인터페이스와 특수 클래스

추상클래스

- 추상메소드
  메서드 본체를 완성하지 못한 메서드로 무엇을 할 지 선언은 가능하지만 어떻게 하는지 정의는 할 수 없다.
  추상 메소드 선언하는 방법

```
abstract 반환타입 메서드이름() ;
// 메서드의 본체가 없이 abstract로 추상 메소드라는 것을 나타낸다.
```

- 추상클래스
  보통 하나 이상의 추상 메소드를 포함하지만 없을 수도 있다.
  주로 상속 계층에서 자식 멤버 (필드, 메소드)의 이름을 통일하기 위하여 사용한다.

```
추상 클래스 s = new 추상클래스();
// 추상 클래스는 인스턴스를 생성하지 못한다.
```

따라서 추상클래스를 선언하는 방법이다.

```
abstract class 클래스 이름 {
    // 필드
    // 생성자
    // 메서드
}
```

인터페이스 기초

- 인터페이스 의미
  다른 클래스를 작성할 때 기본이 되는 틀을 제공하면서, 다른 클래스 사이의 중간 매개 역할까지 담당하는 일종의 추상클래스이다.
  즉, 자바의 다형성을 극대화시켜 개발 코드의 수정을 줄이고 프로그램의 유지보수성을 높이기 위해 인터페이스를 사용한다.

- 인터페이스에 의한 장점
  인터페이스만 준수하면 통합에 신경 쓰지 않고 다양한 형태로 새로운 클래스를 개발할 수 있다. 클래스의 다중상속을 지원하지 않지만, 인터페이스로 다중 상속 효과를 간접적으로 얻을 수 있다.

- 인터페이스 VS 추상 클래스

분류	인터페이스	추상 클래스
구현 메서드	포함 불가 (단, 디폴트 메소드와 정적 메소드는 예외)	포함가능 (실제로는 사용하지 않음.)
인스턴스 변수	포함 불가	포함 가능
다중 상속	가능	불가능
디폴트 메서드	선언 가능	선언 불가능
생성자와 main()	선언 불가능	선언 가능
상속에서의 부모	인터페이스	인터페이스, 추상 클래스
접근 범위	모든 멤버를 공개	추상 메서드를 최소한 자식에게 공개

- 인터페이스의 예
  자바가 기본적으로 제공하는 인터페이스는 다양하다.
  대표적인 자바에서 제공되는 인터페이스
  1) java.lang 패키지의 CharSequence, Comparable, Runnable 등
  2) java.util 패키지의 Collection, Comparator, List 등

위의 제공되는 인터페이스 중 Comparable 인터페이스의 예시 (객체의 크기를 비교)

```
public interface Comparable {
int compareTo(Object other); // 객체 other보다 크면 양수, 같으면 0, 작으면 음수 반환
}
public interface Comparable<T> { int compareTo(T o);
}
```

- 인터페이스 구조 interface

```
인터페이스 이름 { // 상수 필드
// abstract 메소드 키워드 생략 가능
-> 상수만 가능하기 때문에 public static final 키워드 생략 가능 -> 인터페이스의 모든 메소드가 public ab
stract이기 때문에
// 아래는 JDK 8부터 가능함.
// default 메소드 // static 메소드 // private 메소드
}
```

- 디폴트(default) 메소드와 정적(static) 메소드
  default 메소드는 오버라이딩이 될 수 있지만 static 메소드는 오버라이딩이 될 수 없다.
  => default 메소드는 implements한 클래스에서 @Override로 재정의 할 수 있지만, static 메

소드는 @Override로 재정의 할 수 없다.
=> default 메소드는 인스턴스 메소드이므로 객체를 생성한 후 호출하지만, static 메소드는 인터페이스로 직접 호출한다.
=> default 메소드는 참조변수를 이용해 호출이 가능하다. 재정의 했을 경우 재정의한 메소드가 호출된다.
=> static 메소드는 "클래스명.static메소드();" 로만 호출이 가능하다. "참조변수.static 메소드();"를 호출하면 에러가 발생한다.
=> default메소드와 static 메소드는 interface에서 메소드 구현이 가능하다.
=> 인스턴스 메소드: 인스턴스 변수와 관련된 작업을 하는 메소드이다.
즉, 메소드의 작업을 수행하는데 인스턴스를 필요로 하는 메소드이다.

- 인터페이스의 구조

```
* 인터페이스 멤버에 명시된 public, static, final abstract 키워드는 생략 가능하다.
* 생략한 키워드는 컴파일 과정에서 자동으로 추가 된다.
* 인터페이스 파일 확장자는 .java 이다.
* 컴파일하면 확장자가 class인 파일을 생성한다.
```

```
interface MyInterface {
int Max = 10; void sayHello();
}
```

MyInterface.java

⬇

```
interface MyInterface {
// 상수 필드 (public static final) // 추상 메소드
public static final int Max = 10;
public abstract void sayHello(); }
```

MyInterface.class

- 인터페이스 상속
  인터페이스를 상속하려면 extends 키워드를 사용한다.

```
interface 자식인터페이스 extends 부모인터페이스 {
}
```

인터페이스를 구현하려면 implements 키워드를 사용한다.

```
class 자식 클래스 implements 부모 인터페이스 {
}
```

+) 다중상속

```
interface 자식인터페이스 extends 부모인터페이스1,부모인터페이스2 { }
class 자식 클래스 implements 부모인터페이스1, 부모인터페이스2 { }
class 자식 클래스 extends 부모 클래스 implements 부모인터페이스1, 부모인터페이스2 {
  }
```

+) 클래스의 다중 상속은 불가능

```
// 안되는 예시
class 자식클래스 extends 부모클래스1, 부모클래스2 { }
```

인터페이스 응용

- 인터페이스의 상속과 구현 클래스

```
public interface Controllable {
    default void repair() { show(“수리한다.”); }
    static void reset() { System.out.println(“초기화”);}

private void show(String s){System.out.println(s); }
    void turnOn();
    void turnOff();
}

public interface Remote Controllable extends Controllable {
    void remoteOn();
    void remoteOff();
}
public class TV implements Contorllable {
    @Override
    public void turnOn() { System.out.println(“TV 켠다.”);
    }

    @Override
    public void turnOff() {
    System.out.println(“TV 끈다.”); }
    }
    public class Computer implements Controllable {
        public void turnOn() { System.out.println(“컴퓨터 켠다.”); }
        public void turnOff() { System.out.println(“컴퓨터 끈다.”); }
    }

    public class CintrollableDemo {
        public static void main(String[] args) {
            TV tv = new TV();
            Computer com = new Computer();
            tv.turnOn(); // TV 켠다.
            tv.turnOff(); // TV 끈다.
            tv.repair(); // 수리한다.
            com.turnOn(); // 컴퓨터를 켠다.
            com.turnOff(); // 컴퓨터를 끈다.
            com.repair(); // 수리한다.
```

```
            Controllable.reset(); // 초기화
        }
}
```

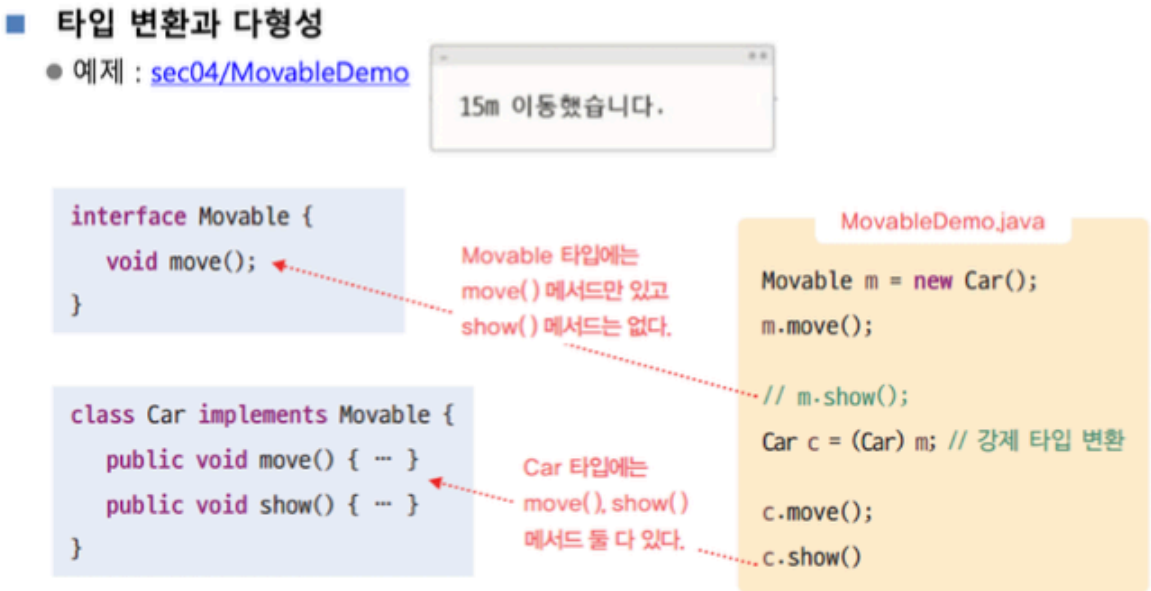
인터페이스와 다형성

- 인터페이스 타입
  인터페이스는 클래스처럼 하나의 타입으로 변수를 인터페이스 타입으로 선언 가능 인터페이스의 구현클래스는 그 인터페이스의 자식 타입
  => 인터페이스타입 변수 = 구현객체

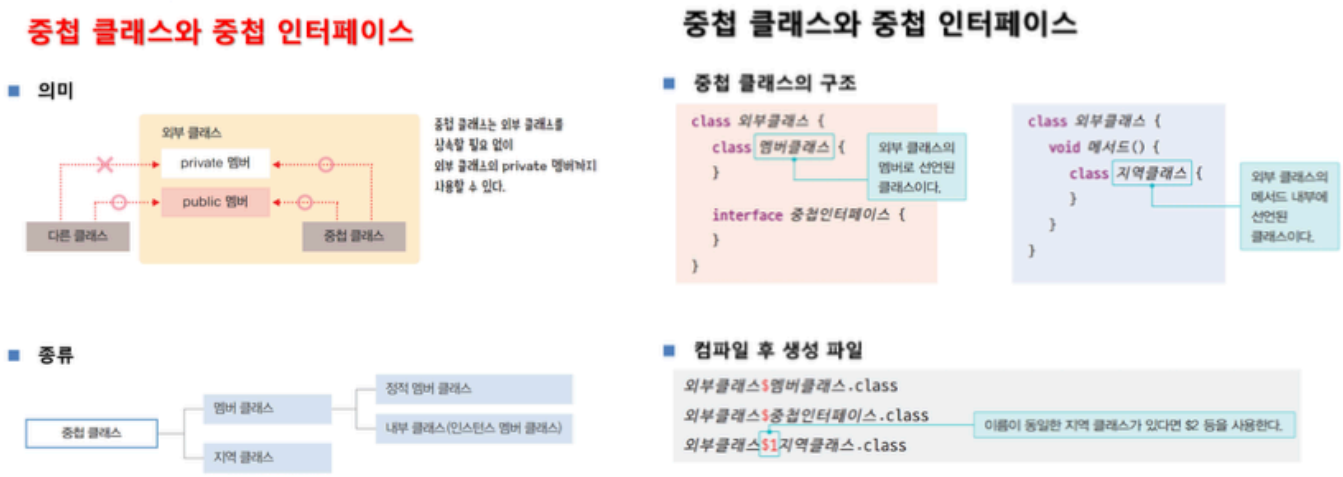
```
* 구현 객체는 인터페이스 타입으로 자동 변환 된다.
```

인터페이스 타입 변수가 구현 객체를 참조한다면 강제 타입 변환 가능
=> 구현클래스타입 변수 = (구현클래스타입) 인터페이스타입변수

```
* 구현클래스타입: 타입 변환 연산자
* 인터페이스타입변수: 인터페이스 구현 객체를 참조하는 변수
* 중첩 클래스와 중첩 인터페이스
* 외부 클래스 접근 외부클래스.this
```



중첩 클래스의 객체 생성



외부 클래스 접근

```
외부클래스.this
```

중첩 클래스의 객체 생성

```
외부클래스.내부클래스 변수 = 외부클래스의 객체변수.new 내부클래스생성자();
외부클래스.정적멤버클래스 변수 = new 외부클래스.정적멤버클래스생성자();
```

- 익명 클래스
  중첩 클래스의 특수한 형태로 코드가 단순해지기 때문에 이벤트 처리나 스레드 등에서 자주 사용