타입변환과 다형성

타입변환과 다형성

❖ 다형성

- 하나의 타입에 여러 가지 객체 대입해 다양한 실행 결과를 얻는 것

- 다형성을 구현하는 기술
  - 상속 또는 인터페이스의 자동 타입 변환(Promotion)
  - 오버라이딩(Overriding)

- 다형성의 효과
  - 다양한 실행 결과를 얻을 수 있음
  - 객체를 부품화시킬 수 있어 유지보수 용이 (메소드의 매개변수로 사용)

타입변환과 다형성

❖ 다형성

[프로그램]

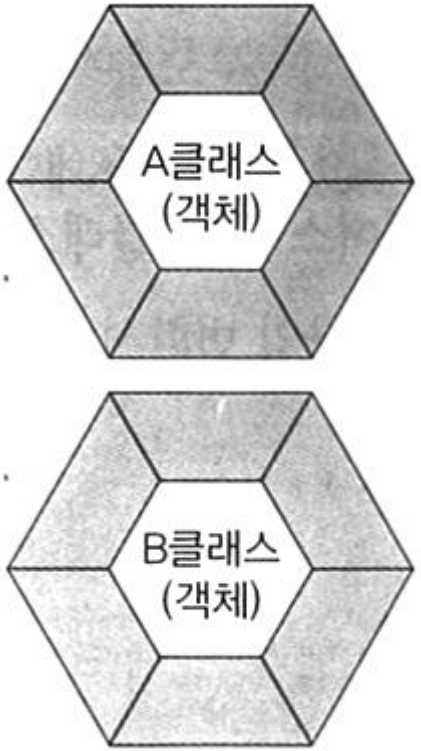
~~I i = new A();~~ 수정
I i = new B();

i.method1();
i.method2(); } 수정이 필요 없음

```
interface I {
   void method1();
   void method2();
}
```

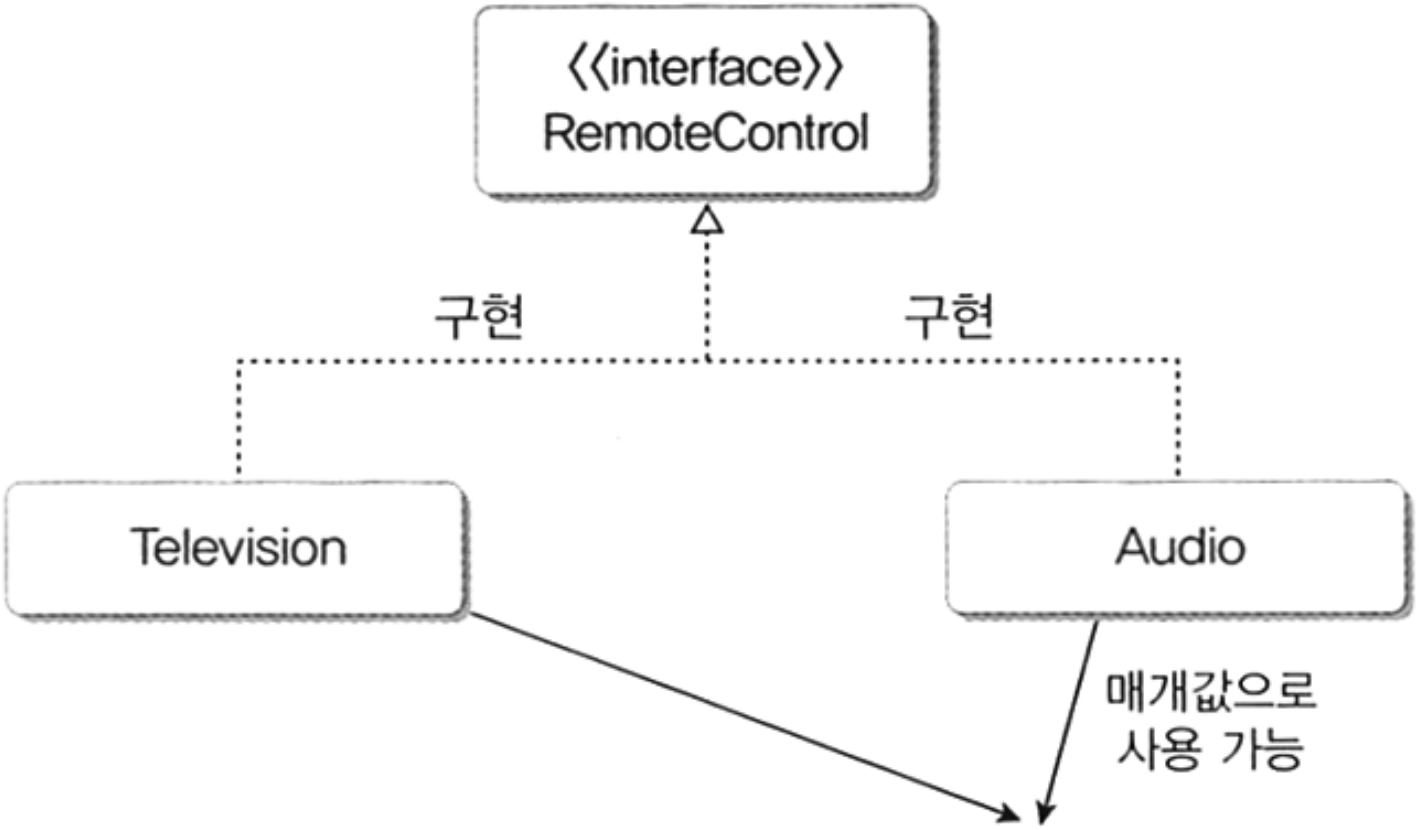
구현

구현



타입변환과 다형성

❖ 다형성



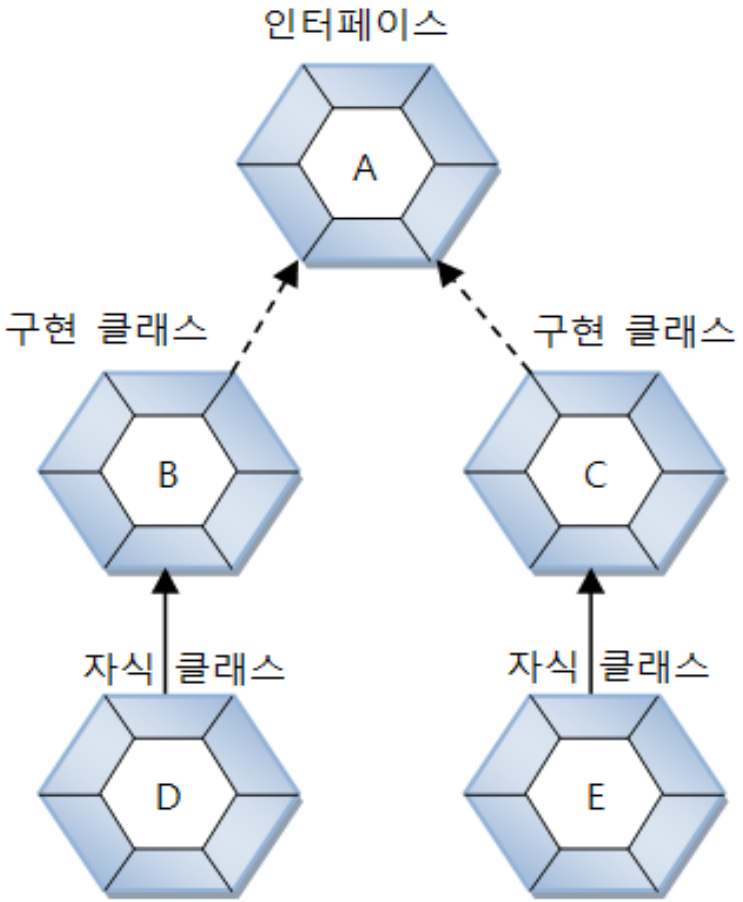

```
public void useRemoteControl( RemoteControl rc ) { … }
```

타입변환과 다형성

❖ 자동 타입 변환(Promotion)

자동 타입 변환

인터페이스 변수 = 구현객체;

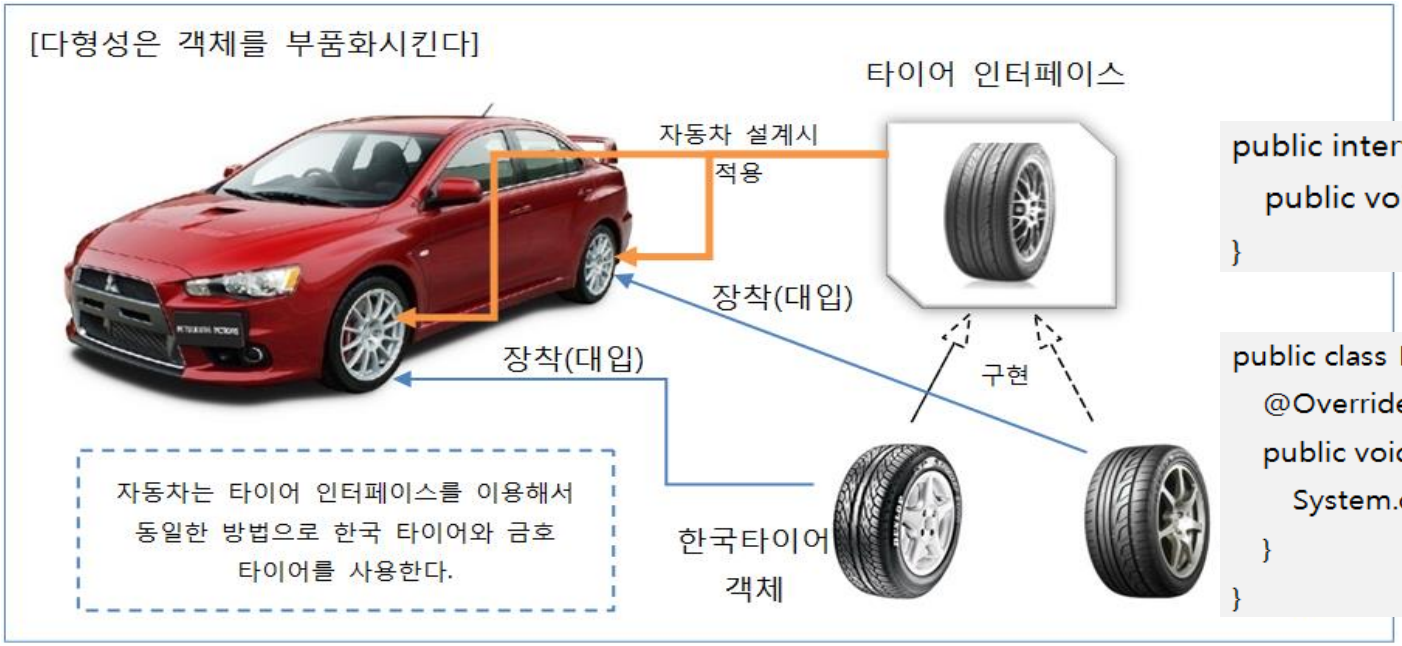



```
B b = new B();
C c = new C();
D d = new D();
E e = new E();
```

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

타입변환과 다형성

❖ 필드의 다형성




```
public interface Tire {
    public void roll();
}
```

```
public class HankookTire implements Tire {
    @Override
    public void roll() {
        System.out.println("한국 타이어가 굴러갑니다.");
    }
}
```

```
public class Car {
    Tire frontLeftTire = new HankookTire();
    Tire frontRightTire = new HankookTire();
    Tire backLeftTire = new HankookTire();
    Tire backRightTire = new HankookTire();

    void run() {
        frontLeftTire.roll();
        frontRightTire.roll();
        backLeftTire.roll();
        backRightTire.roll();
    }
```

```
Car myCar = new Car();
myCar.frontLeftTire = new KumhoTire();
myCar.frontRightTire = new KumhoTire();
```

```
myCar.run();
```

타입변환과 다형성

❖ 인터페이스: Tire.java

```
public interface Tire {
    public void roll();
}
```

❖ 구현 클래스: HankookTire.java

```
public class HankookTire implements Tire {
    @Override
    public void roll() {
        System.out.println("한국 타이어가 굴러갑니다.");
    }
}
```

❖ 구현 클래스: KumhoTire.java

```
public class KumhoTire implements Tire {
    @Override
    public void roll() {
        System.out.println("금호 타이어가 굴러갑니다.");
    }
}
```

❖ 필드 다형성: Car.java

```
public class Car {
    Tire frontLeftTire = new HankookTire();
    Tire frontRightTire = new HankookTire();
    Tire backLeftTire = new HankookTire();
    Tire backRightTire = new HankookTire();

    void run() {
        frontLeftTire.roll();
        frontRightTire.roll();
        backLeftTire.roll();
        backRightTire.roll();
    }
}
```

❖ 필드 다형성 테스트: CarExample.java

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

        myCar.run();

        myCar.frontLeftTire = new KumhoTire();
        myCar.frontRightTire = new KumhoTire();

        myCar.run();
    }
}
```

타입변환과 다형성

❖ 인터페이스 배열로 구현한 객체 관리

```
Tire[] tires = {
    new HankookTire(),
    new HankookTire(),
    new HankookTire(),
    new HankookTire()
};
```

```
tires[1] = new KumhoTire();
```

```
void run() {
    for(Tire tire : tires) {
        tire.roll();
    }
}
```

❖ 필드 다형성 : Car.java

```
public class Car {
    Tire[] tires = {
        new HankookTire(),
        new HankookTire(),
        new HankookTire(),
        new HankookTire()
    };

    void run() {
        for(Tire tire : tires) {
            tire.roll();
        }
    }
}
```

❖ 필드 다형성 테스트: CarExample.java

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

        myCar.run();

        myCar.tires[0] = new KumhoTire();
        myCar.tires[1] = new KumhoTire();

        myCar.run();
    }
}
```

❖ 매개변수의 다형성

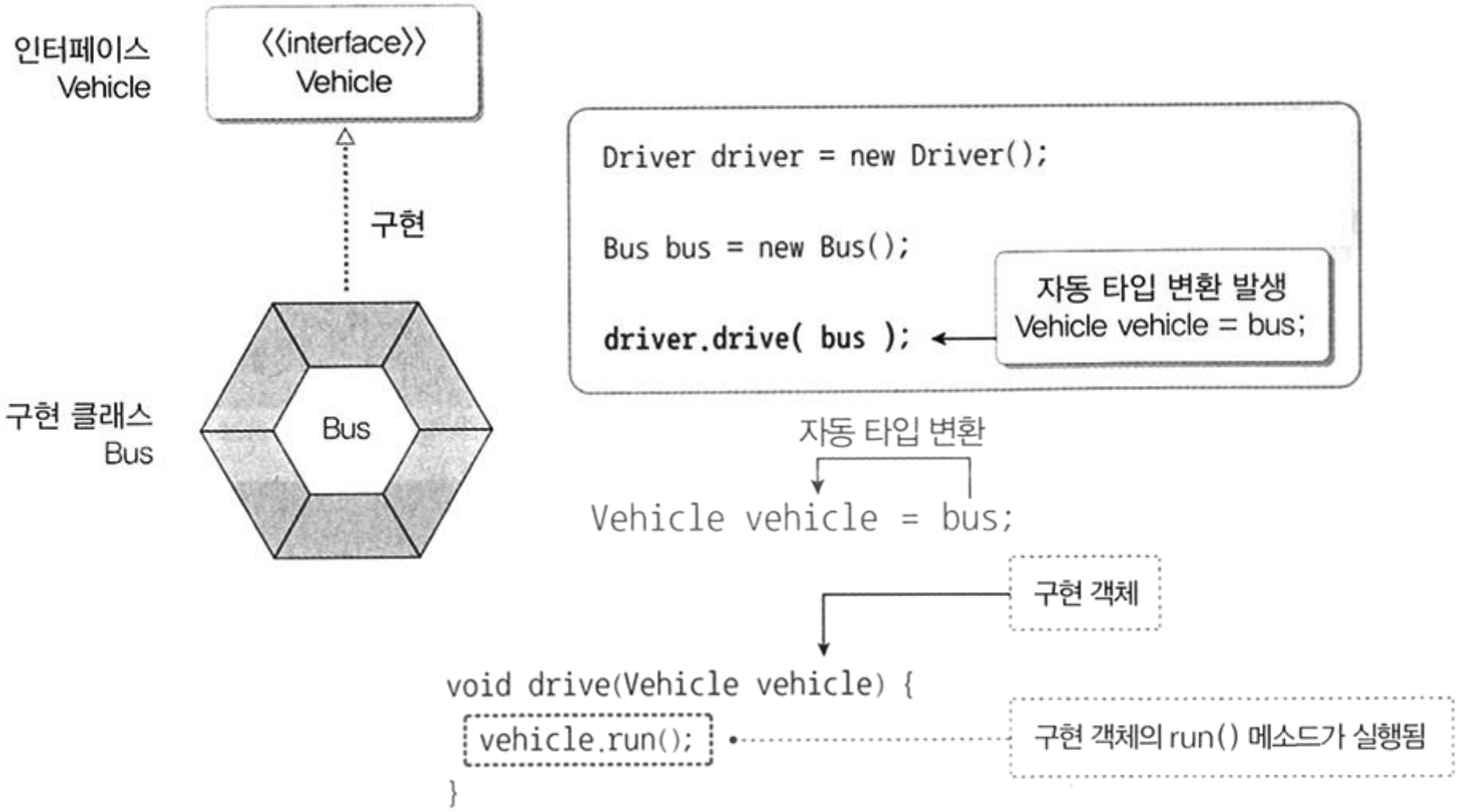
- 매개 변수의 타입이 인터페이스인 경우
  - 어떠한 구현 객체도 매개값으로 사용 가능
  - 구현 객체에 따라 메소드 실행결과 달라짐

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

public interface Vehicle {
      public void run();
}
```

❖ 매개변수의 다형성

- 매개 변수의 타입이 인터페이스인 경우



타입변환과 다형성

❖ 인터페이스: Driver.java

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

❖ 인터페이스: Vehicle.java

```
public interface Vehicle {
    public void run();
}
```

타입변환과 다형성

❖ 구현 클래스: Bus.java

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

❖ 구현 클래스: Taxi.java

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

❖ 매개 변수의 다형성 테스트: DriverExample.java

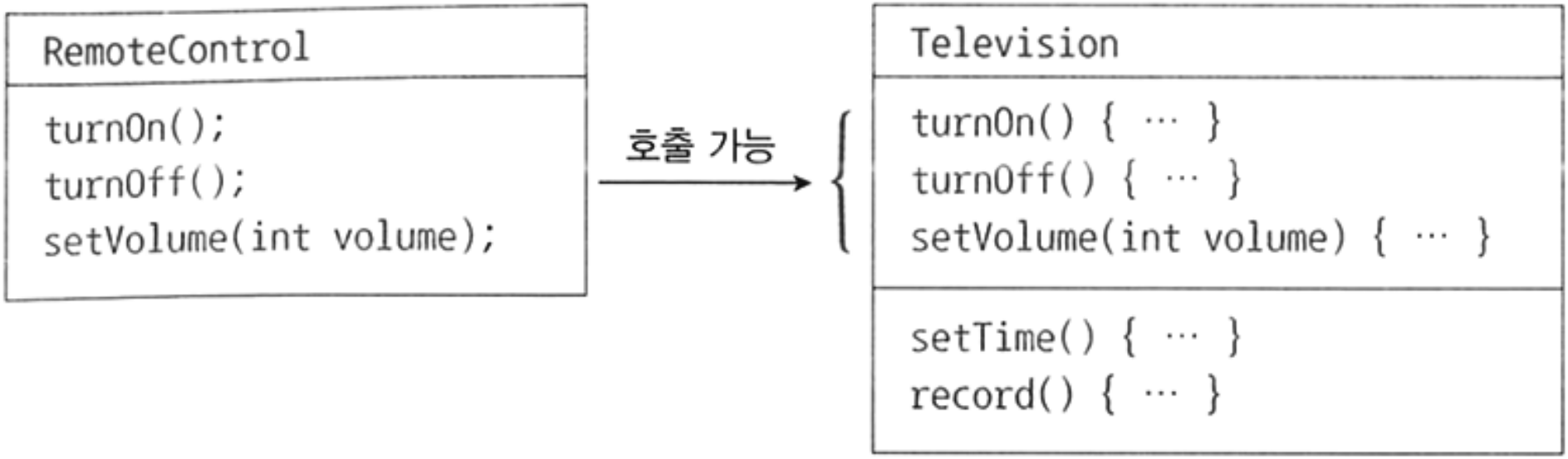
```
public class DriverExample {
    public static void main(String[] args) {
        Driver driver = new Driver();

        Bus bus = new Bus();
        Taxi taxi = new Taxi();

        driver.drive(bus);
        driver.drive(taxi);
    }
}
```

❖ 강제 타입 변환(Casting)

- 인터페이스 타입으로 자동 타입 변환 후, 구현 클래스 타입으로 변환
  - 필요성: 구현 클래스 타입에 선언된 다른 멤버 사용하기 위해



❖ 강제 타입 변환(Casting)

- 인터페이스 타입으로 자동 타입 변환 후, 구현 클래스 타입으로 변환

강제 타입 변환

```
구현클래스 변수 = (구현클래스) 인터페이스변수;
```

```
interface Vehicle {
  void run();
}
```

구현

```
class Bus implements Vehicle {
  void run() { … };
  void checkFare() { … }
}
```

```
Vehicle vehicle = new Bus();

vehicle.run();          (가능)
vehicle.checkFare();    (불가능)

Bus bus = (Bus) vehicle;  //강제타입변환

bus.run();              (가능)
bus.checkFare();        (가능)
```

타입변환과 다형성

❖ 인터페이스: Vehicle.java

```
public interface Vehicle {
    public void run();
}
```

❖ 구현 클래스: Bus.java

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

    public void checkFare() {
        System.out.println("승차요금을 체크합니다.");
    }
}
```

❖ 강제 타입 변환: VehicleExample.java

```
public class VehicleExample {
    public static void main(String[] args) {
        Vehicle vehicle = new Bus();

        vehicle.run();
        //vehicle.checkFare(); (x) – Vehicle 인터페이스에는 checkFare()가 없음

        Bus bus = (Bus) vehicle;  //강제타입변환

        bus.run();
        bus.checkFare();    // Bus 클래스에는 checkFare()가 있음
    }
}
```

❖ 객체 타입 확인(instanceof 연산자)

- 강제 타입 변환 전 구현 클래스 타입 조사
- 캐스팅이 불가능한 경우 ClassCastException 발생

```
Vehicle vehicle = new Taxi();
Bus bus = (Bus) vehicle;
```

```
public void drive(Vehicle vehicle) {
    Bus bus = (Bus) vehicle;
    bus.checkFare();
    vehicle.run();
}
```

```
if( vehicle instanceof Bus ) {
    Bus bus = (Bus) vehicle;
}
```

❖ 객체 타입 확인: Driver.java

```
public class Driver {
    public void drive(Vehicle vehicle) {
        if (vehicle instanceof Bus) {
            Bus bus = (Bus) vehicle;
            bus.checkFare();
        }
        vehicle.run();
    }
}
```

❖ 객체 타입 확인 테스트: DriverExample.java

```
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);
    }
}
```