

Chapter

07

상속



07-1. 상속

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 클래스 상속
- 부모 생성자 호출
- 메소드 재정의
- final 클래스와 final 메소드
- 키워드로 끝내는 핵심 포인트

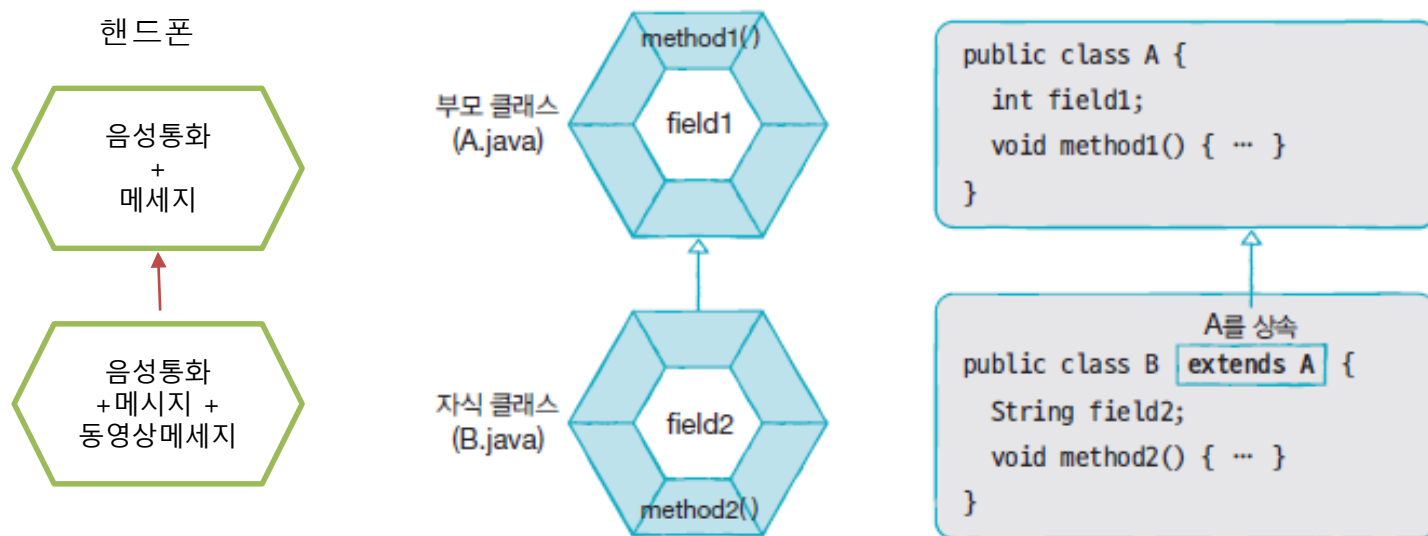


시작하기 전에

객체 지향 프로그램에서 부모 클래스의 멤버를 자식 클래스에게 물려줄 수 있다.

❖ 상속

- 이미 개발된 클래스를 재사용하여 새로운 클래스를 만들기에 중복되는 코드를 줄임
- 부모 클래스의 한번의 수정으로 모든 자식 클래스까지 수정되는 효과가 있어 유지보수 시간이 줄어듦



❖ 클래스 상속

- 자식 클래스 선언 시 부모 클래스 선택
- extends 뒤에 부모 클래스 기술

```
class 자식클래스 extends 부모클래스 {  
    //필드  
    //생성자  
    //메소드  
}
```

```
class SportsCar extends Car {  
}
```

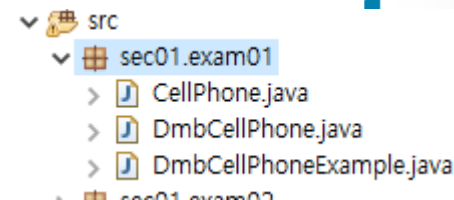
- 여러 개의 부모 클래스 상속할 수 없음
- 부모 클래스에서 private 접근 제한 갖는 필드와 메소드는 상속 대상에서 제외
- 부모와 자식 클래스가 다른 패키지에 존재할 경우 default 접근 제한된 필드와 메소드 역시 제외



클래스 상속

```
1 package sec01.exam01;
2
3 public class CellPhone {
4     //필드
5     String model;
6     String color;
7
8     //생성자
9
10    //메소드
11    void powerOn() { System.out.println("전원을 켭니다."); }
12    void powerOff() { System.out.println("전원을 끕니다."); }
13    void bell() { System.out.println("벨이 울립니다."); }
14    void sendVoice(String message) { System.out.println("자기: " + message); }
15    void receiveVoice(String message) { System.out.println("상대방: " + message); }
16    void hangUp() { System.out.println("전화를 끊습니다."); }
17 }
18
19
```

```
1 package sec01.exam01;
2
3 public class DmbCellPhone extends CellPhone {
4     //필드
5     int channel;
6
7     //생성자
8     DmbCellPhone(String model, String color, int channel) {
9         this.model = model;
10        this.color = color;
11        this.channel = channel;
12    }
13
14    //메소드
15    void turnOnDmb() {
16        System.out.println("채널 " + channel + "번 DMB 방송 수신을 시작합니다.");
17    }
18    void changeChannelDmb(int channel) {
19        this.channel = channel;
20        System.out.println("채널 " + channel + "번으로 바꿉니다.");
21    }
22    void turnOffDmb() {
23        System.out.println("DMB 방송 수신을 멈춥니다.");
24    }
25 }
26
```



클래스 상속

```
1 package sec01.exam01;
2
3 public class DmbCellPhoneExample {
4     public static void main(String[] args) {
5         //DmbCellPhone 객체 생성
6         DmbCellPhone dmbCellPhone = new DmbCellPhone("자바폰", "검정", 10);
7
8         //CellPhone으로부터 상속 받은 필드
9         System.out.println("모델: " + dmbCellPhone.model);
10        System.out.println("색상: " + dmbCellPhone.color);
11
12        //DmbCellPhone의 필드
13        System.out.println("채널: " + dmbCellPhone.channel);
14
15        //CellPhone으로부터 상속 받은 메소드 호출
16        dmbCellPhone.powerOn();
17        dmbCellPhone.bell();
18        dmbCellPhone.sendVoice("여보세요");
19        dmbCellPhone.receiveVoice("안녕하세요! 저는 홍길동인데요");
20        dmbCellPhone.sendVoice("아~ 예 반갑습니다.");
21        dmbCellPhone.hangUp();
22
23        //DmbCellPhone의 메소드 호출
24        dmbCellPhone.turnOnDmb();
25        dmbCellPhone.changeChannelDmb(12);
26        dmbCellPhone.turnOffDmb();
27    }
28 }
29 }
```

```
1 package sec01.exam01;
2
3 public class CellPhone {
4     //필드
5     String model;
6     String color;
7
8     //생성자
9
10    //메소드
11    void powerOn() { System.out.println("전원을 켭니다."); }
12    void powerOff() { System.out.println("전원을 끕니다."); }
13    void bell() { System.out.println("벨이 울립니다."); }
14    void sendVoice(String message) { System.out.println("자기: " + message); }
15    void receiveVoice(String message) { System.out.println("상대방: " + message); }
16    void hangUp() { System.out.println("전화를 끊습니다."); }
17 }
18
19 }
```

```
1 package sec01.exam01;
2
3 public class DmbCellPhone extends CellPhone {
4     //필드
5     int channel;
6
7     //생성자
8     DmbCellPhone(String model, String color, int channel) {
9         this.model = model;
10        this.color = color;
11        this.channel = channel;
12    }
13
14    //메소드
15    void turnOnDmb() {
16        System.out.println("채널 " + channel + "번 DMB 방송 수신을 시작합니다.");
17    }
18    void changeChannelDmb(int channel) {
19        this.channel = channel;
20        System.out.println("채널 " + channel + "번으로 바꿉니다.");
21    }
22    void turnOffDmb() {
23        System.out.println("DMB 방송 수신을 종료합니다.");
24    }
25 }
26 }
```

src

- sec01.exam01
 - CellPhone.java
 - DmbCellPhone.java
 - DmbCellPhoneExample.java

부모 생성자 호출

- ❖ 자식 객체 생성할 때 부모 객체가 먼저 생성되고 그 다음 자식 객체가 생성됨

```
DmbCellPhone dmbCellPhone = new DmbCellPhone();
```

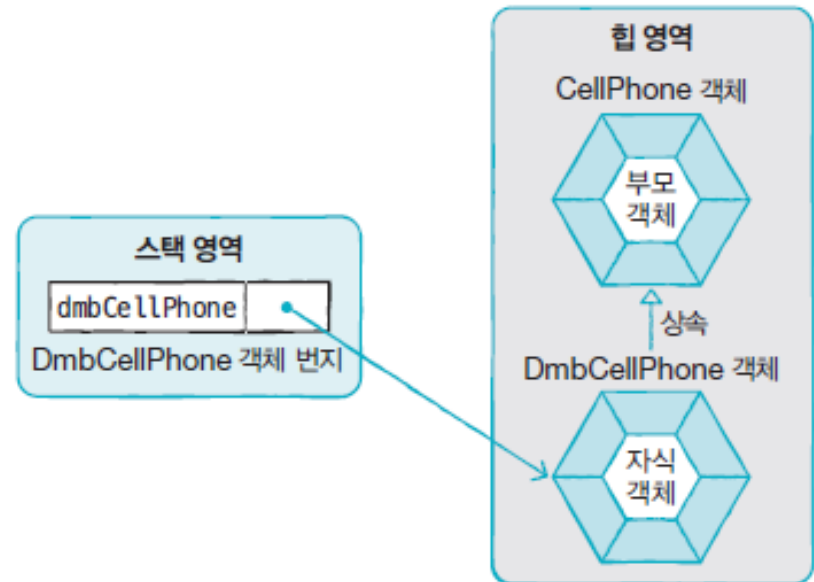
- 자식 생성자의 맨 첫 줄에서 부모 생성자가 호출됨

```
public DmbCellPhone() {  
    super();  
}
```

```
public CellPhone() {  
}
```

- 명시적으로 부모 생성자 호출하려는 경우

```
자식클래스( 매개변수선언, ... ) {  
    super( 매개값, ... );  
    ...  
}
```



부모 생성자 호출

0x200

```
1 package sec01.exam02;
2
3 public class People {
4     public String name;
5     public String ssn;
6
7     public People(String name, String ssn) {
8         this.name = name;
9         this.ssn = ssn;
10    }
11 }
```

```
1 package sec01.exam02;
2
3 public class Student extends People{
4     public int studentNo;
5
6     public Student(String name, String ssn, int studentNo) {
7         super(name, ssn);
8         this.studentNo = studentNo;
9     }
10 }
```

```
1 package sec01.exam02;
2
3 public class StudentExample {
4     public static void main(String[] args) {
5         Student student = new Student("홍길동", "123456-1234567", 1);
6         System.out.println("name : " + student.name);
7         System.out.println("ssn : " + student.ssn);
8         System.out.println("studentNo : " + student.studentNo);
9     }
10 }
```

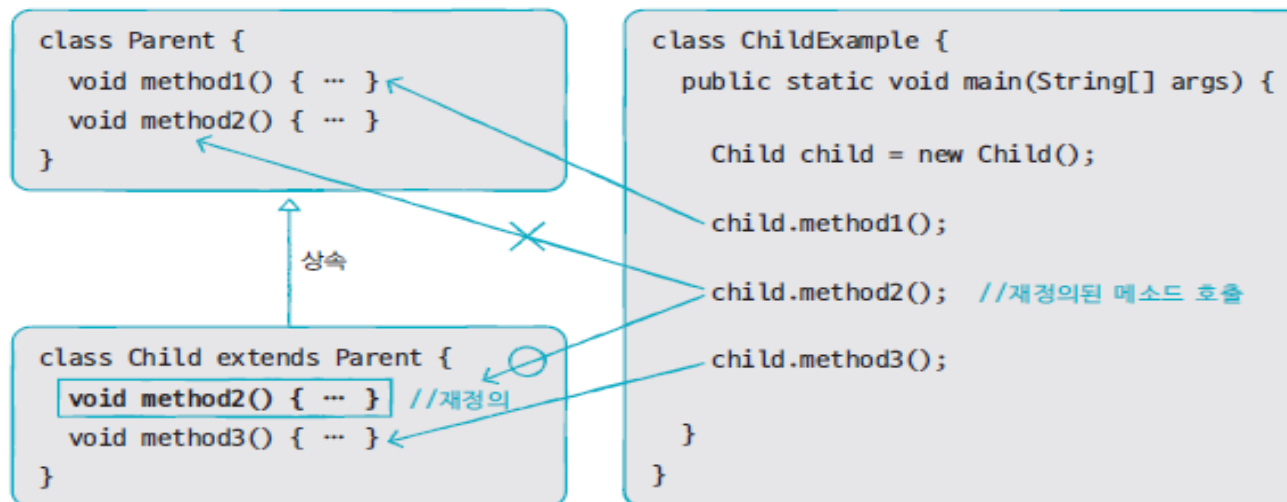
0x10

- sec01.exam02
 - People.java
 - Student.java
 - StudentExample.java

메소드 재정의

❖ 메소드 재정의 (오버라이딩 / Overriding)

- 부모 클래스의 메소드가 자식 클래스에서 사용하기에 부적합할 경우 자식 클래스에서 수정하여 사용
- 메소드 재정의 방법
 - 부모 메소드와 동일한 시그니처 가져야 함
 - 접근 제한 더 강하게 재정의할 수 없음
 - 새로운 예외를 throws 할 수 없음
- 메소드가 재정의될 경우 부모 객체 메소드가 숨겨지며, 자식 객체에서 메소드 호출하면 재정의된 자식 메소드가 호출됨



메소드 재정의

```
1 package sec01.exam03;
2
3 public class Calculator {
4     double areaCircle(double r) {
5         System.out.println("Calculator 객체의 areaCircle() 실행");
6         return 3.14159 * r * r;
7     }
8 }
```

```
1 package sec01.exam03;
2
3 public class Computer extends Calculator {
4     @Override
5     double areaCircle(double r) {
6         System.out.println("Computer 객체의 areaCircle() 실행");
7         return Math.PI * r * r;
8     }
9 }
```

```
1 package sec01.exam03;
2
3 public class ComputerExample {
4     public static void main(String[] args) {
5         int r = 10;
6         Calculator calculator = new Calculator();
7         System.out.println("원면적 : " + calculator.areaCircle(r));
8         System.out.println();
9         Computer computer = new Computer();
10        System.out.println("원면적 : " + computer.areaCircle(r));
11    }
12 }
```

- ✓ sec01.exam03
 - > Calculator.java
 - > Computer.java
 - > ComputerExample.java

메소드 재정의

■ 부모 메소드 호출

자식 클래스 내부에서 재정의된 부모 클래스 메소드 호출해야 하는 경우
명시적으로 super 키워드 붙여 부모 메소드 호출

```
super.부모메소드();
```

```
class Parent {  
    void method1() { ... }  
    void method2() { ... }  
}
```

상속

부모 메소드 호출

```
class Child extends Parent {  
    void method2() { ... } //재정의  
    void method3() {  
        method2();  
        super.method2();  
    }  
}
```

재정의된 호출



메소드 재정의

```
1 package sec01.exam04;
2
3 public class SupersonicAirplane extends Airplane {
4     public static final int NORMAL = 1;
5     public static final int SUPERSONIC = 2;
6
7     public int flyMode = NORMAL;
8
9     @Override
10    public void fly() {
11        if(flyMode == SUPERSONIC) {
12            System.out.println("초음속비행합니다.");
13        } else {
14            //Airplane 객체의 fly() 메소드 호출
15            super.fly();
16        }
17    }
18 }
```

```
1 package sec01.exam04;
2
3 public class SupersonicAirplaneExample {
4     public static void main(String[] args) {
5         SupersonicAirplane sa = new SupersonicAirplane();
6         sa.takeOff();
7         sa.fly();
8         sa.flyMode = SupersonicAirplane.SUPERSONIC;
9         sa.fly();
10        sa.flyMode = SupersonicAirplane.NORMAL;
11        sa.fly();
12        sa.land();
13    }
14 }
```

```
1 package sec01.exam04;
2
3 public class Airplane {
4     public void land() {
5         System.out.println("착륙합니다.");
6     }
7     public void fly() {
8         System.out.println("일반비행합니다.");
9     }
10    public void takeOff() {
11        System.out.println("이륙합니다.");
12    }
13 }
14
```

▼ sec01.exam04

- > Airplane.java
- > SupersonicAirplane.java
- > SupersonicAirplaneExample.java

final 클래스와 final 메소드

❖ final 키워드

- 해당 선언이 최종 상태이며 수정될 수 없음을 의미
- 클래스 및 메소드 선언 시 final 키워드를 사용하면 상속과 관련됨

❖ 상속할 수 없는 final 클래스

- 부모 클래스가 될 수 없어 자식 클래스 만들 수 없음을 의미

```
public final class 클래스 { ... }
```

```
public final class String { ... }
```


```
public class NewString extends String { ... }
```



final 클래스와 final 메소드

```
1 package sec01.exam05;  
2  
3 public final class Member {  
4 }  
5
```

```
1 package sec01.exam05;  
2  
3 //public class VeryVeryImportantPerson extends Member {  
4 public class VeryImportantPerson {  
5 }  
6
```



- sec01.exam05
 - Member.java
 - VeryImportantPerson.java

final 클래스와 final 메소드

❖ 재정의할 수 없는 final 메소드

- 부모 클래스에 선언된 final 메소드는 자식 클래스에서 재정의 할 수 없음

```
public final 리턴타입 메소드( [매개변수, ...] ) { ... }
```

```
1 package sec01.exam06;
2
3 public class Car {
4     //필드
5     public int speed;
6
7     //메소드
8     public void speedUp() {
9         speed += 1;
10    }
11
12    //final 메소드
13    public final void stop() {
14        System.out.println("차를 멈춤");
15        speed = 0;
16    }
17 }
18
```

```
1 package sec01.exam06;
2
3 public class SportsCar extends Car {
4     @Override
5     public void speedUp() {
6         speed += 10;
7     }
8
9     //오버라이딩을 할 수 없음
10    /*
11    @Override
12    public void stop() {
13        System.out.println("스포츠카를 멈춤");
14        speed = 0;
15    }
16    */
17 }
18
```


키워드로 끝내는 핵심 포인트

- **상속**: 부모 클래스의 필드와 메소드를 자식 클래스에서 사용할 수 있도록 한다.
- **메소드 재정의**: 부모 메소드를 자식 클래스에서 다시 정의하는 것을 의미한다.
- **final 클래스**: final 클래스는 부모 클래스로 사용할 수 없다.
- **final 메소드**: 자식 클래스에서 재정의할 수 없는 메소드이다.



Chapter

07

상속



07-2. 타입 변환과 다형성

혼자 공부하는 자바 (신용권 저)

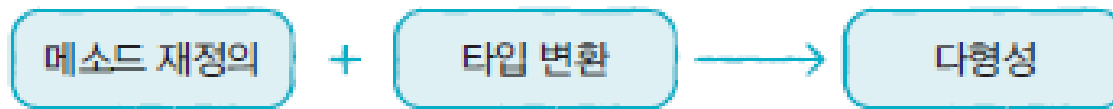
- 시작하기 전에
- 자동 타입 변환
- 필드의 다형성
- 매개변수의 다형성
- 강제 타입 변환
- 객체 타입 확인
- 키워드로 끝내는 핵심 포인트



기본 타입과 마찬가지로 클래스도 타입 변환이 있다.
이를 활용하면 객체 지향 프로그래밍의 다형성을 구현할 수 있다.

❖ 다형성

- 사용 방법은 동일하지만 다양한 객체 활용해 여러 실행결과가 나오도록 하는 성질
- 메소드 재정의와 타입 변환으로 구현



자동 타입 변환

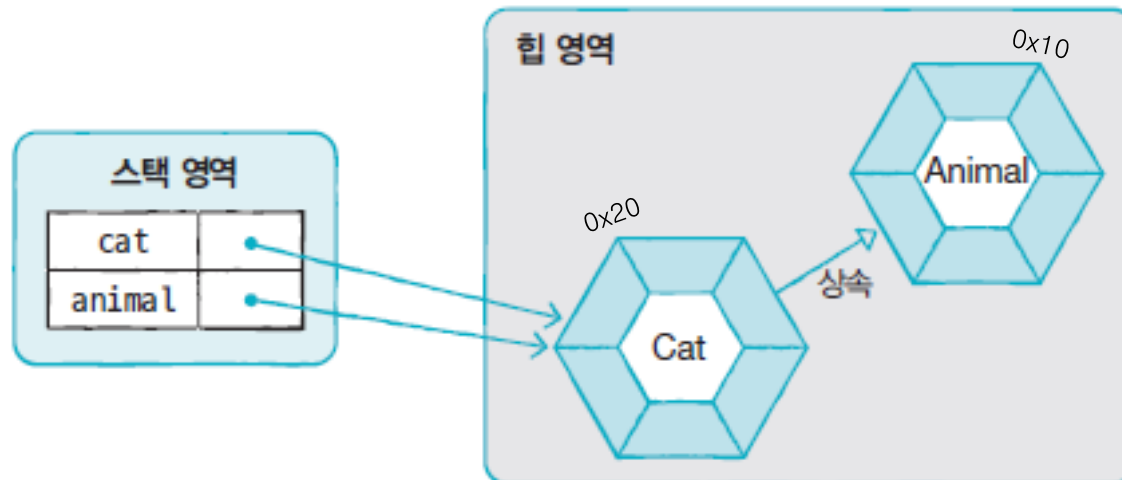
❖ 자동 타입 변환 (promotion)

- 프로그램 실행 도중 자동으로 타입 변환 일어나는 것

자동 타입 변환
↓
부모타입 변수 = 자식타입;

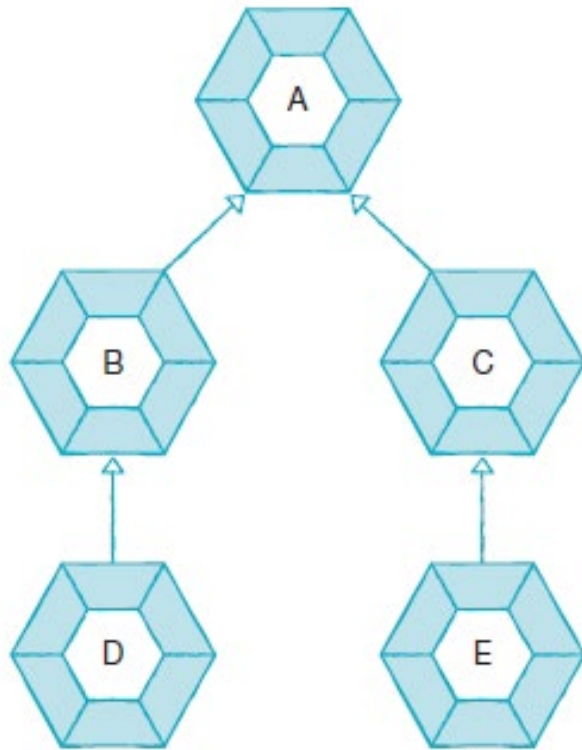
```
Cat cat = new Cat();  
Animal animal = cat;
```

← Animal animal = new Cat(); 도 가능



자동 타입 변환

- 바로 위 부모가 아니더라도 상속 계층에서 상위 타입인 경우 자동 타입 변환 일어날 수 있음



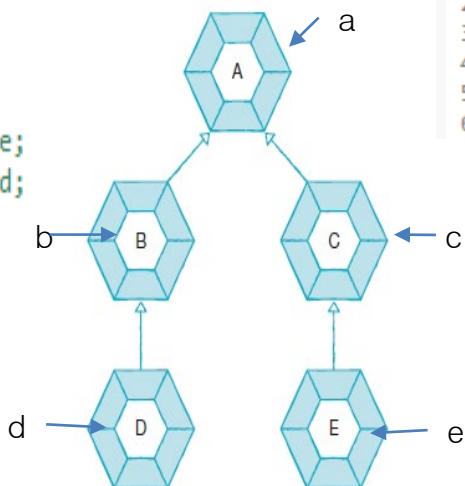
```
B b = new B( );  
C c = new C( );  
D d = new D( );  
E e = new E( );
```

```
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)
```



자동 타입 변환

```
1 package sec02.exam01;
2
3 public class PromotionExample {
4     public static void main(String[] args) {
5         B b = new B();
6         C c = new C();
7         D d = new D();
8         E e = new E();
9
10        A a1 = b;
11        A a2 = c;
12        A a3 = d;
13        A a4 = e;
14
15        B b1 = d;
16        C c1 = e;
17
18        //B b3 = e;
19        //C c2 = d;
20    }
21 }
22
```



```
1 package sec02.exam01;
2
3 public class D extends B {
4
5 }
6
```

```
1 package sec02.exam01;
2
3 public class E extends C {
4
5 }
6
```

```
1 package sec02.exam01;
2
3 public class C extends A {
4
5 }
6
```

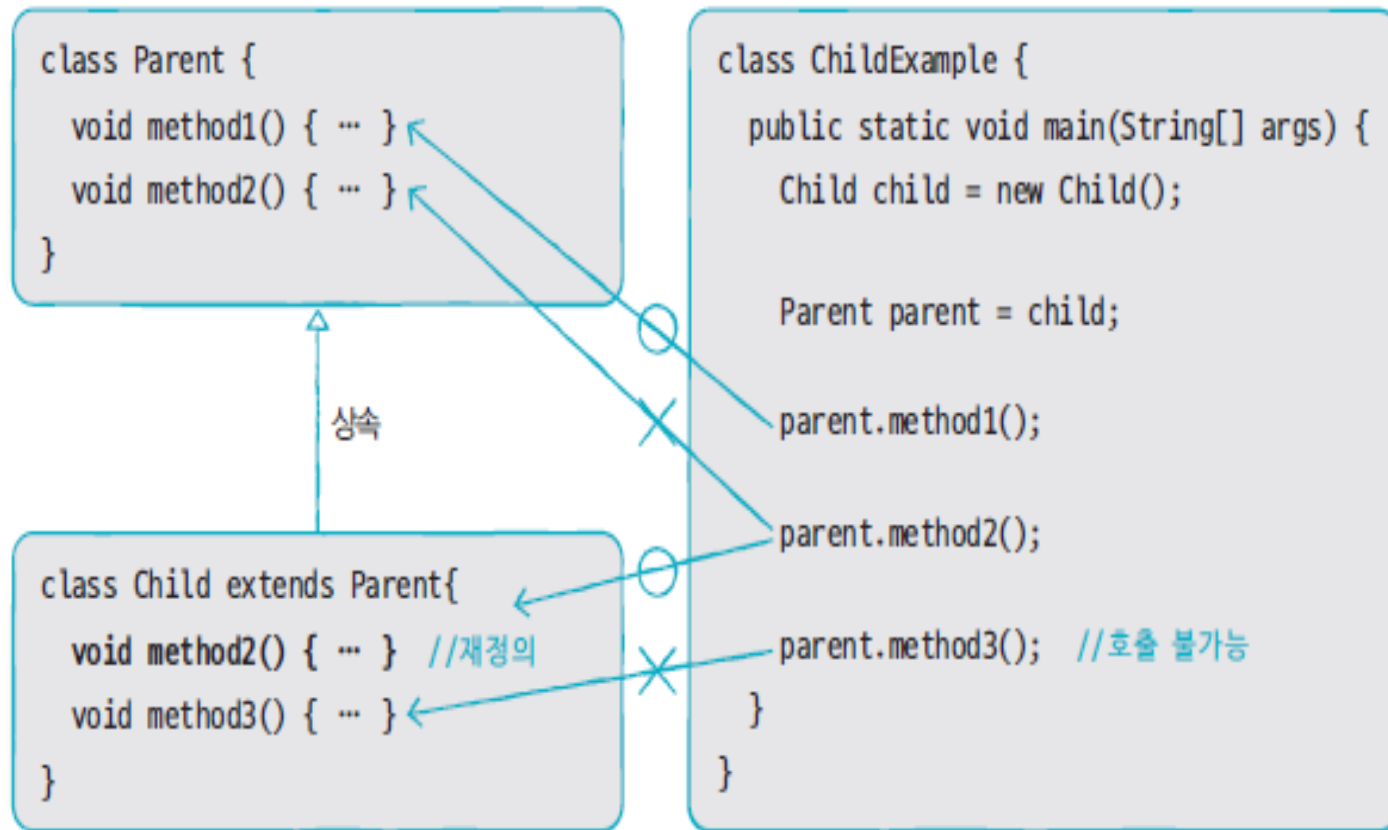
```
1 package sec02.exam01;
2
3 public class A {
4
5 }
6
```

```
1 package sec02.exam01;
2
3 public class B extends A {
4
5 }
6
```

- sec02.exam01
 - A.java
 - B.java
 - C.java
 - D.java
 - E.java
 - PromotionExample.java

자동 타입 변환

- 부모 타입으로 자동 타입 변환 이후에는 부모 클래스에 선언된 필드 및 메소드만 접근 가능
- 예외적으로, 메소드가 자식 클래스에서 재정의될 경우 자식 클래스의 메소드가 대신 호출



자동 타입 변환

ox200

```
1 package sec02.exam02;
2
3 public class Parent {
4     public void method1() {
5         System.out.println("Parent-method1()");
6     }
7
8     public void method2() {
9         System.out.println("Parent-method2()");
10    }
11 }
12
```

```
1 package sec02.exam02;
2
3 public class Child extends Parent {
4     @Override
5     public void method2() {
6         System.out.println("Child-method2()");
7     }
8
9     public void method3() {
10        System.out.println("Child-method3()");
11    }
12 }
13
```

```
1 package sec02.exam02;
2
3 public class ChildExample {
4     public static void main(String[] args) {
5         Child child = new Child();
6
7         Parent parent = child;
8
9         parent.method1();
10        parent.method2();
11
12        //parent.method3(); (호출 불가능)
13
14    }
15 }
```

ox10

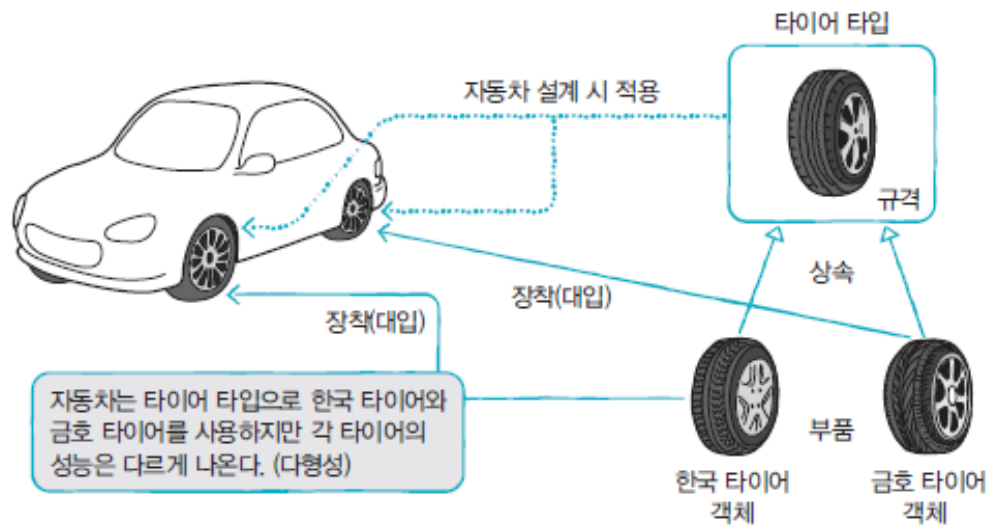
- sec02.exam02
 - Child.java
 - ChildExample.java
 - Parent.java

❖ 필드의 다형성

- 필드 타입을 부모 타입으로 선언할 경우

다양한 자식 객체가 저장되어 필드 사용 결과 달라질 수 있음

```
class Car {  
    //필드  
    Tire frontLeftTire = new Tire();  
    Tire frontRightTire = new Tire();  
    Tire backLeftTire = new Tire();  
    Tire backRightTire = new Tire();  
    //메소드  
    void run() {  
        frontLeftTire.roll();  
        frontRightTire.roll();  
        backLeftTire.roll();  
        backRightTire.roll();  
    }  
}
```



```
Car myCar = new Car();  
myCar.frontRightTire = new HankookTire();  
myCar.backLeftTire = new KumhoTire();  
myCar.run();
```



```

1 package sec02.exam03;
2
3 public class HankookTire extends Tire {
4     //필드
5     //생성자
6     public HankookTire(String location, int maxRotation) {
7         super(location, maxRotation);
8     }
9     //메소드
10    @Override
11    public boolean roll() {
12        ++accumulatedRotation;
13        if(accumulatedRotation < maxRotation) {
14            System.out.println(location + " HankookTire 수명: " + (maxRotation - accumulatedRotation) + "회");
15            return true;
16        } else {
17            System.out.println("*** " + location + " HankookTire 펑크 ***");
18            return false;
19        }
20    }
21 }

```

```

1 package sec02.exam03;
2
3 public class KumhoTire extends Tire {
4     //필드
5     //생성자
6     public KumhoTire(String location, int maxRotation) {
7         super(location, maxRotation);
8     }
9     //메소드
10    @Override
11    public boolean roll() {
12        ++accumulatedRotation;
13        if(accumulatedRotation < maxRotation) {
14            System.out.println(location + " KumhoTire 수명: " + (maxRotation - accumulatedRotation) + "회");
15            return true;
16        } else {
17            System.out.println("*** " + location + " KumhoTire 펑크 ***");
18            return false;
19        }
20    }
21 }

```

car.frontRightTire = new KumhoTire("앞오른쪽", 13);

```

1 package sec02.exam03;
2
3 public class Tire {
4     //필드
5     public int maxRotation;           //최대 회전
6     public int accumulatedRotation;   //누적 회전수
7     public String location;           //타이어의
8
9     //생성자
10    public Tire(String location, int maxRotation) {
11        this.location = location;
12        this.maxRotation = maxRotation;
13    }
14
15    //메소드
16    public boolean roll() {
17        ++accumulatedRotation;

```

- sec02.exam03
 - > Car.java
 - > CarExample.java
 - > HankookTire.java
 - > KumhoTire.java
 - > Tire.java

필드의 다형성

```
1 package sec02.exam03;
2
3 public class Tire {
4     //필드
5     public int maxRotation;           //최대 회전수(최대 수명)
6     public int accumulatedRotation;   //누적 회전수
7     public String location;           //타이어의 위치
8
9     //생성자
10    public Tire(String location, int maxRotation) {
11        this.location = location;
12        this.maxRotation = maxRotation;
13    }
14
15    //메소드
16    public boolean roll() {
17        ++accumulatedRotation;
18        if(accumulatedRotation < maxRotation) {
19            System.out.println(location + " Tire 수명: " + (maxRotation - accumulatedRotation) + "회");
20            return true;
21        } else {
22            System.out.println("*** " + location + " Tire 펑크 ***");
23            return false;
24        }
25    }
26 }
27
28
```

```
1 package sec02.exam03;
2
3 public class Car {
4     //필드
5     Tire frontLeftTire = new Tire("앞왼쪽", 6);
6     Tire frontRightTire = new Tire("앞오른쪽", 2);
7     Tire backLeftTire = new Tire("뒤왼쪽", 3);
8     Tire backRightTire = new Tire("뒤오른쪽", 4);
9
10
```

- sec02.exam03
 - Car.java
 - CarExample.java
 - HankookTire.java
 - KumhoTire.java
 - Tire.java

필드의 다형성

```
1 package sec02.exam03;
2
3 public class Car {
4     //필드
5     Tire frontLeftTire = new Tire("앞왼쪽", 6);
6     Tire frontRightTire = new Tire("앞오른쪽", 2);
7     Tire backLeftTire = new Tire("뒤왼쪽", 3);
8     Tire backRightTire = new Tire("뒤오른쪽", 4);
9
10    //생성자
11
12    //메소드
13    int run() {
14        System.out.println("[자동차가 달립니다.]");
15        if(frontLeftTire.roll()==false) { stop(); return 1; };
16        if(frontRightTire.roll()==false) { stop(); return 2; };
17        if(backLeftTire.roll()==false) { stop(); return 3; };
18        if(backRightTire.roll()==false) { stop(); return 4; };
19        return 0;
20    }
21
22    void stop() {
23        System.out.println("[자동차가 멈춥니다.]");
24    }
25 }
26
27
```

0x10 frontleft 0x80 fronringt 0x120 backleft 0x180 backright

```
1 package sec02.exam03;
2
3 public class CarExample {
4     public static void main(String[] args) {
5         Car car = new Car();
6
7         for(int i=1; i<=5; i++) {
8             int problemLocation = car.run();
9             switch(problemLocation) {
10                 case 1:
11                     System.out.println("앞왼쪽 HankookTire로 교체");
12                     car.frontLeftTire = new HankookTire("앞왼쪽", 15);
13                     break;
14                 case 2:
15                     System.out.println("앞오른쪽 KumhoTire로 교체");
16                     car.frontRightTire = new KumhoTire("앞오른쪽", 13);
17                     break;
18                 case 3:
19                     System.out.println("뒤왼쪽 HankookTire로 교체");
20                     car.backLeftTire = new HankookTire("뒤왼쪽", 14);
21                     break;
22                 case 4:
23                     System.out.println("뒤오른쪽 KumhoTire로 교체");
24                     car.backRightTire = new KumhoTire("뒤오른쪽", 17);
25                     break;
26             }
27             System.out.println("-----");
28         }
29     }
30 }
```

sec02.exam03

- > Car.java
- > CarExample.java
- > HankookTire.java
- > KumhoTire.java
- > Tire.java

매개 변수의 다형성

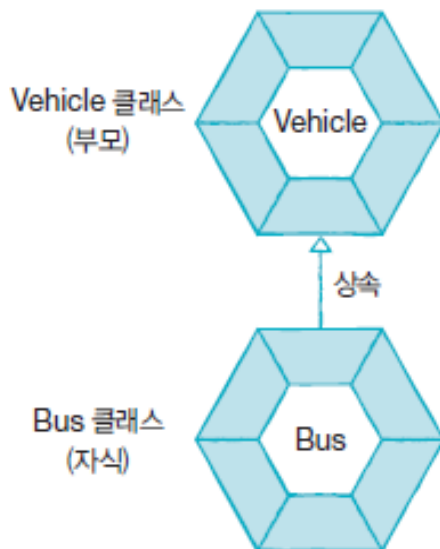
❖ 매개 변수의 다형성

- 매개 변수를 부모 타입으로 선언하는 효과

메소드 호출 시 매개값으로 부모 객체 및 모든 자식 객체를 제공할 수 있음
자식의 재정의된 메소드가 호출 -> 다형성

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

```
Driver driver = new Driver();  
Vehicle vehicle = new Vehicle();  
driver.drive(vehicle);
```



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

자동 타입 변환 발생
Vehicle vehicle = bus;



매개 변수의 다형성

```
1 package sec02.exam04;
2
3 public class Vehicle {
4     public void run() {
5         System.out.println("차량이 달립니다.");
6     }
7 }
8
```

```
1 package sec02.exam04;
2
3 public class Bus extends Vehicle {
4     @Override
5     public void run() {
6         System.out.println("버스가 달립니다.");
7     }
8 }
9
```

```
1 package sec02.exam04;
2
3 public class Taxi extends Vehicle {
4     @Override
5     public void run() {
6         System.out.println("택시가 달립니다.");
7     }
8 }
9
```

```
1 package sec02.exam04;
2
3 public class DriverExample {
4     public static void main(String[] args) {
5         Driver driver = new Driver();
6
7         Bus bus = new Bus();
8         Taxi taxi = new Taxi();
9
10        driver.drive(bus);
11        driver.drive(taxi);
12    }
13 }
14
```

```
1 package sec02.exam04;
2
3 public class Driver {
4     public void drive(Vehicle vehicle) {
5         vehicle.run();
6     }
7 }
8
```

- sec02.exam04
 - Bus.java
 - Driver.java
 - DriverExample.java
 - Taxi.java
 - Vehicle.java

❖ 강제 타입 변환 (casting)

- 부모 타입을 자식 타입으로 변환

조건: 자식 타입이 부모 타입으로 자동 타입 변환한 후 다시 반대로 변환할 때 사용

자식타입 변수 = (자식타입) 부모타입;
부모 타입을 자식 타입으로 변환

```
Parent parent = new Child(); //자동 타입 변환  
Child child = (Child) parent; //강제 타입 변환
```

```
class Parent {  
    String field1;  
    void method1() { ... }  
    void method2() { ... }  
}
```

상속

```
class Child extends Parent {  
    String field2;  
    void method3() { ... }  
}
```

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

강제 타입 변환

```
1 package sec02.exam05;
2
3 public class Child extends Parent {
4     public String field2;
5
6     public void method3() {
7         System.out.println("Child-method3()");
8     }
9 }
10
```

```
1 package sec02.exam05;
2
3 public class Parent {
4     public String field1;
5
6     public void method1() {
7         System.out.println("Parent-method1()");
8     }
9
10    public void method2() {
11        System.out.println("Parent-method2()");
12    }
13 }
14
```

```
1 package sec02.exam05;
2
3 public class ChildExample {
4     public static void main(String[] args) {
5         Parent parent = new Child();
6         parent.field1 = "data1";
7         parent.method1();
8         parent.method2();
9         /*
10         parent.field2 = "data2"; //(불가능)
11         parent.method3();         //(불가능)
12         */
13
14         Child child = (Child) parent;
15         child.field2 = "yyy"; //(가능)
16         child.method3();      //(가능)
17     }
18 }
19
```

sec02.exam05

- Child.java
- ChildExample.java
- Parent.java

객체 타입 확인

❖ instanceof 연산자

- 어떤 객체가 어느 클래스의 인스턴스인지 확인

- 메소드 내 강제 타입 변환 필요한 경우

타입 확인하지 않고 강제 타입 변환 시도 시 `ClassCastException` 발생할 수 있음
instanceof 연산자 통해 확인 후 안전하게 실행

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

```
Parent parent = new Parent();
```

```
Child child = (Child) parent;    //강제 타입 변환을 할 수 없음
```

```
      Parent      Child  
      객체       객체  
public void method(Parent parent) {  
    if(parent instanceof Child) { ← Parent 매개 변수가 참조하는  
        Child child = (Child) parent;      객체가 Child인지 조사  
    }  
}
```



객체 타입 확인

```
1 package sec02.exam06;
2
3 public class InstanceofExample {
4     public static void method1(Parent parent) {
5         if(parent instanceof Child) {
6             Child child = (Child) parent;
7             System.out.println("method1 - Child로 변환 성공");
8         } else {
9             System.out.println("method1 - Child로 변환되지 않음");
10        }
11    }
12
13    public static void method2(Parent parent) {
14        Child child = (Child) parent;
15        System.out.println("method2 - Child로 변환 성공");
16    }
17
18    public static void main(String[] args) {
19        Parent parentA = new Child();
20        method1(parentA);
21        method2(parentA);
22
23        Parent parentB = new Parent();
24        method1(parentB);
25        method2(parentB); //예외 발생
26    }
27 }
28
```

```
1 package sec02.exam06;
2
3 public class Parent {
4 }
5
```

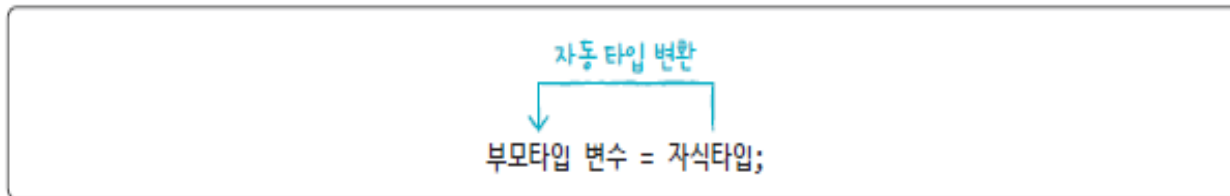
```
1 package sec02.exam06;
2
3 public class Child extends Parent {
4 }
5
```

▼ sec02.exam06

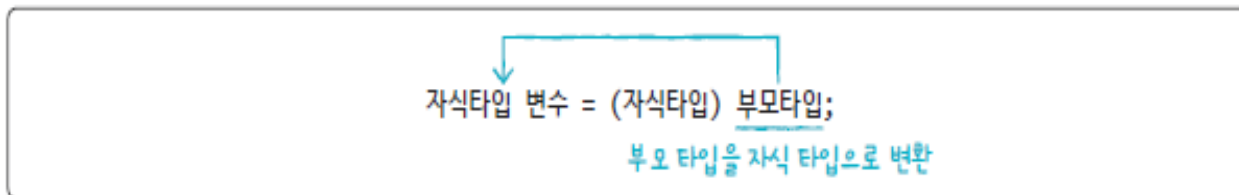
- > Child.java
- > InstanceofExample.java
- > Parent.java

키워드로 끝내는 핵심 포인트

- **클래스 타입 변환** : 다른 클래스 타입으로 객체를 대입
- **자동 타입 변환** : 자식 객체를 부모 타입 변수에 대입할 때에는 자동으로 타입이 변환됨



- **강제 타입 변환** : 부모 타입 객체를 다시 자식 타입에 대입할 때 강제 타입 변환일 필요



- **instanceof 연산자** : 객체가 어떤 타입인지 조사할 때 instanceof 연산자 사용.
- **다형성** : 객체 사용 방법은 동일하나 실행결과가 다양하게 나오는 성질.
메소드 재정의와 타입 변환으로 구현.



Chapter

07

상속



07-3. 추상 클래스

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 추상 클래스의 용도
- 추상 클래스 선언
- 추상 메소드와 재정의
- 키워드로 끝내는 핵심 포인트

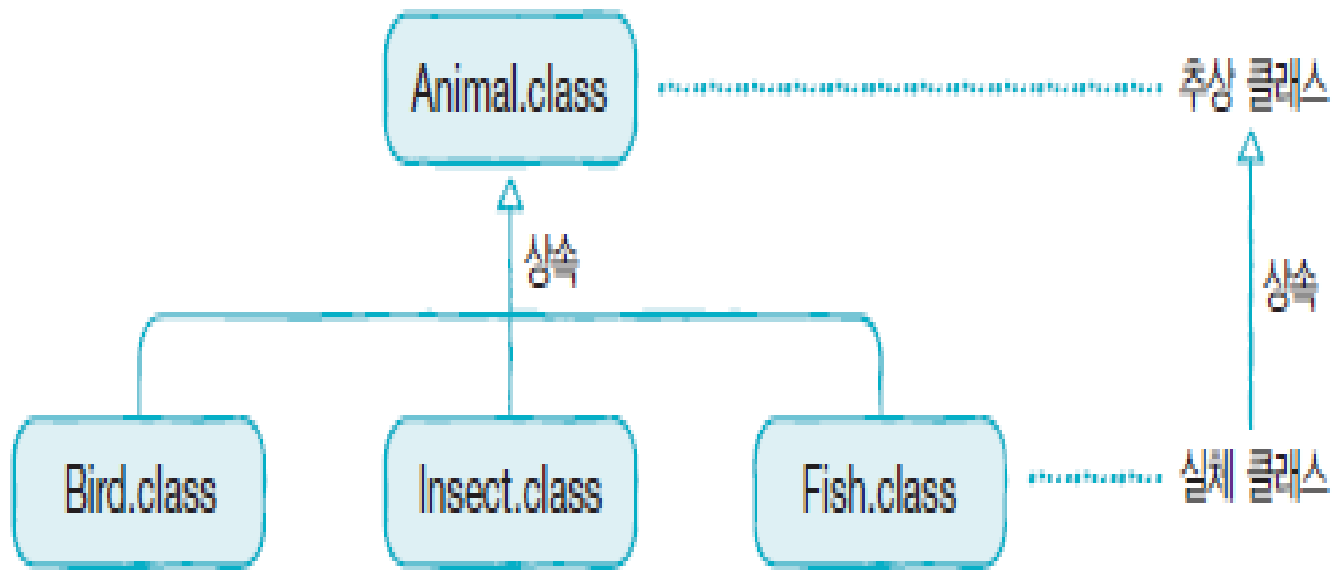


시작하기 전에

여러 클래스의 공통된 특성(필드, 메소드)를 추출해서 선언한 것을 추상 클래스라고 한다.

❖ 추상 클래스

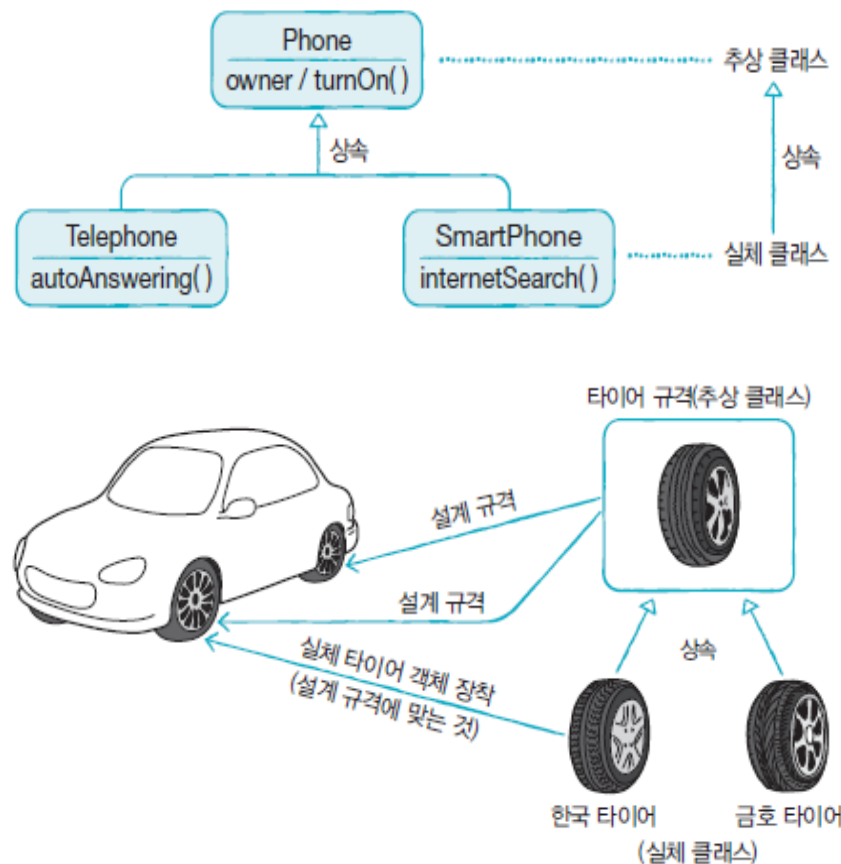
- 실체 클래스(객체 생성용 클래스)들의 공통적인 특성(필드, 메소드)을 추출하여 선언한 것
- 추상 클래스와 실체 클래스는 부모, 자식 클래스로서 상속 관계를 가짐



추상 클래스의 용도

❖ 추상 클래스의 용도

- 실체 클래스에 반드시 존재해야 할 필드와 메소드의 선언(실체 클래스의 설계 규격 - 객체 생성용이 아님)
- 실체 클래스에는 공통된 내용은 빠르게 물려받고, 다른 점만 선언하면 되므로 시간 절약



추상 클래스 선언

❖ 추상 클래스 선언

■ abstract 키워드

상속 통해 자식 클래스만 만들 수 있게 만듦(부모로서의 역할만 수행)

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

- 추상 클래스도 일반 클래스와 마찬가지로 필드, 생성자, 메소드 선언 할 수 있음
- 직접 객체를 생성할 수 없지만 자식 객체 생성될 때 객체화 됨.
자식 생성자에서 super(...) 형태로 추상 클래스의 생성자 호출



추상 클래스 선언

```
1 package sec03.exam01;
2
3 public class SmartPhone extends Phone {
4     //생성자
5     public SmartPhone(String owner) {
6         super(owner);
7     }
8     //메소드
9     public void internetSearch() {
10         System.out.println("인터넷 검색을 합니다.");
11     }
12 }
13
```

```
1 package sec03.exam01;
2
3 public class PhoneExample {
4     public static void main(String[] args) {
5         //Phone phone = new Phone(); (x)
6
7         SmartPhone smartPhone = new SmartPhone("홍길동");
8
9         smartPhone.turnOn();
10        smartPhone.internetSearch();
11        smartPhone.turnOff();
12    }
13 }
14
```

```
1 package sec03.exam01;
2
3 public abstract class Phone {
4     //필드
5     public String owner;
6
7     //생성자
8     public Phone(String owner) {
9         this.owner = owner;
10    }
11
12    //메소드
13    public void turnOn() {
14        System.out.println("폰 전원을 켭니다.");
15    }
16    public void turnOff() {
17        System.out.println("폰 전원을 끕니다.");
18    }
19 }
20
```

sec03.exam01

- Phone.java
- PhoneExample.java
- SmartPhone.java

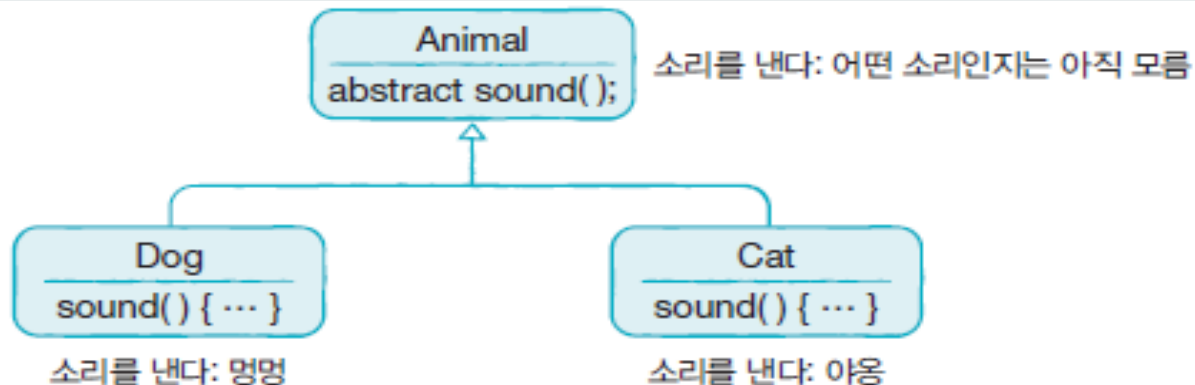
추상 메소드와 재정의

❖ 추상 메소드

- 메소드 선언만 통일하고 실행 내용은 실제 클래스마다 달라야 하는 경우
- abstract 키워드로 선언되고 중괄호가 없는 메소드
- 하위 클래스는 반드시 재정의해서 실행 내용을 채워야 함.

```
[public | protected] abstract 리턴타입 메소드이름(매개변수, ...);
```

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



추상 메소드와 재정의

```
1 package sec03.exam02;
2
3 public class Dog extends Animal {
4     public Dog() {
5         this.kind = "포유류";
6     }
7
8     @Override
9     public void sound() {
10         System.out.println("멍멍");
11     }
12 }
```

```
1 package sec03.exam02;
2
3 public class AnimalExample {
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         //변수의 자동 타입 변환
12         Animal animal = null;
13         animal = new Dog();
14         animal.sound();
15         animal = new Cat();
16         animal.sound();
17         System.out.println("-----");
18     }
19 }
```

```
1 package sec03.exam02;
2
3 public class Cat extends Animal {
4     public Cat() {
5         this.kind = "포유류";
6     }
7
8     @Override
9     public void sound() {
10         System.out.println("야옹");
11     }
12 }
13
```

```
1 package sec03.exam02;
2
3 public abstract class Animal {
4     public String kind;
5
6     public void breathe() {
7         System.out.println("숨을 쉽니다.");
8     }
9
10    public abstract void sound();
11 }
12
13
```

```
18
19     //매개변수의 자동 타입 변환
20     animalSound(new Dog());
21     animalSound(new Cat());
22 }
23
24 public static void animalSound(Animal animal) {
25     animal.sound();
26 }
27 }
28
```

- sec03.exam02
 - Animal.java
 - AnimalExample.java
 - Cat.java
 - Dog.java

키워드로 끝내는 핵심 포인트

- **추상 클래스**: 클래스들의 공통적인 필드와 메소드 추출하여 선언한 클래스
- **추상 메소드** :
추상 클래스에서만 선언할 수 있고, 메소드의 선언부만 있는 메소드.
자식 클래스에서 재정의되어 실행 내용 결정해야 함



Chapter

08

인터페이스



08-1. 인터페이스

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 인터페이스 선언
- 인터페이스 구현
- 인터페이스 사용
- 키워드로 끝내는 핵심 포인트



시작하기 전에

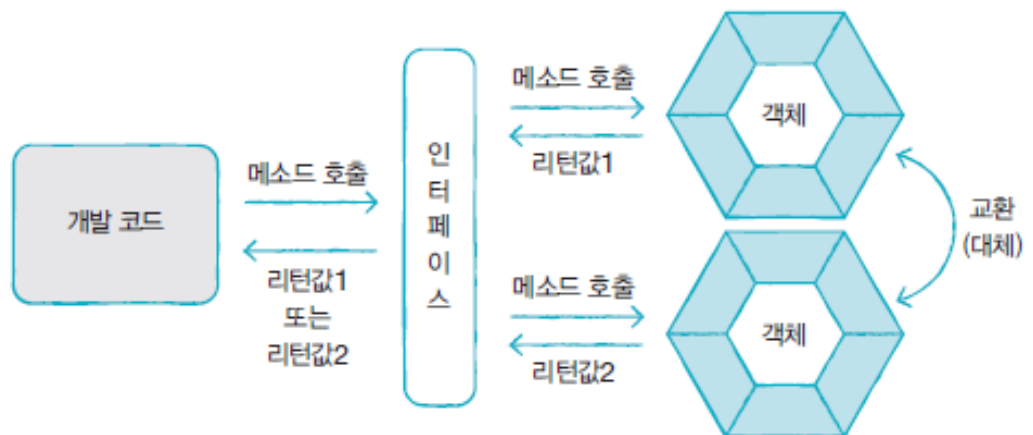
인터페이스란 객체의 사용 방법을 정의한 타입이다.

인터페이스를 통해 다양한 객체를 동일한 사용 방법으로 이용할 수 있다.

인터페이스를 이용해서 다형성을 구현할 수 있다.

❖ 인터페이스 (interface)

- 개발 코드는 인터페이스를 통해서 객체와 서로 통신한다.
- 인터페이스의 메소드 호출하면 객체의 메소드가 호출된다.
- 개발 코드를 수정하지 않으면서 객체 교환이 가능하다.



인터페이스 선언

❖ 인터페이스 선언

- ~.java 형태 소스 파일로 작성 및 컴파일러 통해 ~class 형태로 컴파일된다.
- 클래스와 물리적 파일 형태는 같으나 소스 작성 내용이 다르다.

```
[public] interface 인터페이스이름 { ... }
```

- 인터페이스는 객체로 생성할 수 없으므로 생성자 가질 수 없다.

```
interface 인터페이스이름 {  
    //상수  
    타입 상수이름 = 값;  
    //추상 메소드  
    타입 메소드이름(매개변수,...);  
}
```



인터페이스 선언

❖ 상수 필드 (constant field) 선언

- 데이터를 저장할 인스턴스 혹은 정적 필드 선언 불가
- 상수 필드만 선언 가능

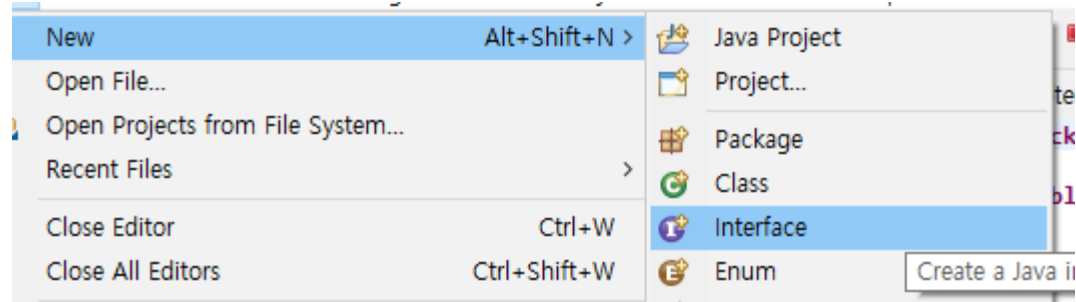
```
[public static final] 타입 상수이름 = 값;
```

- 상수 이름은 대문자로 작성하되 서로 다른 단어로 구성되어 있을 경우 언더바(_)로 연결

```
public interface RemoteControl {  
    public int MAX_VOLUME = 10;  
    public int MIN_VOLUME = 0;  
}
```



인터페이스 선언



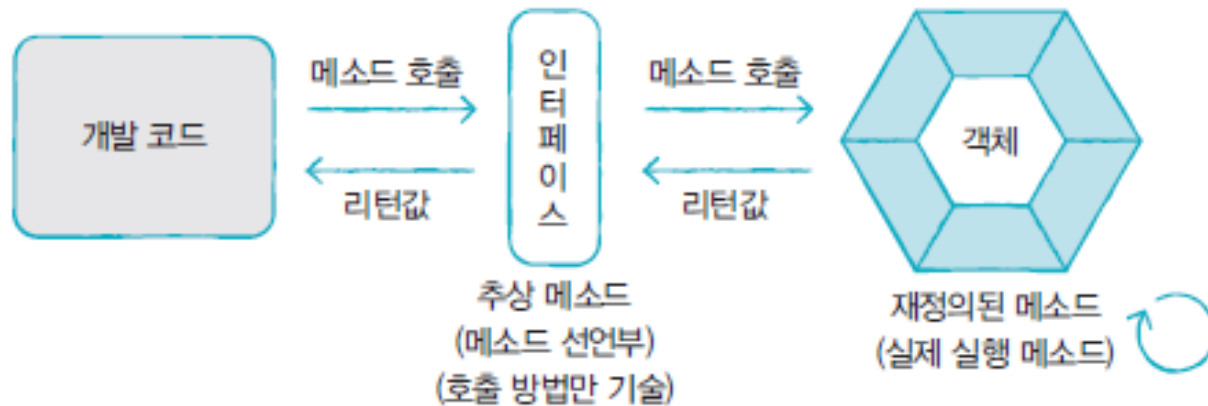
```
1 package sec01.exam02;  
2  
3 public interface RemoteControl {  
4     int MAX_VOLUME = 10;  
5     int MIN_VOLUME = 0;  
6 }  
7
```



인터페이스 선언

❖ 추상 메소드 선언

- 인터페이스 통해 호출된 메소드는 최종적으로 객체에서 실행
- 인터페이스의 메소드는 실행 블록 필요 없는 추상 메소드로 선언



[public abstract] 리턴타입 메소드이름(매개변수, ...);

```
public interface RemoteControl {  
    //추상 메소드  
    public void turnOn();  
    public void turnOff();  
    public void setVolume(int volume);  
}
```

인터페이스 선언

```
1 package sec01.exam03;
2
3 public interface RemoteControl {
4     //상수
5     int MAX_VOLUME = 10;
6     int MIN_VOLUME = 0;
7
8     //추상 메소드
9     void turnOn();
10    void turnOff();
11    void setVolume(int volume);
12 }
13
```



인터페이스 구현

❖ 구현 (implement) 클래스

- 인터페이스에서 정의된 추상 메소드를 재정의해서 실행내용을 가지고 있는 클래스
- 클래스 선언부에 implements 키워드 추가하고 인터페이스 이름 명시

```
public class 구현클래스이름 implements 인터페이스이름 {  
    //인터페이스에 선언된 추상 메소드의 실제 메소드 선언  
}
```

```
public class Television implements RemoteControl {  
    //turnOn() 추상 메소드의 실제 메소드  
    public void turnOn() {  
        System.out.println("TV를 켭니다.");  
    }  
    //turnOff() 추상 메소드의 실제 메소드  
    public void turnOff() {  
        System.out.println("TV를 끕니다.");  
    }  
}
```

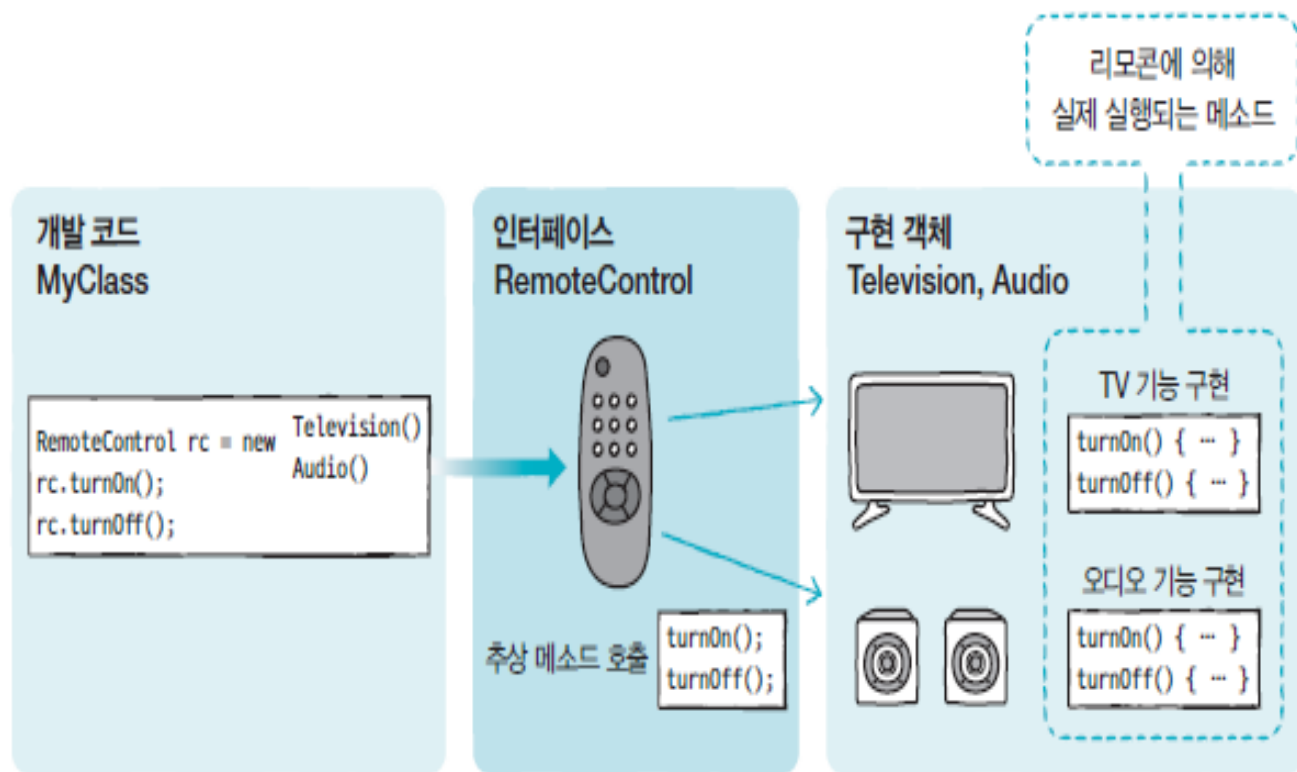


인터페이스 구현

- ❖ 인터페이스와 구현 클래스 사용 방법
 - 인터페이스 변수 선언하고 구현 객체를 대입

```
인터페이스 변수;  
변수 = 구현객체;
```

```
인터페이스 변수 = 구현객체;
```



인터페이스 구현

- sec01.exam04
 - Audio.java
 - RemoteControl.java
 - RemoteControlExample.java
 - Television.java

```
1 package sec01.exam04;
2
3 public class Television implements RemoteControl {
4     //필드
5     private int volume;
6
7     //turnOn() 추상 메소드의 실제 메소드
8     public void turnOn() {
9         System.out.println("TV를 켭니다.");
10    }
11    //turnOff() 추상 메소드의 실제 메소드
12    public void turnOff() {
13        System.out.println("TV를 끕니다.");
14    }
15    //setVolume() 추상 메소드의 실제 메소드
16    public void setVolume(int volume) {
17        if(volume>RemoteControl.MAX_VOLUME) {
18            this.volume = RemoteControl.MAX_VOLUME;
19        } else if(volume<RemoteControl.MIN_VOLUME) {
20            this.volume = RemoteControl.MIN_VOLUME;
21        } else {
22            this.volume = volume;
23        }
24        System.out.println("현재 TV 볼륨: " + this.volume);
25    }
26 }
27
```

```
1 package sec01.exam04;
2
3 public class Audio implements RemoteControl {
4     //필드
5     private int volume;
6
7     //turnOn() 추상 메소드의 실제 메소드
8     public void turnOn() {
9         System.out.println("Audio를 켭니다.");
10    }
11    //turnOff() 추상 메소드의 실제 메소드
12    public void turnOff() {
13        System.out.println("Audio를 끕니다.");
14    }
15    //setVolume() 추상 메소드의 실제 메소드
16    public void setVolume(int volume) {
17        if(volume>RemoteControl.MAX_VOLUME) {
18            this.volume = RemoteControl.MAX_VOLUME;
19        } else if(volume<RemoteControl.MIN_VOLUME) {
20            this.volume = RemoteControl.MIN_VOLUME;
21        } else {
22            this.volume = volume;
23        }
24        System.out.println("현재 Audio 볼륨: " + this.volume);
25    }
26 }
27
```

인터페이스 구현

```
1 package sec01.exam04;
2
3 public class RemoteControlExample {
4     public static void main(String[] args) {
5         RemoteControl rc;
6         rc = new Television();
7         rc = new Audio();
8     }
9 }
10
```

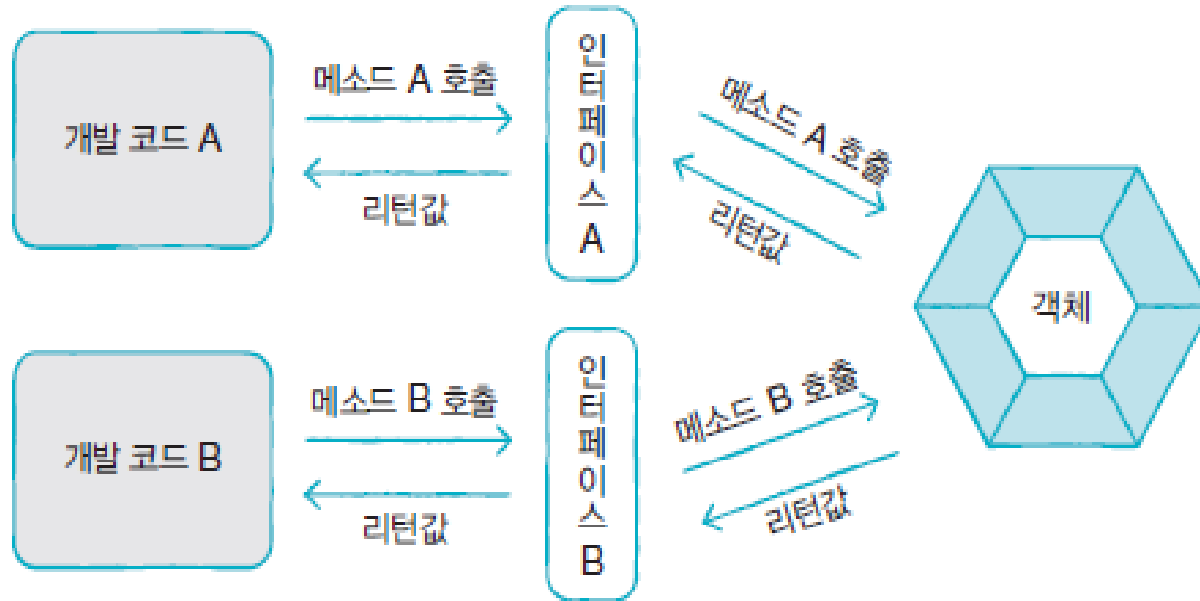
```
1 package sec01.exam04;
2
3 public interface RemoteControl {
4     //상수
5     int MAX_VOLUME = 10;
6     int MIN_VOLUME = 0;
7
8     //추상 메소드
9     void turnOn();
10    void turnOff();
11    void setVolume(int volume);
12 }
13
```

- ▼ sec01.exam04
 - > Audio.java
 - > RemoteControl.java
 - > RemoteControlExample.java
 - > Television.java

인터페이스 구현

❖ 다중 인터페이스 구현 클래스

- 객체는 다수의 인터페이스 타입으로 사용 가능



```
public class 구현클래스이름 implements 인터페이스A, 인터페이스B {  
    //인터페이스 A에 선언된 추상 메소드의 실제 메소드 선언  
    //인터페이스 B에 선언된 추상 메소드의 실제 메소드 선언  
}
```


인터페이스 구현

```
1 package sec01.exam05;
2
3 public class SmartTelevision implements RemoteControl, Searchable {
4     private int volume;
5
6     public void turnOn() {
7         System.out.println("TV를 켭니다.");
8     }
9     public void turnOff() {
10        System.out.println("TV를 끕니다.");
11    }
12    public void setVolume(int volume) {
13        if(volume > RemoteControl.MAX_VOLUME) {
14            this.volume = RemoteControl.MAX_VOLUME;
15        } else if(volume < RemoteControl.MIN_VOLUME) {
16            this.volume = RemoteControl.MIN_VOLUME;
17        } else {
18            this.volume = volume;
19        }
20        System.out.println("현재 TV 볼륨: " + this.volume);
21    }
22
23    public void search(String url) {
24        System.out.println(url + "을 검색합니다.");
25    }
26 }
```

```
1 package sec01.exam05;
2
3 public interface RemoteControl {
4     //상수
5     int MAX_VOLUME = 10;
6     int MIN_VOLUME = 0;
7
8     //추상 메소드
9     void turnOn();
10    void turnOff();
11    void setVolume(int volume);
12 }
```

```
1 package sec01.exam05;
2
3 public interface Searchable {
4     void search(String url);
5 }
6
```

- ▼ sec01.exam05
 - > RemoteControl.java
 - > Searchable.java
 - > SmartTelevision.java
 - > SmartTelevisionExample.java

인터페이스 구현

```
1 package sec01.exam05;  
2  
3 public class SmartTelevisionExample {  
4     public static void main(String[] args) {  
5         SmartTelevision tv = new SmartTelevision();  
6  
7         RemoteControl rc = tv;  
8         Searchable searchable = tv;  
9     }  
10 }  
11
```

- ▼ sec01.exam05
 - > RemoteControl.java
 - > Searchable.java
 - > SmartTelevision.java
 - > SmartTelevisionExample.java

인터페이스 사용

❖ 인터페이스 사용

- 인터페이스는 필드, 매개 변수, 로컬 변수의 타입으로 선언가능

```
public class MyClass {  
    //필드  
    ① RemoteControl rc = new Television();  
  
    //생성자  
    ② MyClass( RemoteControl rc ) {  
        this.rc = rc;  
    }  
  
    //메소드  
    void methodA() {  
        //로컬 변수  
        ③ RemoteControl rc = new Audio();  
    }  
  
    ④ void methodB( RemoteControl rc ) { ... }  
}
```

생성자의 매개값으로 구현 객체 대입
MyClass mc = new MyClass(new Television());

생성자의 매개값으로 구현 객체 대입
mc.methodB(new Audio());



인터페이스 사용

```
1 package sec01.exam06;
2
3 public class MyClass {
4     // 필드
5     RemoteControl rc = new Television();
6
7     // 생성자
8     MyClass() {
9     }
10
11     MyClass(RemoteControl rc) {
12         this.rc = rc;
13         rc.turnOn();
14         rc.setVolume(5);
15     }
16
17     // 메소드
18     void methodA() {
19         RemoteControl rc = new Audio();
20         rc.turnOn();
21         rc.setVolume(5);
22     }
23
24     void methodB(RemoteControl rc) {
25         rc.turnOn();
26         rc.setVolume(5);
27     }
28 }
29
```

```
1 package sec01.exam06;
2
3 public class MyClassExample {
4     public static void main(String[] args) {
5         System.out.println("1-----");
6
7         MyClass myClass1 = new MyClass();
8         myClass1.rc.turnOn();
9         myClass1.rc.setVolume(5);
10
11         System.out.println("2-----");
12
13         MyClass myClass2 = new MyClass(new Audio());
14
15         System.out.println("3-----");
16
17         MyClass myClass3 = new MyClass();
18         myClass3.methodA();
19
20         System.out.println("4-----");
21
22         MyClass myClass4 = new MyClass();
23         myClass4.methodB(new Television());
24     }
25 }
26
```

sec01.exam06

- > Audio.java
- > MyClass.java
- > MyClassExample.java
- > RemoteControl.java
- > Television.java

키워드로 끝내는 핵심 포인트

- **인터페이스**: 객체의 사용 방법 정의한 타입
- **상수 필드**: 인터페이스의 필드는 기본적으로 public static final 특성 가짐
- **추상 메소드**: 인터페이스의 메소드는 public abstract 생략되고 메소드 선언부만 있는 추상 메소드
- **implements**: 구현 클래스에는 어떤 인터페이스로 사용 가능한지 기술하기 위해 사용
- **인터페이스 사용**: 클래스 선언 시 필드, 매개 변수, 로컬 변수로 선언 가능. 구현 객체를 대입.



Chapter

08

인터페이스



08-2. 타입 변환과 다형성

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 자동 타입 변환
- 필드의 다형성
- 매개 변수의 다형성
- 강제 타입 변환
- 객체 타입 확인
- 인터페이스 상속
- 키워드로 끝내는 핵심 포인트

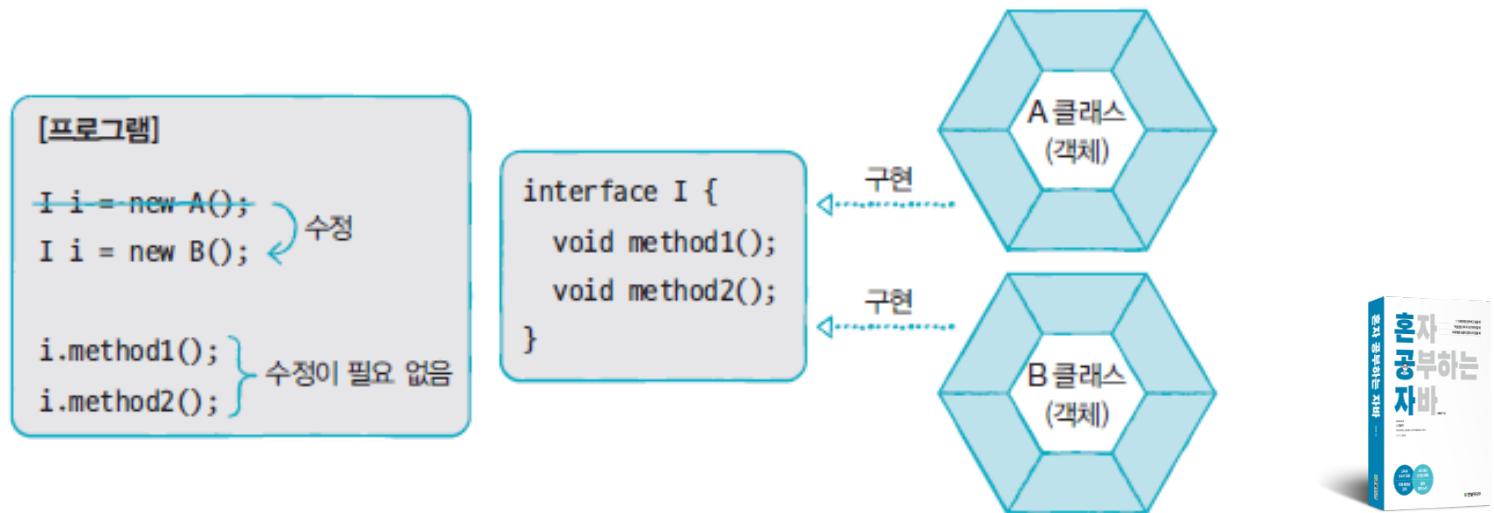


시작하기 전에

인터페이스도 메소드 재정의와 타입 변환되므로 다형성을 구현할 수 있다.

❖ 인터페이스의 다형성

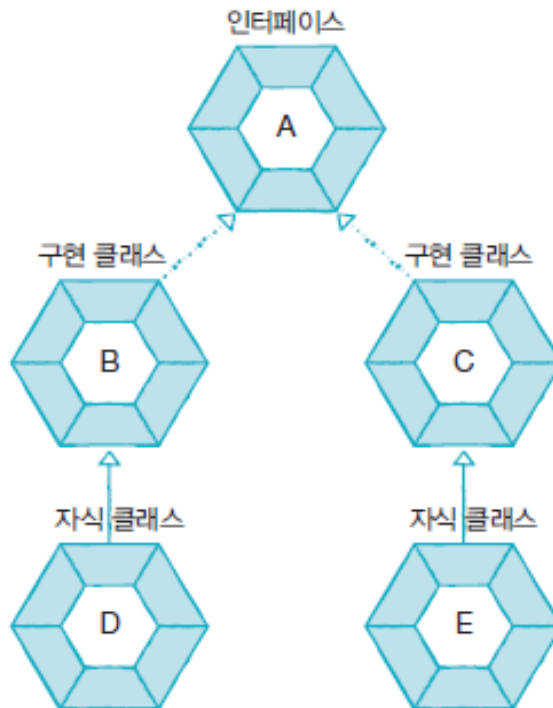
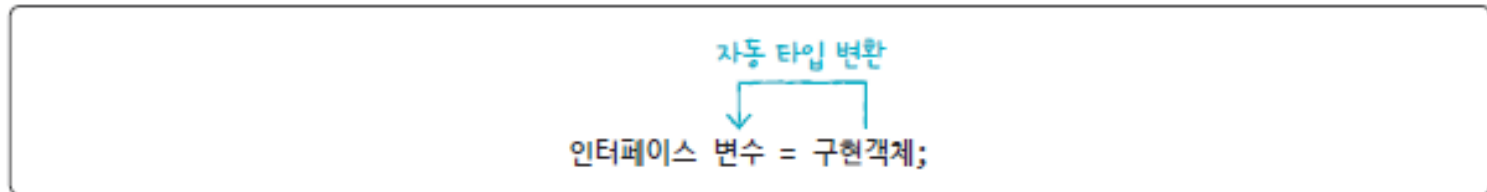
- 인터페이스 사용 방법은 동일하지만 **구현 객체 교체하여** 프로그램 실행 결과를 다양화



자동 타입 변환

❖ 자동 타입 변환 (promotion)

- 구현 객체와 자식 객체는 인터페이스 타입으로 자동 타입 변환 된다.



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

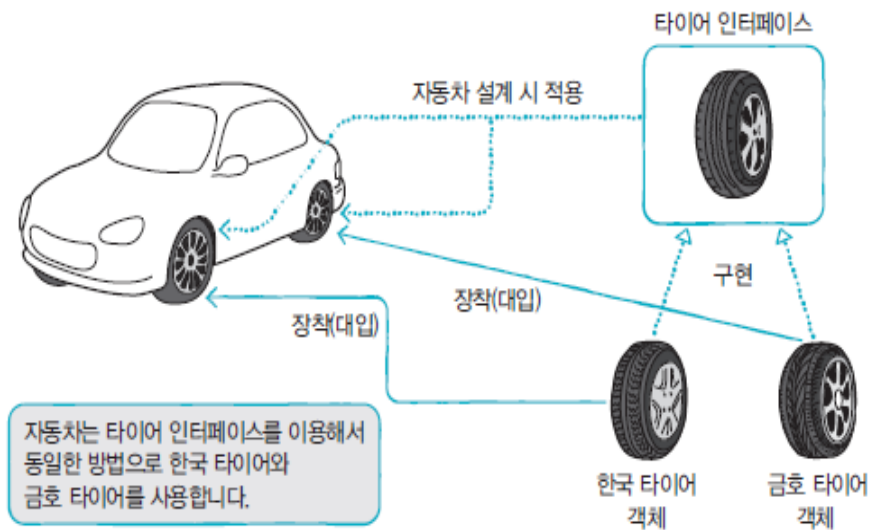
```
A a1 = b; //(가능)  
A a2 = c; //(가능)  
A a3 = d; //(가능)  
A a4 = e; //(가능)
```



❖ 필드의 다형성

```
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();
```



필드의 다형성

```
1 package sec02.exam01;
2
3 public class Car {
4     Tire frontLeftTire = new HankookTire();
5     Tire frontRightTire = new HankookTire();
6     Tire backLeftTire = new HankookTire();
7     Tire backRightTire = new HankookTire();
8
9     void run() {
10         frontLeftTire.roll();
11         frontRightTire.roll();
12         backLeftTire.roll();
13         backRightTire.roll();
14     }
15 }
```

```
1 package sec02.exam01;
2
3 public class HankookTire implements Tire {
4     @Override
5     public void roll() {
6         System.out.println("한국 타이어가 굴러갑니다.");
7     }
8 }
9
```

```
1 package sec02.exam01;
2
3 public interface Tire {
4     public void roll();
5 }
6
```

```
1 package sec02.exam01;
2
3 public class CarExample {
4     public static void main(String[] args) {
5         Car myCar = new Car();
6
7         myCar.run();
8
9         myCar.frontLeftTire = new KumhoTire();
10        myCar.frontRightTire = new KumhoTire();
11
12        myCar.run();
13    }
14 }
15
```

```
1 package sec02.exam01;
2
3 public class KumhoTire implements Tire {
4     @Override
5     public void roll() {
6         System.out.println("금호 타이어가 굴러갑니다.");
7     }
8 }
9
```

sec02.exam01

- > Car.java
- > CarExample.java
- > HankookTire.java
- > KumhoTire.java
- > Tire.java

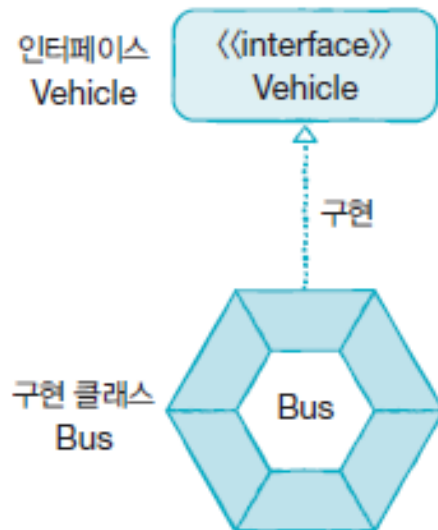
매개 변수의 다형성

❖ 매개변수의 다형성

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

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

구현 객체
구현 객체의 run() 메소드가 실행됨



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

자동 타입 변환 발생
Vehicle vehicle = bus;



매개 변수의 다형성

```
1 package sec02.exam02;
2
3 public class Driver {
4     public void drive(Vehicle vehicle) {
5         vehicle.run();
6     }
7 }
8
```

```
1 package sec02.exam02;
2
3 public class DriverExample {
4     public static void main(String[] args) {
5         Driver driver = new Driver();
6
7         Bus bus = new Bus();
8         Taxi taxi = new Taxi();
9
10        driver.drive(bus);
11        driver.drive(taxi);
12    }
13 }
14
```

```
1 package sec02.exam02;
2
3 public interface Vehicle {
4     public void run();
5 }
6
```

```
1 package sec02.exam02;
2
3 public class Bus implements Vehicle {
4     @Override
5     public void run() {
6         System.out.println("버스가 달립니다.");
7     }
8 }
9
```

```
1 package sec02.exam02;
2
3 public class Taxi implements Vehicle {
4     @Override
5     public void run() {
6         System.out.println("택시가 달립니다.");
7     }
8 }
9
```

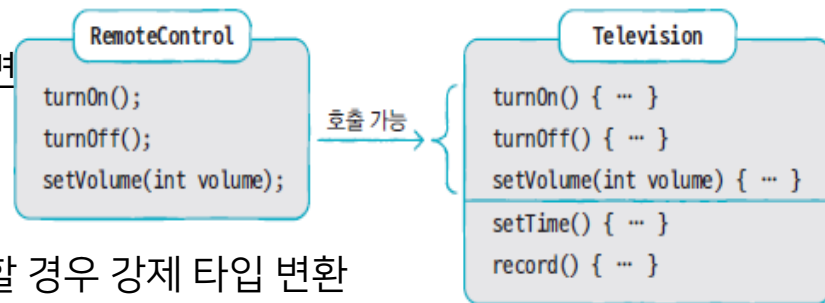
▼ sec02.exam02

- > Bus.java
- > Driver.java
- > DriverExample.java
- > Taxi.java
- > Vehicle.java

강제 타입 변환

❖ 강제 타입 변환 (casting)

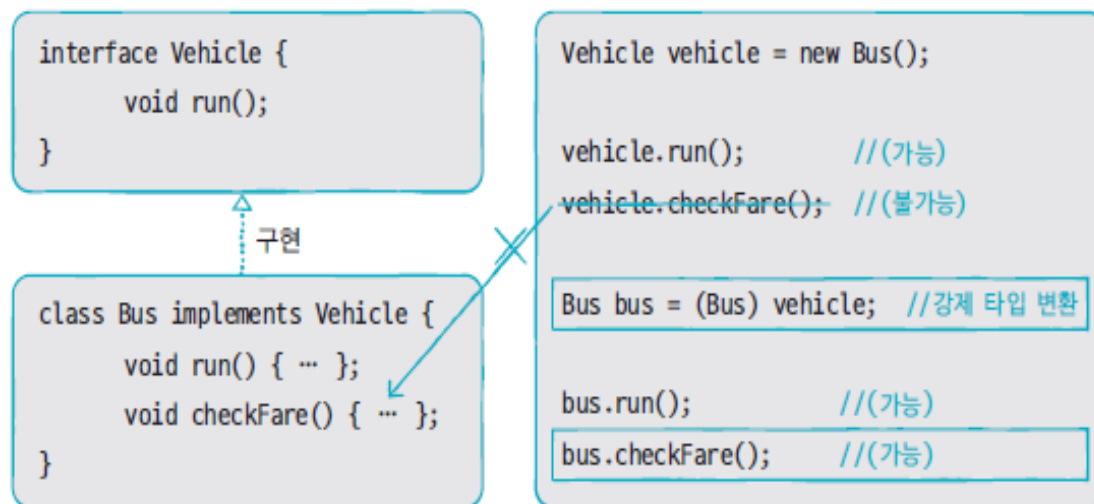
- 구현 객체가 인터페이스 타입으로 자동 변환하면
인터페이스에 선언된 메소드만 사용 가능
- 구현 클래스에만 선언된 필드나 메소드를 사용할 경우 강제 타입 변환



강제타입 변환

↓

구현클래스 변수 = (구현클래스) 인터페이스변수;



강제 타입 변환

```
1 package sec02.exam03;
2
3 public class VehicleExample {
4     public static void main(String[] args) {
5         Vehicle vehicle = new Bus();
6
7         vehicle.run();
8         //vehicle.checkFare(); (x)
9
10        Bus bus = (Bus) vehicle; //강제타입변환
11
12        bus.run();
13        bus.checkFare();
14    }
15 }
16
```

```
1 package sec02.exam03;
2
3 public interface Vehicle {
4     public void run();
5 }
6
```

```
1 package sec02.exam03;
2
3 public class Bus implements Vehicle {
4     @Override
5     public void run() {
6         System.out.println("버스가 달립니다.");
7     }
8
9     public void checkFare() {
10        System.out.println("승차요금을 체크합니다.");
11    }
12 }
13
```

sec02.exam03

- Bus.java
- Vehicle.java
- VehicleExample.java

객체 타입 확인

❖ 객체 타입 확인 instanceof

- 구현 객체가 변환되어 있는지 알 수 없는 상태에서 강제 타입 변환할 경우 ClassCastException 발생

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

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

- instanceof 연산자로 확인 후 안전하게 강제 타입 변환

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

Bus 객체 Taxi 객체
 ↓ ↓
if(vehicle instanceof Bus) { ← vehicle 매개 변수가 참조하는 객체가 Bus인지 조사
 Bus bus = (Bus) vehicle; ← Bus 객체일 경우 안전하게 강제타입 변환
 bus.checkFare(); ← Bus 타입으로 강제 타입 변환을 하는 이유
}



객체 타입 확인

```
1 package sec02.exam04;
2
3 public class Driver {
4     public void drive(Vehicle vehicle) {
5         if(vehicle instanceof Bus) {
6             Bus bus = (Bus) vehicle;
7             bus.checkFare();
8         }
9         vehicle.run();
10    }
11 }
12
```

```
1 package sec02.exam04;
2
3 public class DriverExample {
4     public static void main(String[] args) {
5         Driver driver = new Driver();
6
7         Bus bus = new Bus();
8         Taxi taxi = new Taxi();
9
10        driver.drive(bus);
11        driver.drive(taxi);
12    }
13 }
14
```

sec02.exam04

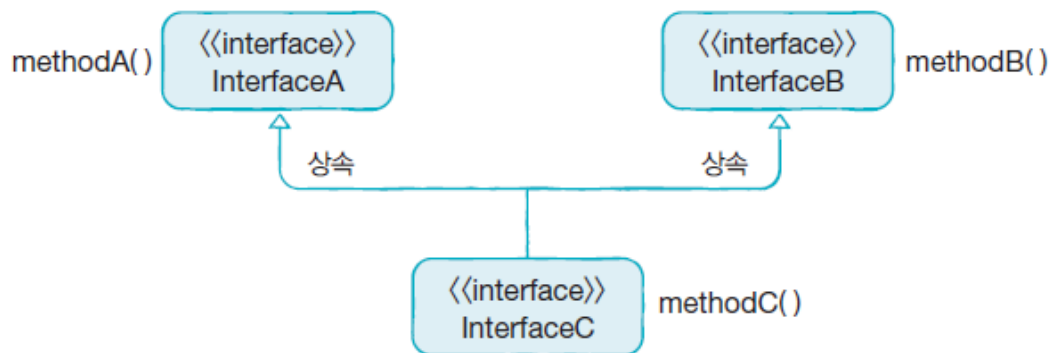
- > Bus.java
- > Driver.java
- > DriverExample.java
- > Taxi.java
- > Vehicle.java

인터페이스 상속

❖ 인터페이스 상속

- 인터페이스는 다중 상속을 할 수 있다.

```
public interface 하위인터페이스 extends 상위인터페이스1, 상위인터페이스2 { ... }
```



```
public interface InterfaceC extends InterfaceA, InterfaceB {
```

```
하위인터페이스 변수 = new 구현클래스(...);  
상위인터페이스1 변수 = new 구현클래스(...);  
상위인터페이스2 변수 = new 구현클래스(...);
```

```
public class ImplementationC implements InterfaceC {  
  
    ImplementationC impl = new ImplementationC();  
  
    InterfaceC ic = impl;  
    InterfaceA ia = impl;  
    InterfaceB ib = impl;
```



인터페이스 상속

```
1 package sec02.exam05;
2
3 public class Example {
4     public static void main(String[] args) {
5         ImplementationC impl = new ImplementationC();
6
7         InterfaceA ia = impl;
8         ia.methodA();
9         System.out.println();
10
11        InterfaceB ib = impl;
12        ib.methodB();
13        System.out.println();
14
15        InterfaceC ic = impl;
16        ic.methodA();
17        ic.methodB();
18        ic.methodC();
19    }
20 }
21
```

```
1 package sec02.exam05;
2
3 public interface InterfaceB {
4     public void methodB();
5 }
6
```

```
1 package sec02.exam05;
2
3 public interface InterfaceA {
4     public void methodA();
5 }
6
```

```
1 package sec02.exam05;
2
3 public interface InterfaceC extends InterfaceA, InterfaceB {
4     public void methodC();
5 }
6
7
```

```
1 package sec02.exam05;
2
3 public class ImplementationC implements InterfaceC {
4     public void methodA() {
5         System.out.println("ImplementationC-methodA() 실행");
6     }
7
8     public void methodB() {
9         System.out.println("ImplementationC-methodB() 실행");
10    }
11
12    public void methodC() {
13        System.out.println("ImplementationC-methodC() 실행");
14    }
15 }
16
```

sec02.exam05

- > Example.java
- > ImplementationC.java
- > InterfaceA.java
- > InterfaceB.java
- > InterfaceC.java

키워드로 끝내는 핵심 포인트

- **자동 타입 변환**: 구현 객체는 인터페이스 변수로 자동 타입 변환된다.
- **다형성**: 인터페이스도 재정의와 타입 변환 기능 제공하므로 다형성을 구현할 수 있다.
- **강제 타입 변환**: 인터페이스에 대입된 구현 객체를 다시 원래 타입으로 변환하는 것을 말한다.
- **instanceof**: 객체가 어떤 타입인지 조사할 때 사용한다. 강제 타입 변환 전에 사용.
- **인터페이스 상속**: 인터페이스는 다중 상속 허용한다.

