roll()) {
13            stop();
14            return 1;    //1번 타이어 리턴
15        }
16        if (!frontRight.roll()) {
17            stop();
18            return 2;    //2번 타이어 리턴
19        }
20        return 0;
21    }
22    //정지 메소드
23    void stop( ) {
24        System.out.println("자동차가 정지합니다.");
25    }
26 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 추가

```
Tire.java  Car.java  CarEx.java  Hankook.java
1 package week10;
2
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 10; i++) {
7             int flat = car.run();
8             if (flat == 1) {
9                 System.out.println("한국타이어로 교체");
10                car.frontLeft = new Hankook("앞왼쪽", 15);
11            } else if (flat == 2){
12                System.out.println("한국타이어로 교체");
13                car.frontRight = new Hankook("앞오른쪽", 15);
14            }
15            System.out.println("-----");
16        }
17    }
18 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 추가

```
Tire.java Car.java CarEx.java Hankook.java
1 package week10;
2
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 10; i++) {
7             int flat = car.run();
8             if (flat == 1) {
9                 System.out.println("한국타이어로 교체");
10                car.frontLeft = new Hankook("앞왼쪽", 15);
11            } else if (flat == 2){
12                System.out.println("한국타이어로 교체");
13                car.frontRight = new Hankook("앞오른쪽", 15);
14            }
15            System.out.println("-----");
16        }
17    }
18 }
```

자동차가 달립니다.
앞왼쪽 남은수명: 5회
앞오른쪽 남은수명: 1회

자동차가 달립니다.
앞왼쪽 남은수명: 4회
*** 앞오른쪽 타이어 펑크 ***
자동차가 정지합니다.
한국타이어로 교체

자동차가 달립니다.
앞왼쪽 남은수명: 3회
앞오른쪽 한국타이어 남은수명: 14회

자동차가 달립니다.
앞왼쪽 남은수명: 2회
앞오른쪽 한국타이어 남은수명: 13회

자동차가 달립니다.
앞왼쪽 남은수명: 1회
앞오른쪽 한국타이어 남은수명: 12회

자동차가 달립니다.
*** 앞왼쪽 타이어 펑크 ***
자동차가 정지합니다.
한국타이어로 교체

자동차가 달립니다.
앞왼쪽 한국타이어 남은수명: 14회
앞오른쪽 한국타이어 남은수명: 11회

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 자식 클래스 추가

```
Tire.java  Car.java  CarEx.java  Hankook.java  Kumho.java ✖
1 package week10;
2 public class Kumho extends Tire {
3     public Kumho(String location, int maxRotation) {
4         super(location, maxRotation);
5     }
6
7     @Override
8     public boolean roll() {
9         ++accRotation;
10        if (accRotation < maxRotation) {
11            //타이어 수명이 남아있는 경우
12            System.out.println(location+" 금호타이어 남은수명: "+(maxRotation-accRotation)+"회");
13            return true;
14        } else {
15            //타이어 수명이 다 되고 펑크난 경우
16            System.out.println("*** "+location+" 금호타이어 펑크 ***");
17            return false;
18        }
19    }
20 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 - 자식 클래스 추가

- 뒷 타이어의 수명이 다 된 경우 금호타이어 클래스를 이용하여 교체한다
- 금호타이어의 최대 회전수는 200이다

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 - 자식 클래스 추가

- 뒷 타이어의 수명이 다 된 경우 금호타이어 클래스를 이용하여 교체한다
- 금호타이어의 최대 회전수는 200이다
- Car.java

```
9      //주행 메소드
10     int run() {
11         System.out.println("자동차가 달립니다.");
12         if (!frontLeft.roll()) {
13             stop();
14             return 1;           //1번 타이어 리턴
15         }
16         if (!frontRight.roll()) {
17             stop();
18             return 2;          //2번 타이어 리턴
19         }
20         if (!backLeft.roll()) {
21             stop();
22             return 3;          //3번 타이어 리턴
23         }
24         if (!backRight.roll()) {
25             stop();
26             return 4;          //4번 타이어 리턴
27         }
28         return 0;
29     }
```

7절. 타입변환과 다형성(polymorphism)

❖ 필드의 다형성 예제 - 자식 클래스 추가

- 뒷 타이어의 수명이 다 된 경우 금호타이어 클래스를 이용하여 교체한다
- 금호타이어의 최대 회전수는 200이다
- CarEx.java

```
3 public class CarEx {
4     public static void main(String[] args) {
5         Car car = new Car();
6         for (int i = 0; i <= 10; i++) {
7             int flat = car.run();
8             if (flat == 1) {
9                 System.out.println("한국타이어로 교체");
10                car.frontLeft = new Hankook("앞왼쪽", 15);
11            } else if (flat == 2){
12                System.out.println("한국타이어로 교체");
13                car.frontRight = new Hankook("앞오른쪽", 15);
14            } else if (flat == 3){
15                System.out.println("금호타이어로 교체");
16                car.backLeft = new Hankook("뒤왼쪽", 20);
17            } else if (flat == 4){
18                System.out.println("금호타이어로 교체");
19                car.backRight = new Hankook("뒤오른쪽", 20);
20            }
21            System.out.println("-----");
22        }
23    }
24 }
```

7절. 타입변환과 다형성(polymorphism)

❖ 하나의 배열로 객체 관리

```
class Car {  
    Tire frontLeftTire = new Tire("앞왼쪽", 6);  
    Tire frontRightTire = new Tire("앞오른쪽", 2);  
    Tire backLeftTire = new Tire("뒤왼쪽", 3);  
    Tire backRightTire = new Tire("뒤오른쪽", 4);  
}
```

```
class Car {  
    Tire[] tires = {  
        new Tire("앞왼쪽", 6),  
        new Tire("앞오른쪽", 2),  
        new Tire("뒤왼쪽", 3),  
        new Tire("뒤오른쪽", 4)  
    };  
}
```

```
tires[1] = new KumhoTire("앞오른쪽", 13);
```

```
int run() {  
    System.out.println("[자동차가 달립니다.]");  
    for(int i=0; i<tires.length; i++) {  
        if(tires[i].roll()==false) {  
            stop();  
            return (i+1);  
        }  
    }  
    return 0;  
}
```

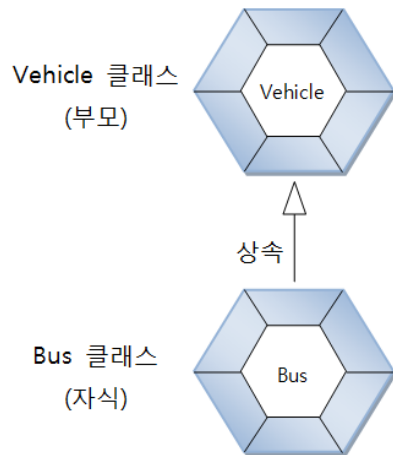
7절. 타입변환과 다형성(polymorphism)

❖ 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;

자식 객체

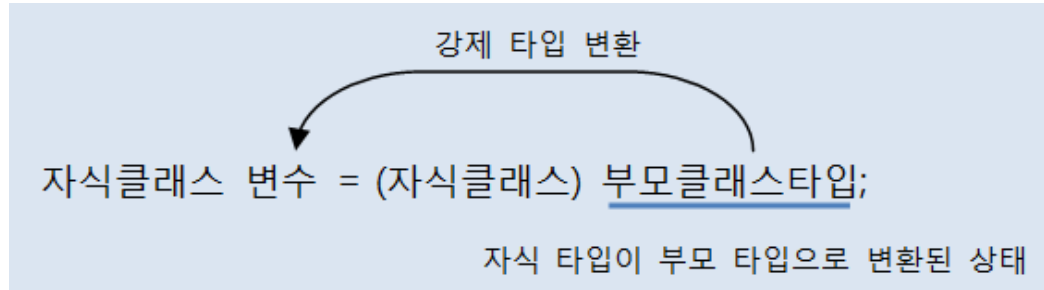
```
void drive(Vehicle vehicle) {  
    vehicle.run();  
}
```

자식 객체가 재정의한 run() 메소드 실행

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 변환 이 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)

❖ 객체 타입 확인(instanceof)

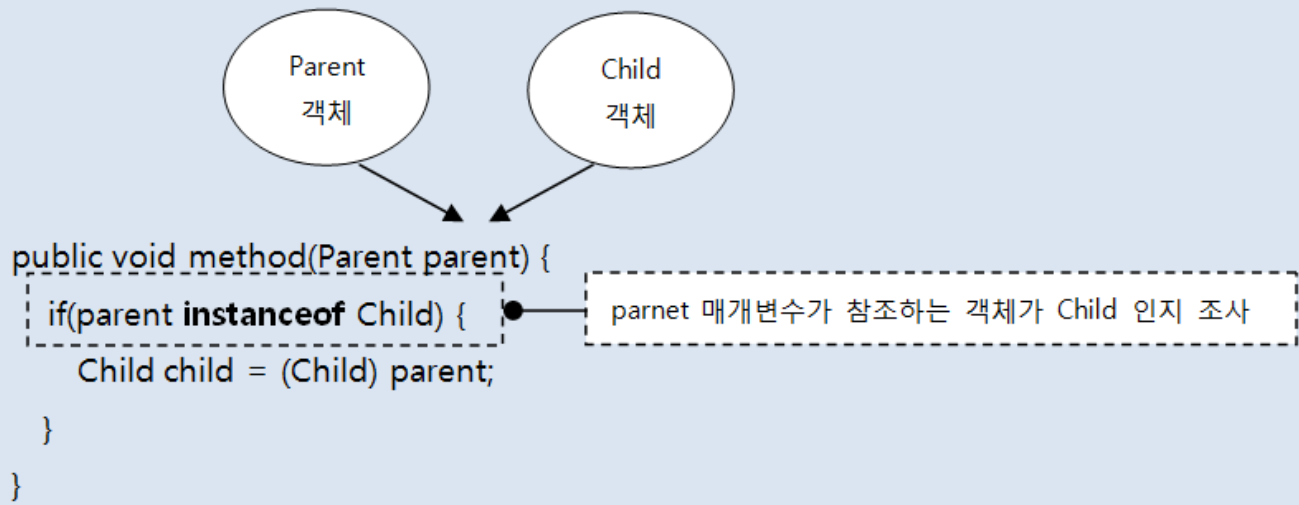
- 부모 타입이면 모두 자식 타입으로 강제 타입 변환할 수 있는 것 아님
 - **ClassCastException** 예외 발생 가능

```
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)
```



8절. 추상 클래스(Abstract Class)

❖ 추상 클래스 개념

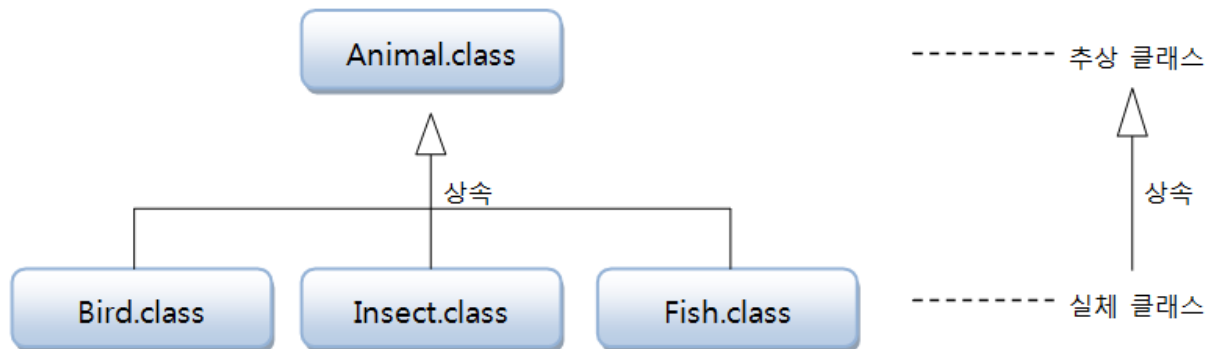
■ 추상(abstract)

- 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 메소드 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

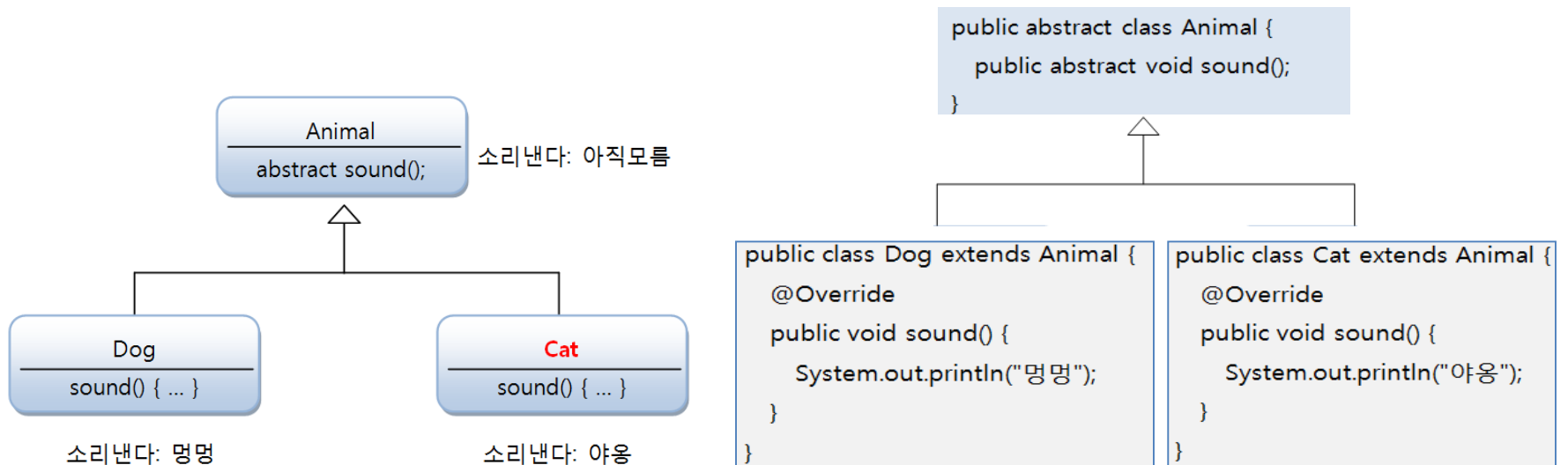
- 클래스 선언에 **abstract** 키워드 사용
 - **New** 연산자로 객체 생성하지 못하고 **상속 통해 자식 클래스만 생성 가능**

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

8절. 추상 클래스(Abstract Class)

❖ 추상 메소드와 오버라이딩(재정의)

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



8절. 추상 클래스(Abstract Class)

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

```
1 package week10;
2
3 public abstract class Animal {
4     public String kind;
5
6     public void breathe() {
7         System.out.println("숨을 쉽니다.");
8     }
9     public abstract void sound();
10 }
```

Animal 클래스

Dog 클래스

AnimalEx.java

```
1 package week10;
2
3 public class Dog extends Animal {
4
5     public Dog() {
6         this.kind = "포유류";
7     }
8
9     @Override
10    public void sound() {
11        System.out.println("멍멍");
12    }
13
14    public void dogLife() {
15        System.out.println("Dog 평균 수명은 약 15~20년입니다.");
16    }
17 }
```

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 ✕
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:\W

멍멍

야옹

멍멍

야옹

멍멍

숨을 쉽니다.

Dog 객체로 변환 가능

Dog 평균 수명은 약 15~20년입니다.

야옹

숨을 쉽니다.

Cat 객체로 변환 가능

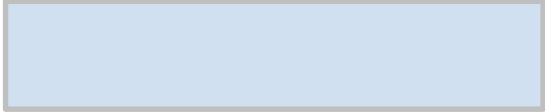
Cat 평균 수명은 약 12~15년입니다.

실습


< Car2 클래스 >

```
public abstract class Car2 {  
    int    speed = 0;  
    String color;  
  
    void upSpeed(int speed) {  
        this.speed += speed;  
    }  
  
    abstract void work();  
}
```


< Car2Ex 클래스 >

```
public class Car2Ex {  
    public static void main(String[] args) {  
        Sedan sedan = new Sedan();  
        Truck truck = new Truck();  
  
          
    }  
}
```

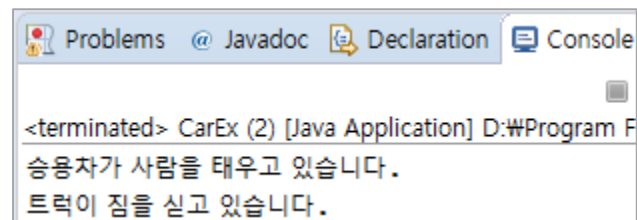
< Sedan 클래스 >

```
public class Sedan extends Car2 {  
  
      
}
```

< Truck 클래스 >

```
public class Truck extends Car2 {  
  
      
}
```

< 실행결과 >



실습

< Car2 클래스 >

```
public abstract class Car2 {  
    int    speed = 0;  
    String color;  
  
    void upSpeed(int speed) {  
        this.speed += speed;  
    }  
  
    abstract void work();  
}
```

< Car2Ex 클래스 >

```
public class Car2Ex {  
    public static void main(String[] args) {  
        Sedan sedan = new Sedan();  
        Truck truck = new Truck();  
  
        sedan.work();  
        truck.work();  
    }  
}
```

< Sedan 클래스 >

```
public class Sedan extends Car2 {  
    @Override  
    void work() {  
        System.out.println("승용차가 사람을 태우고 있습니다.");  
    }  
}
```

< Truck 클래스 >

```
public class Truck extends Car2 {  
    @Override  
    void work() {  
        System.out.println("트럭이 짐을 싣고 있습니다.");  
    }  
}
```

< 실행결과 >

