

** 타입변환과 다형성(polymorphism)

❖ 다형성 (多形性, Polymorphism)

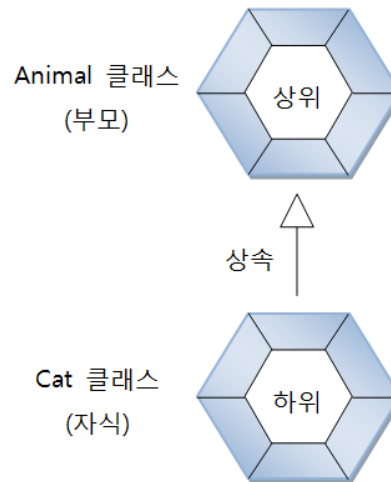
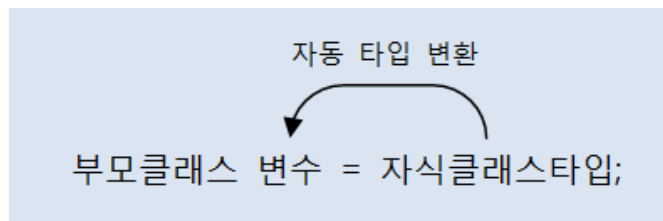
- 같은 타입이지만 실행 결과가 다양한 객체 대입(이용) 가능한 성질
 - 부모 타입에는 모든 자식 객체가 대입 가능
 - 자식 타입은 부모 타입으로 자동 타입 변환
- 효과: 객체 부품화 가능



** 타입변환과 다형성(polymorphism)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동 타입 변환이 일어나는 것



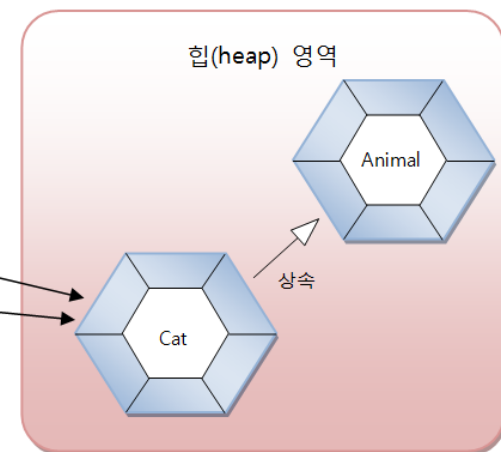
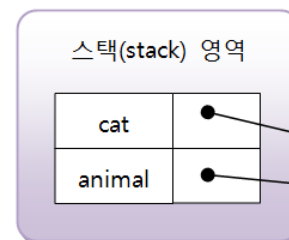
```
class Animal {  
    ...  
}
```

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

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

Animal animal = new Cat(); 도 가능하다.

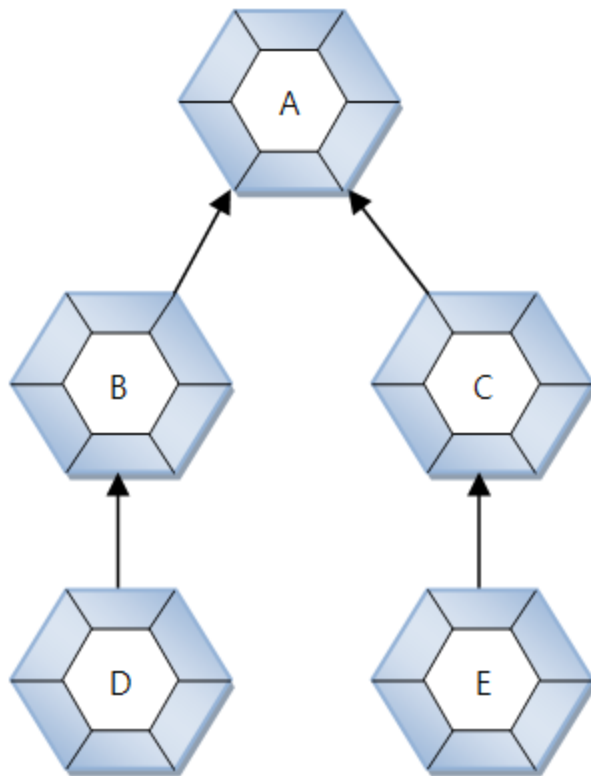
```
cat == animal //true
```



** 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)
```



** 타입변환과 다형성(polymorphism)

```
class A{}  
class B extends A{}  
class C extends A{}  
class D extends B{}  
class E extends C{}
```

```
public class PromotionExample {  
    public static void main(String[] args)  
    {  
  
        B b = new B();  
        C c = new C();  
        D d = new D();  
        E e = new E();  
  
        A a1 = b;  
        A a2 = c;  
        A a3 = d;  
        A a4 = e;  
  
        B b1 = d;  
        C c1 = e;  
  
        //B b3 = e;  
        //C c2 = d;  
  
    }  
}
```



** 타입변환과 다형성(polymorphism)

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

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

```
public class ChildExample {  
    public static void main(String[] args) {  
        Child child = new Child();  
  
        Parent parent = child; // 형변환  
  
        parent.method1();  
  
        parent.method2(); // 재정의 호출  
  
        //parent.method3(); (호출 불가능)  
    }  
}
```

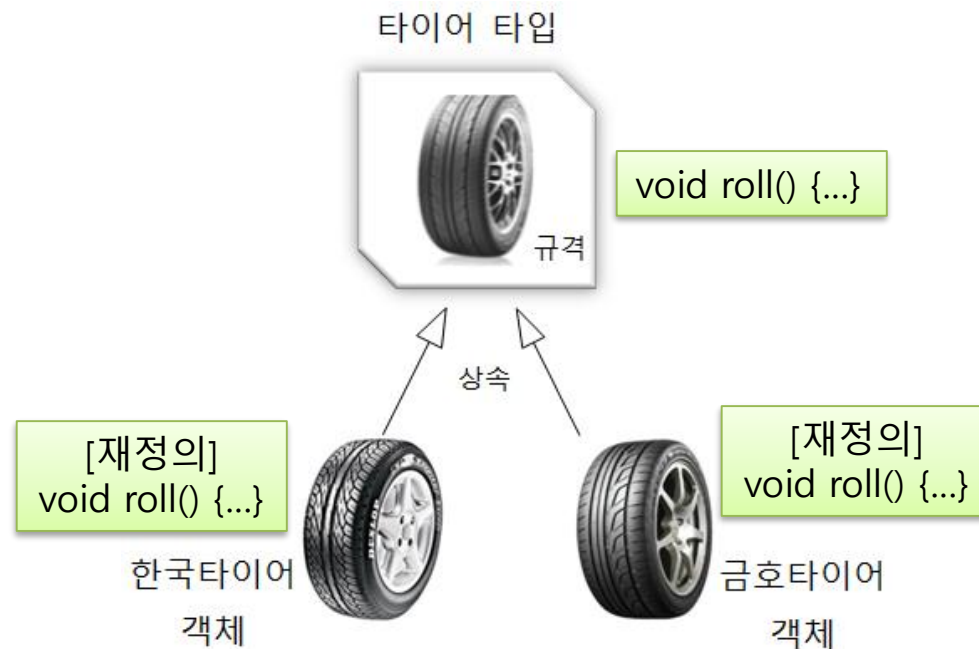


** 타입변환과 다형성(polymorphism)

❖ 필드의 다형성

■ 다형성을 구현하는 기술적 방법

- 부모 타입으로 자동 변환
- 재정의된 메소드(오버라이딩)



** 타입변환과 다형성(polymorphism)

```
public class Tire {  
    //필드  
    public int maxRotation;           //최대 회전수(최대 수명)  
    public int accumulatedRotation;   //누적 회전수  
    public String location;           //타이어의 위치  
  
    //생성자  
    public Tire(String location, int maxRotation) {  
        this.location = location;  
        this.maxRotation = maxRotation;  
    }  
  
    //메소드  
    public boolean roll() {  
        ++accumulatedRotation;  
        if(accumulatedRotation < maxRotation) {  
            System.out.println(location + " Tire 수명: " + (maxRotation - accumulatedRotation) + "회");  
            return true;  
        } else {  
            System.out.println("*** " + location + " Tire 펑크 ***");  
            return false;  
        }  
    }  
}
```



** 타입변환과 다형성(polymorphism)

```
public class Car {  
    //필드  
    Tire frontLeftTire = new Tire("앞왼쪽", 6);  
    Tire frontRightTire = new Tire("앞오른쪽", 2);  
    Tire backLeftTire = new Tire("뒤왼쪽", 3);  
    Tire backRightTire = new Tire("뒤오른쪽", 4);  
  
    //생성자  
  
    //메소드  
    int run() {  
        System.out.println("[자동차가 달립니다.]");  
        if(frontLeftTire.roll()==false) { stop(); return 1; };  
        if(frontRightTire.roll()==false) { stop(); return 2; };  
        if(backLeftTire.roll()==false) { stop(); return 3; };  
        if(backRightTire.roll()==false) { stop(); return 4; };  
        return 0;  
    }  
  
    void stop() {  
        System.out.println("[자동차가 멈춥니다.]");  
    }  
}
```



** 타입변환과 다형성(polymorphism)

```
public class HankookTire extends Tire {
    //필드
    //생성자
    public HankookTire(String location, int maxRotation) {
        super(location, maxRotation);
    }
    //메소드
    @Override
    public boolean roll() {
        ++accumulatedRotation;
        if(accumulatedRotation<maxRotation) {
            System.out.println(location + " HankookTire 수명: " + (maxRotation-accumulatedRotation)
                                + "회");

            return true;
        } else {
            System.out.println("*** " + location + " HankookTire 펑크 ***");
            return false;
        }
    }
}
```



** 타입변환과 다형성(polymorphism)

```
public class KumhoTire extends Tire {
    //필드
    //생성자
    public KumhoTire (String location, int maxRotation) {
        super(location, maxRotation);
    }
    //메소드
    @Override
    public boolean roll() {
        ++accumulatedRotation;
        if(accumulatedRotation<maxRotation) {
            System.out.println(location + " KumhoTire 수명: " + (maxRotation-accumulatedRotation)
                                + "회");

            return true;
        } else {
            System.out.println("*** " + location + " KumhoTire펑크 ***");
            return false;
        }
    }
}
```



** 타입변환과 다형성(polymorphism)

```
public class CarExample {
    public static void main(String[] args) {
        Car car = new Car();

        for(int i=1; i<=5; i++) {
            int problemLocation = car.run();
            switch(problemLocation) {
                case 1:
                    System.out.println("앞왼쪽 HankookTire로 교체");
                    car.frontLeftTire = new HankookTire("앞왼쪽", 15);
                    break;

                case 2:
                    System.out.println("앞오른쪽 KumhoTire로 교체");
                    car.frontRightTire = new KumhoTire("앞오른쪽", 13);
                    break;

                case 3:
                    System.out.println("뒤왼쪽 HankookTire로 교체");
                    car.backLeftTire = new HankookTire("뒤왼쪽", 14);
                    break;

                case 4:
                    System.out.println("뒤오른쪽 KumhoTire로 교체");
                    car.backRightTire = new KumhoTire("뒤오른쪽", 17);
                    break;
            }
            System.out.println("-----");
        }
    }
}
```



** 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;  
}
```

** 타입변환과 다형성(polymorphism)

```
public class Tire {  
    //필드  
    public int maxRotation;           //최대 회전수(최대 수명)  
    public int accumulatedRotation;   //누적 회전수  
    public String location;           //타이어의 위치  
  
    //생성자  
    public Tire(String location, int maxRotation) {  
        this.location = location;  
        this.maxRotation = maxRotation;  
    }  
  
    //메소드  
    public boolean roll() {  
        ++accumulatedRotation;  
        if(accumulatedRotation < maxRotation) {  
            System.out.println(location + " Tire 수명: " + (maxRotation - accumulatedRotation) + "회");  
            return true;  
        } else {  
            System.out.println("*** " + location + " Tire 펑크 ***");  
            return false;  
        }  
    }  
}
```



** 타입변환과 다형성(polymorphism)

```
public class Car {  
    //필드  
    Tire[] tires = {  
        new Tire("앞왼쪽", 6),  
        new Tire("앞오른쪽", 2),  
        new Tire("뒤왼쪽", 3),  
        new Tire("뒤오른쪽", 4)  
    };  
  
    //메소드  
    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;  
    }  
  
    void stop() {  
        System.out.println("[자동차가 멈춥니다.]");  
    }  
}
```



** 타입변환과 다형성(polymorphism)

```
public class HankookTire extends Tire {  
    //필드  
    //생성자  
    public HankookTire(String location, int maxRotation) {  
        super(location, maxRotation);  
    }  
    //메소드  
    @Override  
    public boolean roll() {  
        ++accumulatedRotation;  
        if(accumulatedRotation<maxRotation) {  
            System.out.println(location + " HankookTire 수명: " + (maxRotation-accumulatedRotation) +  
                                "회");  
            return true;  
        } else {  
            System.out.println("*** " + location + " HankookTire 펑크 ***");  
            return false;  
        }  
    }  
}
```



** 타입변환과 다형성(polymorphism)

```
public class KumhoTire extends Tire {  
    //필드  
    //생성자  
    public KumhoTire (String location, int maxRotation) {  
        super(location, maxRotation);  
    }  
    //메소드  
    @Override  
    public boolean roll() {  
        ++accumulatedRotation;  
        if(accumulatedRotation<maxRotation) {  
            System.out.println(location + " KumhoTire 수명: " + (maxRotation-accumulatedRotation) +  
                                "회");  
            return true;  
        } else {  
            System.out.println("*** " + location + " KumhoTire 펑크 ***");  
            return false;  
        }  
    }  
}
```



** 타입변환과 다형성(polymorphism)

```
public class CarExample {  
    public static void main(String[] args) {  
        Car car = new Car();  
  
        for(int i=1; i<=5; i++) {  
            int problemLocation = car.run();  
            if(problemLocation != 0) {  
                System.out.println(car.tires[problemLocation-1].location + " HankookTire로 교체");  
                car.tires[problemLocation-1] = new HankookTire(car.tires[problemLocation-1].location,  
                                                                15);  
            }  
            System.out.println("-----");  
        }  
    }  
}
```



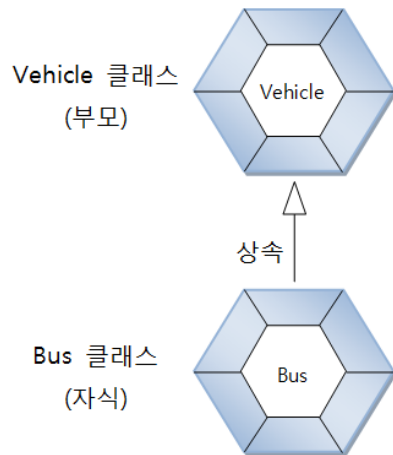
** 타입변환과 다형성(polymorphism)

❖ 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 Driver();  
  
Bus bus = new Bus();  
  
driver.drive( bus );
```

자동 타입 변환 발생
Vehicle vehicle = bus;

자식 객체

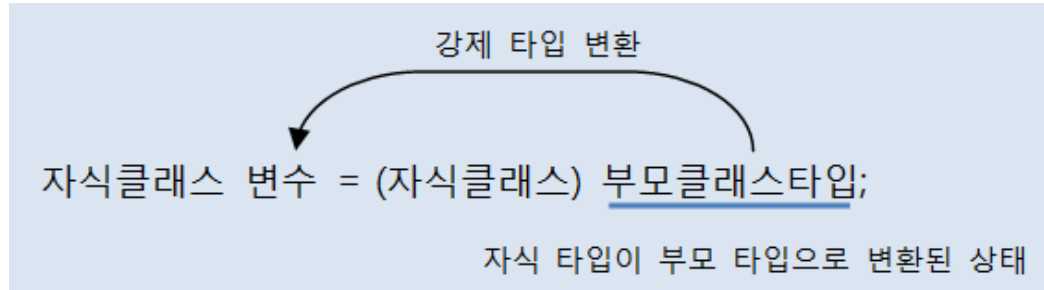
```
void drive(Vehicle vehicle) {  
    vehicle.run();  
}
```

자식 객체가 재정의한 run() 메소드 실행

** 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



** 타입변환과 다형성(polymorphism)

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

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

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

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



** 타입변환과 다형성(polymorphism)

```
public class DriverExample {  
    public static void main(String[] args) {  
        Driver driver = new Driver();  
  
        Bus bus = new Bus();  
        Taxi taxi = new Taxi();  
  
        driver.drive(bus);  
        driver.drive(taxi);  
    }  
}
```



** 타입변환과 다형성(polymorphism)

❖ 객체 타입 확인(instanceof)

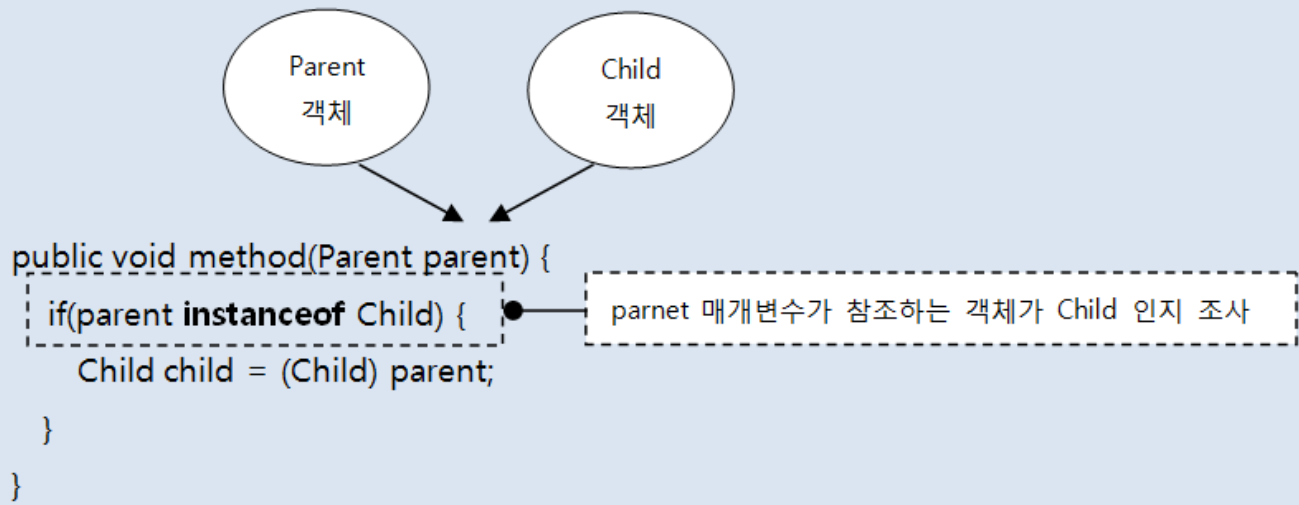
- 부모 타입이면 모두 자식 타입으로 강제 타입 변환할 수 있는 것 아님
 - **ClassCastException** 예외 발생 가능

```
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)
```



** 타입변환과 다형성(polymorphism)

```
public class Parent {  
    public String field1;  
  
    public void method1() { System.out.println("Parent-method1()"); }  
    public void method2() { System.out.println("Parent-method2()"); }  
}
```

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

```
public class ChildExample {  
    public static void main(String[] args) {  
        Parent parent = new Child();  
        parent.field1 = "data1";  
        parent.method1();  
        parent.method2();  
        // parent.field2 = "data2"; //(불가능)  
        // parent.method3();      //(불가능)  
  
        Child child = (Child) parent;  
        child.field2 = "yyy"; //(가능)  
        child.method3();      //(가능)  
    }  
}
```

