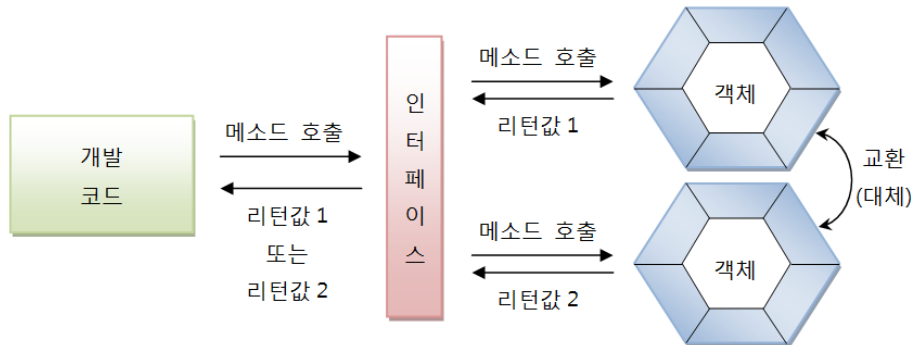


08장 인터페이스(interface)

8.1 인터페이스의 역할

- 객체의 교환성을 높여주기 때문에 **다형성을 구현하는 매우 중요한 역할**을 한다.



8.2 인터페이스 선언

8.2.1 인터페이스 선언

- 인터페이스는 **상수**와 **추상 메소드**로 구성되어 있다. 또한 **디폴트 메소드**와 **정적 메소드**는 자바 8에서 추가된 인터페이스의 새로운 멤버이다.

```
interface 인터페이스명 {
    타입 상수명 = 값; //상수
    타입 메소드명(매개변수,...); //추상 메소드
    default 타입 메소드명(매개변수,...) {...} // 디폴트 메소드
    static 타입 메소드명(매개변수) {...} // 정적 메소드
}

public interface RemoteControl {           //인터페이스
    int MAX_VOLUME = 10;                   //상수 (public static final생략가능)
    void turnOn();                          //추상 메소드 (public abstract 생략가능)
    default void setMute() {...}            //디폴트 메소드 (public 생략가능)
    static void changeBattery() {...}       //정적 메소드 (public 생략가능)
}
```

8.2.2 상수 필드 선언

- 인터페이스는 상수 필드만 선언 가능
 - 데이터 저장하지 않음
- 인터페이스에 선언된 필드는 모두 `public static final`
 - 생략하더라도 자동적으로 컴파일 과정에서 붙음
- 상수명은 **대문자로 작성**
 - 서로 다른 단어로 구성되어 있을 경우에는 **언더 바(_)로 연결**
- 선언과 동시에 초기값 지정

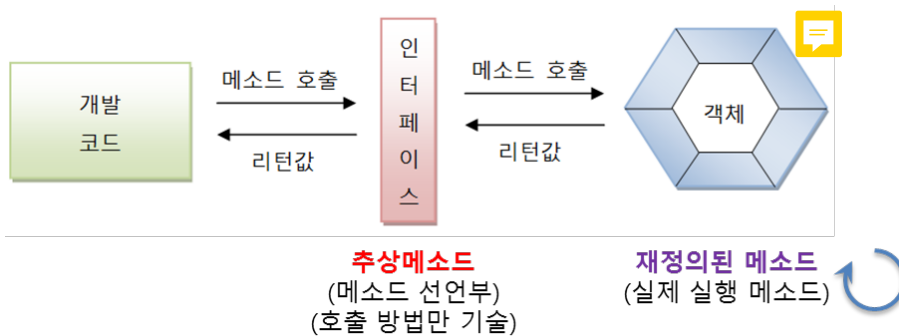
[RemoteControl.java] 인터페이스

```
package sec02.exam02_constant_field;

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

8.2.3 추상 메소드 선언

- 인터페이스 통해 호출된 메소드는 최종적으로 객체에서 실행
 - 인터페이스의 메소드는 기본적으로 실행 블록이 없는 추상 메소드로 선언
 - public abstract를 생략하더라도 자동적으로 컴파일 과정에서 붙게 됨



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

[RemoteControl.java] 인터페이스

```
package sec02.exam03_abstract_method;

public interface RemoteControl {
    // 상수
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    // 추상 메소드
    void turnOn();

    void turnOff();

    void setVolume(int volume);
}
```

8.2.4 디폴트 메소드 선언

- 자바8에서 추가된 인터페이스의 새로운 멤버
- 실행 블록을 가지고 있는 메소드
- default 키워드를 반드시 붙여야
- 기본적으로 public 접근 제한
 - 생략하더라도 컴파일 과정에서 자동 붙음

[RemoteControl.java] 인터페이스

```
package sec02.exam04_default_method;

public interface RemoteControl {
    // 상수
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    // 추상 메소드
    void turnOn();

    void turnOff();

    void setVolume(int volume);

    // 디폴트 메소드
    default void setMute(boolean mute) {
        if (mute) {
            System.out.println("무음 처리합니다.");
        } else {
            System.out.println("무음 해제합니다.");
        }
    }
}
```

8.2.5 정적 메소드 선언

- 자바8에서 추가된 인터페이스의 새로운 멤버
- 기본형: [public] static 리턴타입 메소드명(매개변수, ...) {...}

[RemoteControl.java] 인터페이스

```
package sec02.exam05_static_method;

public interface RemoteControl {
    // 상수
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    // 추상 메소드
    void turnOn();

    void turnOff();

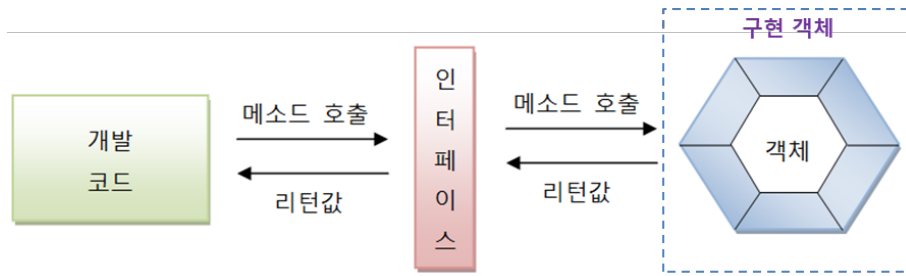
    void setVolume(int volume);

    // 디폴트 메소드
    default void setMute(boolean mute) {
        if (mute) {
            System.out.println("무음 처리합니다.");
        } else {
            System.out.println("무음 해제합니다.");
        }
    }

    // 정적 메소드
    static void changeBattery() {
        System.out.println("건전지를 교환합니다.");
    }
}
```

8.3 인터페이스 구현

- 인터페이스의 추상 메소드 대한 실제 메소드를 가진 객체 = 구현 객체
- 구현 객체를 생성하는 클래스 = 구현 클래스



8.3.1 구현 클래스

- 인터페이스는 자체적으로 객체를 생성할 수 없어 구현 클래스에 상속하여 사용한다.
- 인터페이스를 상속 받을때는 **implements**로 상속을 받는다.
- 인터페이스를 상속받은 일반 클래스는 인터페이스 안에 들어있는 추상 메소드를 반드시 Method Overriding해야 된다.

```

interface A{
    public abstract void check();
}
class S implements A{
    public void check(){ // method overriding시에 public을 생략할 수 없다.
    }
}
    
```

[Television.java] 구현 클래스

```

package cp08_interface.se03_implement.ex01_name;

public class Television implements RemoteControl {
    // 필드
    private int volume;

    // 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.");
    }

    // setVolume() 추상 메소드의 실제 메소드
    public void setVolume(int volume) {
        if (volume > RemoteControl.MAX_VOLUME) {
            this.volume = RemoteControl.MAX_VOLUME;
        } else if (volume < RemoteControl.MIN_VOLUME) {
            this.volume = RemoteControl.MIN_VOLUME;
        }
    }
}
    
```

```

    } else {
        this.volume = volume;
    }
    System.out.println("현재 TV 볼륨: " + volume);
}
}

```

[RemoteControlExample.java]

```

package sec03.exam01_name_implement_class;

public class RemoteControlExample {
    public static void main(String[] args) {
        RemoteControl rc;
        rc = new Television();
        rc = new Audio();
    }
}

```

8.3.2 익명 구현 객체



- 소스 파일을 만들지 않고도 구현 객체를 만들 수 있는 방법이다. 작성 시 주의할 점은 하나의 실행문이므로 끝에는 세미콜론(;)을 반드시 붙여야 한다.

```

인터페이스 변수 = new 인터페이스() {
    //인터페이스에 선언된 추상 메소드의 실제 메소드 선언
};

```

[RemoteControlExample.java] 익명 구현 클래스

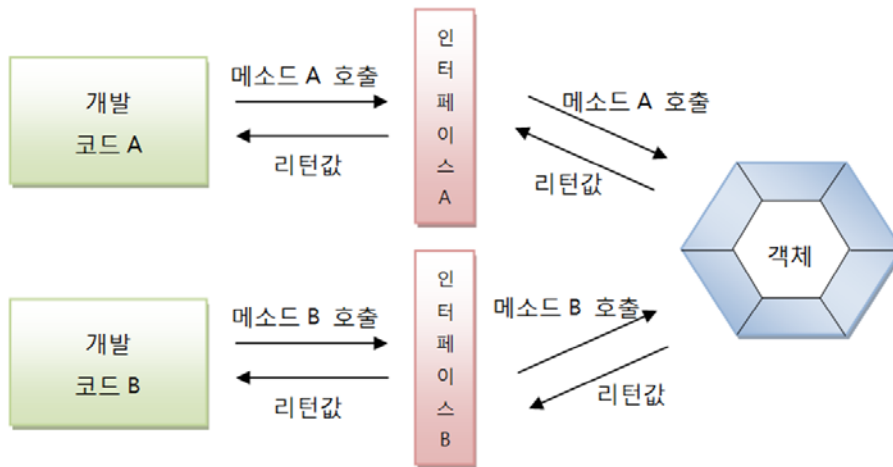
```

package sec03.exam02_noname_implement_class;

public class RemoteControlExample {
    public static void main(String[] args) {
        RemoteControl rc = new RemoteControl() {
            public void turnOn() { /*실행문*/ }
            public void turnOff() { /*실행문*/ }
            public void setVolume(int volume) { /*실행문*/ }
        };
    }
}

```

8.3.3 다중 인터페이스 구현 클래스



8.4 인터페이스 사용

- 개발 코드에서 인터페이스는 클래스의 필드, 생성자 또는 메소드의 매개 변수, 생성자 또는 메소드의 로컬 변수로 선언될 수 있다.

```
// 기본형
인터페이스 변수;
변수 = 구현객체;
인터페이스 변수 = 구현객체;

// 인터페이스 사용의 예
public class MyClass {
    RemoteControl rc = new Television(); // 필드
    MyClass(RemoteControl rc) { // 생성자, MyClass mc = new MyClass(new Television());
        this.rc = rc;
    }
    void methodA() { // 메소드
        RemoteControl rc = new Audio(); // 로컬 변수
    }
    void methodB(RemoteControl rc) {...} // 매개 변수, mc.methodB(new Audio());
}
```

8.4.1 추상 메소드 사용

```
RemoteControl rc = new Television();
rc.turnOn();    → Television 의 turnOn() 실행
rc.turnOff();   → Television 의 turnOff() 실행
```



8.4.2 디폴트 메소드 사용

- 인터페이스만으로는 사용 불가
 - 구현 객체가 인터페이스에 대입되어야 호출할 수 있는 인스턴스 메소드
- 모든 구현 객체가 가지고 있는 기본 메소드로 사용
 - 필요에 따라 구현 클래스가 디폴트 메소드 재정의해 사용

8.4.3 정적 메소드 사용

- 인터페이스로 바로 호출 가능

[RemoteControlExample.java]

```
package sec04.exam03_static_method_use;

public class RemoteControlExample {
    public static void main(String[] args) {
        RemoteControl.changeBattery();
    }
}
```

[RemoteControl.java]

```
package sec04.exam03_static_method_use;

public interface RemoteControl {
    //상수
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    //추상 메소드
    void turnOn();
    void turnOff();
    void setVolume(int volume);

    //디폴트 메소드
    default void setMute(boolean mute) {
        if(mute) {
            System.out.println("무음 처리합니다.");
        } else {
            System.out.println("무음 해제합니다.");
        }
    }
}
```

```

    }

    //정적 메소드
    static void changeBattery() {
        System.out.println("건전지를 교환합니다.");
    }
}

```

[꿀팁] 추상 클래스와 인터페이스를 모두 상속을 받는 경우

- 추상 클래스를 먼저 상속을 받고, 인터페이스는 그 다음으로 상속 받아야 한다. (상속 받는 순서가 바뀌면 안된다.)

```

abstract class A{}
interface B{}

class S extends A implements B{}

```

[InterfaceTest04.java] 추상 클래스와 인터페이스를 동시에 상속

```

package sec04.exam01_abstract_method_use;

interface IHello4 { // 인터페이스
    void sayHello(String name); // public abstract를 생략할 수 있다.
}

abstract class GoodBye4 { // 추상 클래스
    public abstract void sayGoodBye4(String name); // public abstract를 생략할 수 없다.
}

class SubClass4 extends GoodBye4 implements IHello4 {
    public void sayHello(String name) {
        System.out.println(name + "씨 안녕하세요!");
    }

    public void sayGoodBye4(String name) {
        System.out.println(name + "씨 안녕히 가세요!");
    }
}

public class InterfaceTest04 {
    public static void main(String[] args) {
        SubClass4 test = new SubClass4();
        test.sayHello("홍길동");
        test.sayGoodBye4("이순신");
    }
}

```

8.5 타입 변환과 다형성

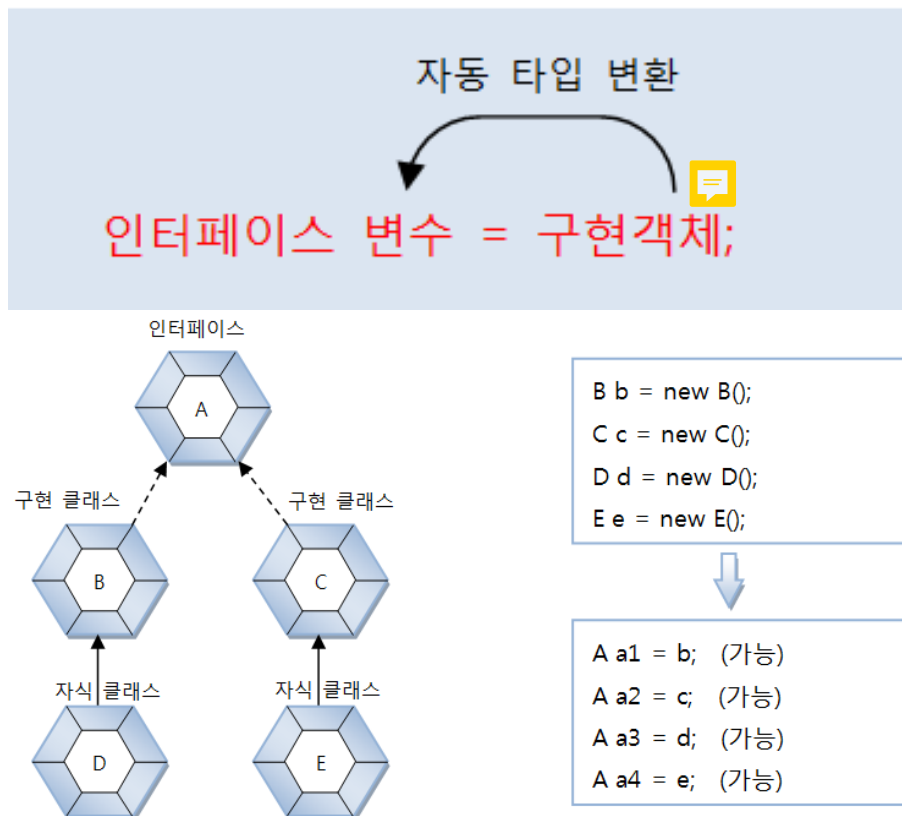
- 다형성: 하나의 타입에 여러 가지 객체 대입해 다양한 실행 결과를 얻는 것
- 다형성을 구현하는 기술
 - 상속 또는 인터페이스의 자동 타입 변환(Promotion)
 - 오버라이딩(Overriding)

■ 다형성의 효과

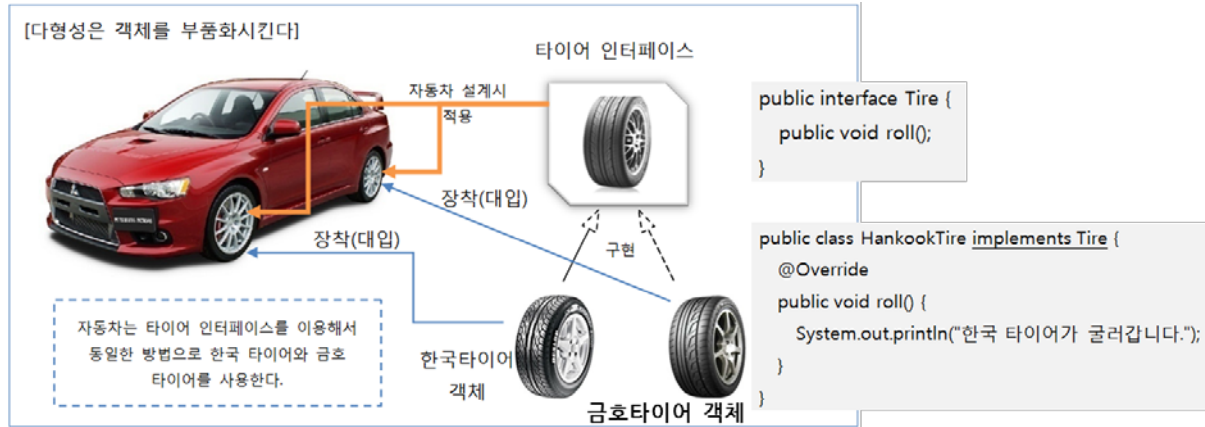
- 다양한 실행 결과를 얻을 수 있음
- 객체를 부품화시킬 수 있어 유지보수 용이 (메소드의 매개변수로 사용)

8.5.1 자동 타입 변환(Promotion)

- 기본형: 인터페이스 변수 = 구현객체; //자동 타입 변환
- 구현 객체가 인터페이스 타입으로 변환되는 것이다. -> 인터페이스에 선언된 메소드만 사용 가능하다. (참조 영역 축소)



8.5.2 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();
```

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

8.5.3 인터페이스 배열로 구현 객체 관리

- 인터페이스 배열로 관리할 수도 있다.

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

tires[1] = new KumhoTire();

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

8.5.4 매개 변수의 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();
}

Driver driver = new Driver();
Bus bus = new Bus();
driver.drive( bus ); // 자동타입변환 발생, Vehicle vehicle = bus;
```

8.5.5 강제 타입 변환(Casting)

- 기본형: 구현클래스 변수 = (구현클래스) 인터페이스변수; //강제 타입 변환
- 축소된 참조 영역을 원래대로 복원한다. -> 구현클래스의 필드와 메소드를 사용할 수 있다.

8.5.6 객체 타입 확인(instanceof)

- 강제 타입 변환 전 구현 클래스 타입 조사 -> 구현 클래스 타입이 달라 발생할 수 있는 ClassCastException을 방지할 수 있다.

```
[Driver.java]

package sec05.exam05_instanceof;

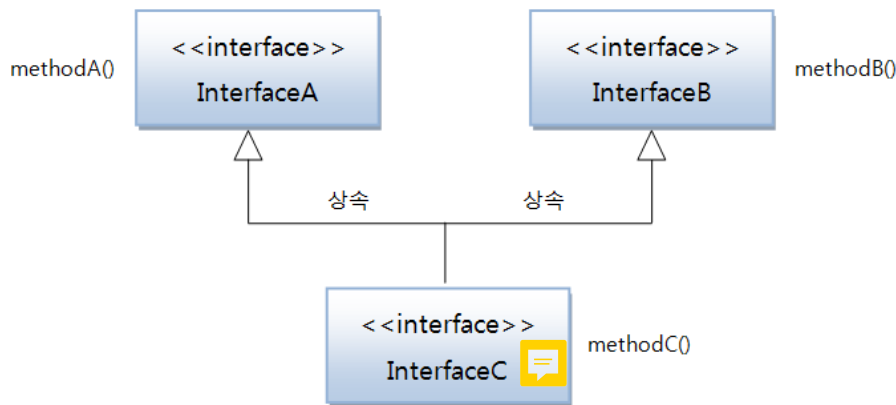
public class Driver {
    public void drive(Vehicle vehicle) {
        if(vehicle instanceof Bus) {
            Bus bus = (Bus) vehicle;
            bus.checkFare();
        }
        vehicle.run();
    }
}
```

8.6 인터페이스 상속

- 인터페이스는 다중상속을 허용한다.
- 하위 인터페이스를 구현하는 클래스는 하위 인터페이스의 메소드뿐만 아니라 상위 인터페이스의 모든 추상 메소드에 대한 실체 메소드를 가지고 있어야 한다.
- 인터페이스 자동 타입 변환 -> 해당 타입의 인터페이스에 선언된 메소드만 호출 가능하다.

```
public interface 하위인터페이스 extends 상위인터페이스1, 상위인터페이스2 {...}
public class 구현클래스 implements 인터페이스1, 인터페이스2 {...}

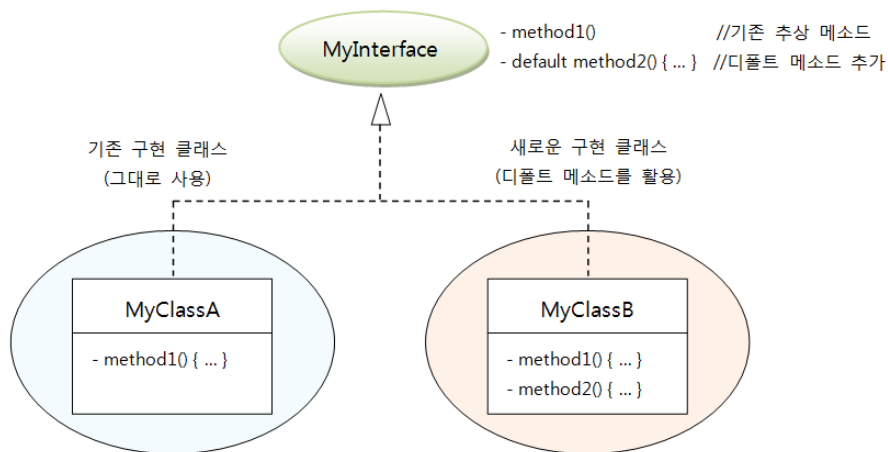
하위인터페이스 변수 = new 구현클래스(...);
상위인터페이스1 변수 = new 구현클래스(...);
상위인터페이스2 변수 = new 구현클래스(...);
```



8.7 디폴트 메소드와 인터페이스 확장

8.7.1 디폴트 메소드의 필요성

- 기존 인터페이스를 확장해서 새로운 기능을 추가하기 위해 사용한다. -> 새로운 추상메소드를 추가하지 않고 디폴트 메소드를 선언하여 사용할 수 있기 때문이다.



8.7.2 디폴트 메소드가 있는 인터페이스 상속

- 자식 인터페이스에서 디폴트 메소드를 활용하는 방법은 다음 세 가지가 있다.
 - 디폴트 메소드를 단순히 상속만 받는다.
 - 디폴트 메소드를 재정의(Override)해서 실행 내용을 변경한다.
 - 디폴트 메소드를 추상 메소드로 재선언한다.

[참고] 추상클래스와 인터페이스의 차이점

구분	추상클래스	인터페이스
1. 목적	상속을 통해 기능 이용/확장시키며, 특히 자식객체 간에 공통된 이름(필드, 메소드)으로 통일화시킨다.	구현 객체의 동일한 사용법을 보장하므로, 객체의 교환성을 높여 다형성을 구현한다.

2.구성멤버	필드, 생성자, 추상메소드, 일반메소드	상수, 추상메소드, 디폴트메소드, 정적메소드
3.객체 생성 /구현 방법	extends로 상속받아 자식 객체를 생성한다.	implements로 구현 객체를 생성한다.
4.Override/Implement 대상	추상메소드만 재정의(Override)하면 된다.	추상메소드는 반드시 재정의(Implement)되어야 하고, 디폴트메소드는 선택 사항이다.
5.다중상속	허용하지 않는다.	허용한다.

[과제] 확인문제

1. 인터페이스에 대한 설명으로 틀린 것은 무엇입니까?

- (1) 인터페이스는 객체 사용 설명서 역할을 한다.
- (2) 구현 클래스가 인터페이스의 추상 메소드에 대한 실제 메소드를 가지고 있지 않으면 추상 클래스가 된다.
- (3) 인터페이스는 인스턴스 필드를 가질 수 있다.
- (4) 구현 객체는 인터페이스 타입으로 자동 변환된다.

2. 인터페이스의 다형성과 거리가 먼 것은?

- (1) 필드가 인터페이스 타입일 경우 다양한 구현 객체를 대입할 수 있다.
- (2) 매개 변수가 인터페이스 타입일 경우 다양한 구현 객체를 대입할 수 있다.
- (3) 배열이 인터페이스 타입인 경우 다양한 구현 객체를 저장할 수 있다.
- (4) 구현 객체를 인터페이스 타입으로 변환하려면 강제 타입 변환을 해야 한다.

3. 다음은 Soundable 인터페이스입니다. sound() 추상 메소드는 객체의 소리를 리턴합니다.

```
public interface Soundable {
    String sound();
}
```

SoundableExample 클래스에서 printSound() 메소드는 Soundable 인터페이스 타입의 매개변수를 가지고 있습니다. main() 메소드에서 printSound()를 호출할 때 Cat과 Dog 객체를 주고 실행하면 각각 "야옹"과 "멍멍"이 출력되도록 Cat과 Dog 클래스를 작성해 보세요.

[SoundableExample.java]

```
01 package verify.exam03;
02
03 public class SoundableExample {
04     private static void printSound(Soundable soundable) {
05         System.out.println(soundable.sound());
06     }
07
08     public static void main(String[] args) {
09         printSound(new Cat());
10         printSound(new Dog());
11     }
12 }
```

```
13
14 // 실행 결과
15 // 아웅
16 // 멍멍
```

4. DaoExample 클래스는 main() 메소드에서 dbWork() 메소드를 호출할 때 OracleDao와 MySqlDao 객체를 매개값으로 주고 호출했습니다. dbWork() 메소드는 두 객체를 모두 매개값으로 받기 위해 DataAccessObject 타입의 매개 변수를 가지고 있습니다. 실행 결과를 보고 DataAccessObject 인터페이스와 OracleDao, MySqlDao 구현 클래스를 각각 작성해 보세요.

```
[DaoExample.java]

01 package verify.exam04;
02
03 public class DaoExample {
04     public static void dbWork(DataAccessObject dao) {
05         dao.select();
06         dao.insert();
07         dao.update();
08         dao.delete();
09     }
10
11     public static void main(String[] args) {
12         dbWork(new OracleDao());
13         dbWork(new MySqlDao());
14     }
15 }
16
17 // 실행 결과
18 // Oracle DB에서 검색
19 // Oracle DB에 삽입
20 // Oracle DB를 수정
21 // Oracle DB에서 삭제
22 // MySql DB에서 검색
23 // MySql DB에 삽입
24 // MySql DB를 수정
25 // MySql DB에서 삭제
```

5. 다음은 Action 인터페이스입니다. work() 추상 메소드는 객체의 작업을 시작시킵니다.

```
public interface Action {
    void work();
}
```

ActionExample 클래스의 main() 메소드에서 Action의 익명 구현 객체를 만들어 다음과 같은 실행 결과가 나올 수 있도록 박스 안에 들어갈 코드를 작성해보세요.

```
[ActionExample.java]

01 package verify.exam05;
02
03 public class ActionExample {
04     public static void main(String[] args) {
05         // 작성 위치
06         Action action =
```

```
07
08
09
10
11
12         action.work();
13     }
14 }
15
16 // 실행 결과
17 // 복사를 합니다.
```