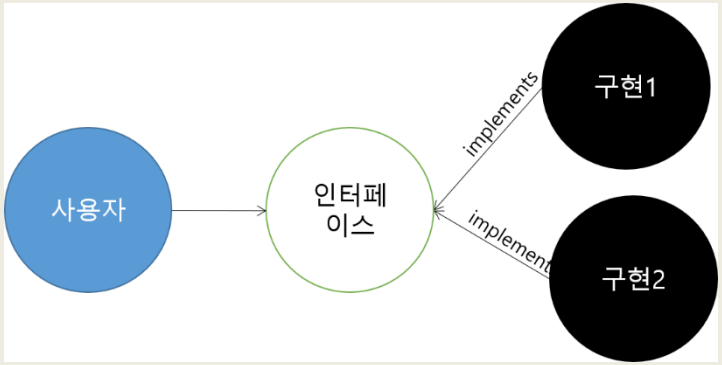
제 9 장
인터페이스

인터페이스(interface)

- 서로 다른 모듈(하드웨어, 소프트웨어) 간의 연결 및 소통을 위하여 상호 정보를 전달하는 (표준) 동작 규격들의 집합(A set of procedures (or functions))
- Example: Home System
- 자바 인터페이스는 **추상메소드**와 **상수**들로만 구성되는 클래스이다
  - 상수는 public static final로 간주
  - 다른 인터페이스로 상속이 될 수도 있고, 상속을 받을 수도 있다
  - 다중 상속 허용 → 필드 선언은 허용이 안된다
  - Java8.0 부터 디폴트 메소드와 정적 메소드 포함 가능
- 다른 클래스에서 상속받아서 구현(implements)하여 이용한다



인터페이스의 선언

```
public interface 인터페이스명 {
        // 상수 선언
        // 추상메소드 선언
        반환형    추상메소드1(...);
        반환형    추상메소드2(...);
        .....
}
```

- 접근지정자(modifier) +
- 키워드 “**interface**” +
- 인터페이스 이름 +
- 인터페이스 상속의 경우 키워드 “**extends**”와 함께 콤마(,)로 분리되는 부모인터페이스들의 리스트
- 상수와 메소드 시그니처, 디폴트 메소드 혹은 정적 메소드로 구성되는 인터페이스 몸체

```
public interface MyInterface extends Interface1, Interface2, Interface3 {
    // 상수 선언
    double E = 2.718282;
    // 자신만의 메소드 시그너쳐
    void doSomething (int I, double x);
    int doSomethingElse(String s);
}
```

- 인터페이스는 추상 메소드로 구성되므로 생성을 통한 인스턴스화를 할 수 없다
- 다른 클래스에 의해 구현(implement)되거나 다른 인터페이스로 상속되거나 상속 받을 수만 있다

```
public interface RemoteControl {
    // 추상 메소드 정의
    public void turnOn();      // 가전 제품을 켠다.
    public void turnOff();     // 가전 제품을 끈다.
    public void changeVolume(int vLevel);       //  볼륨을 조정
    public void changeChannel(int channel); // 채널을 변경
}
```

인터페이스의 구현(상속)

```
public class 클래스이름 implements 인터페이스이름 {
        반환형    추상메소드1(...) {
                  // 메소드 몸체
        }
        반환형    추상메소드2(...) {
                  //메소드 몸체
        }
        ....
}
```

예제

```
public class RemconSamsung implements RemoteControl {
// RemoteControl의 메소드 시그너처를 구현
    public void turnOn( ) {
        System.out.println("TV가 켜졌습니다₩n"); // 비어있는 메소드 몸체 구현
    }
    public void turnOff( ) {
        System.out.println("TV가 꺼졌습니다₩n"); // 비어있는 메소드 몸체 구현
    }
    public void changeVolume(int vLevel) {
        System.out.println("TV 볼륨을 vLevel로 변경합니다₩n");
    }
    public void changeChannel(int channel); {
        System.out.println("TV 채널을 channel로 변경합니다₩n");
    }
    // 필요하면 다른 디폴트 메소드 혹은 정적 메소드 구현
    . . .
}
```

```
RemoconSamsung r = new RemoconSamsung();
// RemoteControl obj = new RemoconSamsung(); 도 가능함
r.turnOn();
r.turnOff();
```

```
public  interface Calculator {
    int sum(int x, int y);
    int diff(int x, int y);
    int multi(int x, int y);
    float div(int x, int y);
}
```

```
public class MyCalculator implements Calculator {
    public int sum(int x, int y) {
        return x+y;
    }
    public int diff(int x, int y) {
        return x-y;
    }
    public int multi(int x, int y) {
        return x*y;
    }
    public float div(int x, int y) {
        return x/y;
    }
}
```

Simple 자율 주행 자동차

- 제작하고자 하는 자율주행 자동차에 대한 정적 속성과 동적 속성을 설계한다 → interface를 설계한다
  - 정적 특성(상수필드) : 최대속력(MAX_SPEED), 최대RPM(max_RPM), 최대기어변속범위(MAX_GEAR_CHANGE)
  - 동적 특성(추상메소드) : startEngine(), gearChange(int gear_val), setSpeed(int speed), trunHandle(int degree), stopEngine() 등

```
public interface AutonomousCar {
   final int MAX_SPEED = 200;
   final int MAX_RPM = 6000;
   final int MAX_GEAR_CHANGE 5;

  void startEngine();
  void stopEngine();
  void gearChange(int gear);
  void setSpeed(int speed);
  void turnHandle(int degree);
}
```

```
public class AutoCar implements AutonomousCar {
    public void startEngine() {
            System.out.println("자동차가 출발합니다.");
    }
    public void stopEngine() {
            System.out.println("자동차가 정지합니다.");
    }
    public void setSpeed(int speed) {
            System.out.println("자동차가 속도를 " + speed + "km/h로 바꿉니다.");
    }
    public void turnHandle(int degree) {
            System.out.println("자동차가 방향을 " + degree + "도 만큼 바꿉니다.");
    }
     public void gearChange(int gear) {
            System.out.println("자동차가 기어를 " + gear + "로 변경했습니다");
    }
}
```

```
public class AutoCarTest {
    public static void main(String[] args) {
            AutonomousCar obj = new AutoCar();
            obj.start();
            obj.setSpeed(30);
            obj.turnHandle(15);
            obj.stop();
    }
}
```

객체 비교하기 예

- ***Relatable*** 인터페이스
  - 2개의 객체의 크기 비교를 규격화하여 정의하는 인터페이스
  - 크기 비교의 결과는 3가지 경우로 분류하여 반환값 1(큰 경우), 0(같은 경우), -1(작은 경우)으로 알려주기로 한다
  - 다음과 같이 정의되는 추상메소드 islargerThan()을 포함한다

```
public interface Relatable {
// 객체의 크기 비교에 따라 1, 0, -1을 리턴하는 메소드 시그너쳐
    public int isLargerThan(Relatable other);
}
```

```
public class Rectangle implements Relatable {
    public int width = 0;
    public int height = 0;
    public Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    // 사각형 면적
    public int getArea() {
        return width * height;
    }
  // Relatable의 구현
  public int isLargerThan(Relatable other) {
    Rectangle otherRect = (Rectangle)other;
    if (this.getArea() < otherRect.getArea())
        return -1;
    else if (this.getArea() > otherRect.getArea())
        return 1;
    else
        return 0;
  }
}
```

```
public class RectangleTest {
  public static void main(String[] args) {
            Rectangle r1 = new Rectangle(100, 30);
            Rectangle r2 = new Rectangle(200, 10);
            int result = r1.isLatgerThan(r2);
            if (result == 1)
                        System.out.println(r1 + "가 더 큽니다.");
            else if (result == 0)
                        System.out.println("같습니다");
            else
                        System.out.println(r2 + "가 더 큽니다.");
  }
}
```

인터페이스 타입

- 인터페이스는 하나의 **객체 타입**으로 간주된다.
- 인터페이스 선언은 새로운 객체 타입을 선언하는 것과 동일 효과

```
RemoteControl obj = new RemoconSamung();
obj.turnOn();
obj.turnOff();
```

인터페이스로 참조 변수를 만들 수 있다.

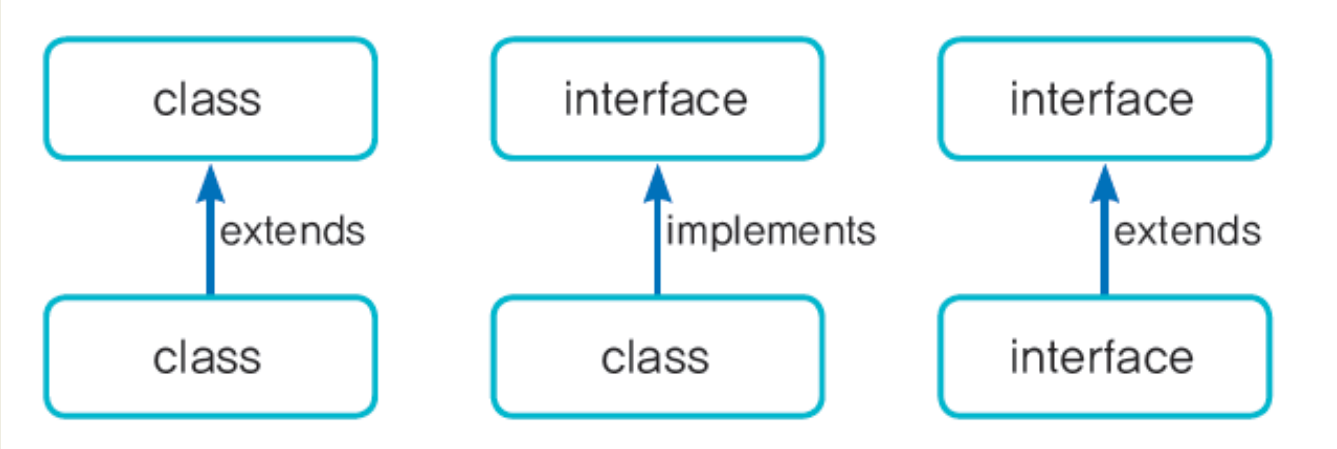
```
public interface Comparable {
    // 이 객체가 다른 객체보다 크면 1, 같으면 0, 작으면 -1을 반환한다.
    int compareTo(Object other);
}
```

```
public Object findLargest(Object object1, Object object2) {
   Comparable obj1 = (Comparable)object1;
   Comparable obj2 = (Comparable)object2;
   if ((obj1).compareTo(obj2) > 0)
      return object1;
   else
      return object2;
}
```

인터페이스 상속

- 인터페이스 간에 상속 가능
  - 인터페이스 상속하여 확장된 인터페이스 작성 가능
- 다중상속 허용

```
public interface AdvancedRemoteControl extends RemoteControl {
        public void volumeUp();       // 가전제품의 볼륨을 높인다.
        public void volumeDown();// 가전제품의 볼륨을 낮춘다.
}
```



다중 상속

- 다중 상속이란 여러 개의 수퍼클래스로부터 상속받는 것인데 자바 클래스 간에는 다중 상속을 지원하지 않는다

```
class SuperA { int x; }
class SuperB { int x; }
class Sub extends SuperA, SuperB          // 만약에 다중 상속이 허용된다면
{
        ...

Sub obj = new Sub();
obj.x = 10;                               // obj.x는 어떤 수퍼 클래스의 x를 참조하는가?
```

- 인터페이스를 이용하면 다중 상속의 효과를 낼 수 있다
- 인터페이스 간에는 다중 상속을 허용한다

- 다중 상속의 효과 : 하나는 클래스의 상속, 하나는 인터페이스의 구현

```
class Shape {
        protected int x, y;
}
interface Drawable {
        void draw();
};
public class Rectangle extends Shape implements Drawable {
        int width, height;
        public void draw() {
                System.out.println("Rectangle Draw");
}
};
```

2개 이상의 인터페이스들의 다중 상속

```
interface Drivable {
  void drive();
}

interface Flyable {
  void fly();
}
```

```
public class FlyingCar1 implements Drivable, Flyable {
  public void drive() {
            System.out.println("I'm driving");
  }

  public void fly() {
            System.out.println("I'm flying");
  }

  public static void main(String args[]) {
            FlyingCar1 obj = new FlyingCar1();
            obj.drive();
            obj.fly();
  }
}
```

I´m driving
I´m flying

상수 공유의 예

```
interface Days {
public static final int SUNDAY = 1, MONDAY = 2, TUESDAY = 3,
        WEDNESDAY = 4, THURSDAY = 5, FRIDAY = 6,
        SATURDAY = 7;
}
```

```
public class DayTest implements Days
{
        public static void main(String[] args)
        {
                System.out.println("일요일: " + SUNDAY);
        }
}
```

상수를 공유하려면
인터페이스를 구현하면 된다.

디폴트 메소드와 정적 메소드

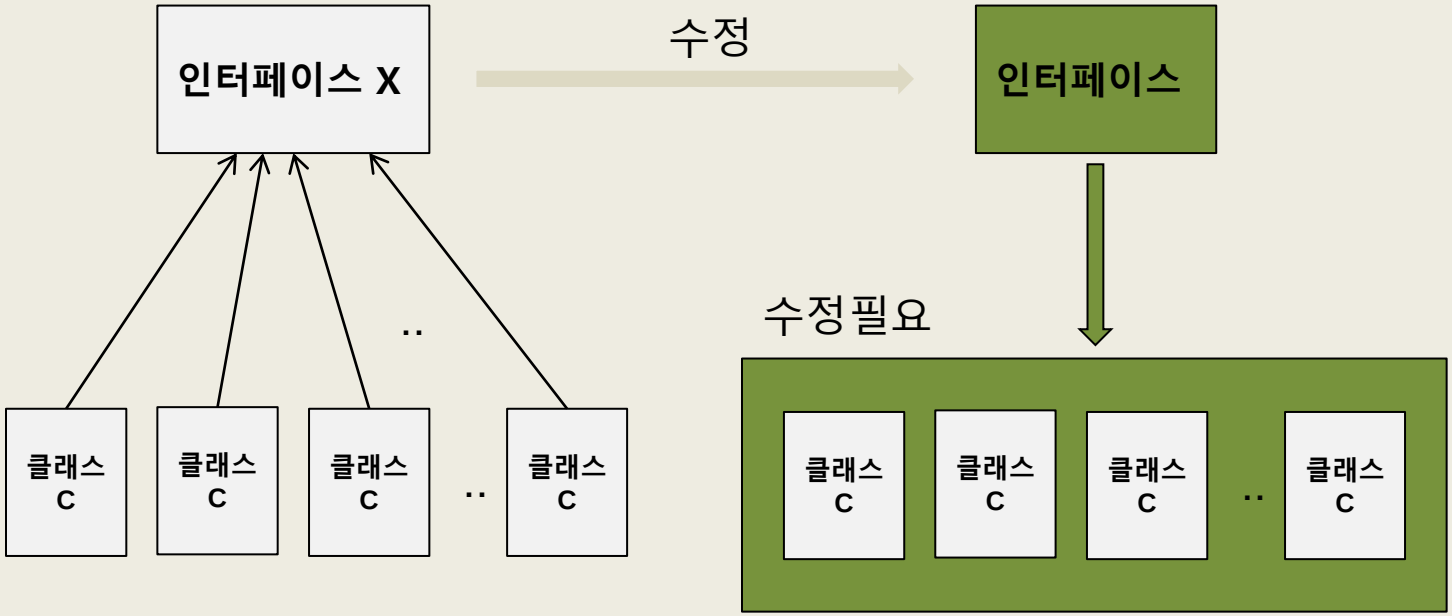
- 디폴트 메소드(default method) – "***default***" 키워드 사용
  - JDK 8부터 적용
  - 인터페이스 개발자가 메소드의 디폴트 구현을 제공
  - 해당 인터페이스를 구현하는 클래스에서 구현하지 않아도 호출 가능

```
interface MyInterface {
   public void myMethod1();
   default void myMethod2() {
            System.out.println("myMethod2()");
   }
}
public class DefaultMethodTest implements MyInterface {
   public void myMethod1() {
            System.out.println("myMethod1()");
   }
   public static void main(String[] args) {
            DefaultMethodTest obj = new DefaultMethodTest();
            obj.myMethod1();
            obj.myMethod2();
   }
}
```

myMethod1()
myMethod2()

Why default method?

- 인터페이스 X를 구현하고 있는 클래스 C가 있다고 할 경우, X에 어떤 추상 메소드를 추가하면, 그 메소드는 클래스 C에 반드시 구현해야 한다
  - 만약에 인터페이스 X를 구현한 클래스가 많다면 그 모든 클래스가 수정되어야 한다
  - 그러므로 그러한 인터페이스에 대해 특정 메소드를 추가하면 클래스에서 구현하지 않아도 호출할 수 있도록 하기 위해서 도입



인터페이스 내의 정적 메소드

- 인터페이스에 정적메소드(static method)의 추가를 허용
  - JDK 8부터 가능
  - Interface 명으로 직접 호출 가능

```
interface MyInterface {
  static void print(String msg) {
            System.out.println(msg + ": 인터페이스의 정적 메소드 호출");
  }
}
public class StaticMethodTest {

  public static void main(String[] args) {
            MyInterface.print("Java 8");
  }
}
```