6주차: 인터페이스

https://github.com/KMUCS-MoGakKo/Java_Study/issues/15

드디어 인터페이스 차례다.
인터페이스는 클래스부터 상속, 패키지 지난 주제에서 꾸준히 나왔다. 그 때마다 인터페이스 차례가 되면 해야지! 하고 미뤘 넘겼다가 드디어 지금이 되었다.

인터페이스

: 인터페이스는 클래스들이 구현해야하는 동작(메소드)를 지정하는데 사용되는 추상(abstract) 자료형이다. 쉽게 말하자면, 모든 메소드들이 추상 메소드인 자료형으로 생각하면 쉬울 것 같다.

그럼 앞에서 배웠던 추상 클래스도 추상 메소드가 들어가는데 모든 메소드들을 인터페이스라고 따로 지정해야하는 이유는 무엇일까?
그건 인터페이스와 추상 클래스의 차이점을 살펴보면 바로 알 수 있다.
추상 클래스는 그 추상 클래스를 상속 받아, **기능을 이용 및 확장**에 있다면 **인터페이스**는 함수의 **구현을 강제**하기 위해 그런 함수들을 모아 포장해둔 것이라 할 수 있다.
앞서 클래스에서 **다중 상속**을 언급하고 간 바가 있다. 자바는 클래스의 다중 상속을 막았기 때문에(다이아몬드 문제) 인터페이스(의 다중 상속)를 통해 이를 해결하려 하였다.

참고하면 좋은 링크 : [JAVA] abstract와 interface의 차이점
위 링크가 진짜 깔끔하게 정리 되어 있어서 첨부함.

- **인터페이스 정의하는 방법**

```
접근제어자 interface 인터페이스명 {
      public static final 타입 상수명 = 값;
      public abstract 메소드명(매개변수);
}
```

인터페이스 정의에는 `interface` 키워드를 사용한다.
단, 모든 필드가 `public static final` 이어야하고, 모든 메소드는 `public abstract` 이어야 한다.
인터페이스의 존재 이유를 생각하면 당연하다. 인터페이스를 구현하는 구현 클래스에서 구현이 필수적이기 때문에 접근이 용이하도록 public 을 기본으로 하는 것이다. 너무도 당연한 것+공통적인

것이기 때문에 이 제어자는 생략할 수 있고, 생략된 제어자는 컴파일 시 자바 컴파일러가 자동으로 추가해준다.

- **인터페이스 구현하는 방법**

```
class 클래스명 implements 인터페이스명 { ... }
```

`interface` 키워드를 사용하여 정의된 인터페이스는 `implements` 키워드를 사용하여 구현할 수 있다.

보통은 위와 같이 구현 클래스를 만들어 사용하는 것이 일반적이고, 재사용도 할 수 있으니 편리하지만 **일회성**일 경우 소스파일을 만들고 클래스를 선언하는 것이 비효율적이기 때문에 인터페이스 레퍼런스를 사용하여 main 클래스에서 직접 오버라이딩 하여 사용하는 방법도 있다. 이를 **익명 구현 객체** 라고도 한다. 예시 코드는 다음과 같다.

```
public class main {
	public static void main(String[] args) {
		TestInterface test1 = new TestInterface() {
			@Override
			public void getTest() {
				System.out.println("Test1");
			}
		}
		test1.getTest(); // 실행 결과: Test1
	}
}
```

위와 같이 인터페이스를 직접 new 연산자로 새 객체를 만들어주고 내부에 추상 메소드들을 오버라이딩해주면 된다.

- **인터페이스 레퍼런스를 통해 구현체를 사용하는 방법**

솔직히 처음에 이 말을 봤을 땐 음? 하고 바로 이해가 안되었다. 일단 예시 코드부터 보자.

```
public interface TestInterface {
	int test = 0;
	String getTest();
}

public class Test1 {
	@Override
	public String getTest() {
		return "Test1";
```

```
	}
}

public class Test2 {
	@Override
	public String getTest() {
		return "Test2";
	}
}

public class Main {
	public static void main(String[] args) {
		TestInterface test1 = new Test1();
		TestInterface test2 = new Test2();
	}
}
// 결과
// Test1
// Test2
```

괜히 인터페이스 레퍼런스를 통해 구현체 사용하는 방법이라고 하니까 뭔가 읽기 싫고... 그랬는데 그냥 다형성을 이용한 것이었다.

저번에 캡슐화, 상속, 다형성에 대하여 공부하다가 캡슐화는 클래스에서, 상속은 상속 파트에서 공부를 하고 다형성은 인터페이스 때 하려고 미뤄뒀었다. 그런데 사실 까보면 이미 다 했던 내용이긴 하다. 대충 정리만 해보자.
다형성의 의미는 하나의 객체가 여러 가지 타입을 가질 수 있는 것을 의미한다.
자바에서는 다형성을 크게 4가지 방법으로 구현하고 있다.

1. 상속 클래스 구현(부모-자식 클래스 구현)
2. 메소드 오버라이딩
3. 업캐스팅하여 객체 선언
4. 부모 클래스 타입의 참조 변수로 자식 클래스 타입의 인스턴스를 참조

위 인터페이스 레퍼런스를 통해 구현체 사용은 4번에 해당한다고 할 수 있다.

- **인터페이스 상속**

인터페이스만 가지는 상속의 특징을 살펴보자.

1. 인터페이스는 인터페이스끼리만 상속 가능.
2. 다중 상속 가능.

사실 2번은 클래스와 상속 파트를 거치며 많이 봐서 이제 알 것 같고... 1번도 인터페이스는 구현을 강제하기 위해 만든 자료형이다보니 구현이 완료된 다른 메소드들을 받아오면 안되기 때문에 당연하다. 한 가지 더 추가해보자면,

3. 클래스와 인터페이스를 같이 상속 및 구현했을 경우 **클래스에 존재하는 메소드가 우선권**을 가진다.

이 역시, 설계도에 불과한 인터페이스보다는 구현이 완료된 클래스의 메소드가 우선권을 갖는 것이 당연하다.

다음 나오는 default method 와 static method 는 Java 8로 업데이트 되며 인터페이스의 구현과 관련이 있다.

Default Method가 나온 이유는 **하위 호환성** 때문이다.

기존 인터페이스를 이용해서 구현된 클래스들을 만들고 사용하고 있는 중, 인터페이스를 **보완하는 과정**에서 추가적으로 구현해야할 필수적인 메소드가 생긴다면, 이미 이 인터페이스를 구현한 클래스와는 호환성이 떨어 지게 되기 때문에 default 메소드를 추가하여 필수적으로 구현해야 할 메소드를 정의하면 하위 호환성은 유지시키며 인터페이스를 보완할 수 있다.

더 자세한 내용은 인터페이스 때 알아보도록 하자!

4주차 상속에서 이렇게 넘어간 적이 있는데 자세히.. 알아볼 시간이 온 것이다...

- **인터페이스의 기본 메소드 (Default Method), 자바 8**

기본 메소드

: 인터페이스에 메소드 선언이 아니라 구현체를 제공하는 방법

기본 메소드가 나오게 된 이유는 인터페이스 상속의 아래 항목 때문이다.

어떤 인터페이스를 구현하는(implements) 클래스는 반드시 자신이 구현하는 인터페이스에 선언된 모든 메소드를 구현해야 한다. 그렇지 않으면 오류가 발생한다.

인터페이스 A 를 상속받은 a, b, c 클래스가 있다고 생각해보자.
만약 A에 새 메소드를 추가하고 싶다. 그러면 a, b, c 에 새 메소드를 추가해주어야한다. 예시에선 3개뿐이지만 실제로 한다면 일일이 A를 상속받은 모든 클래스를 찾는 것부터 일이다.
이런 경우를 대비해서 기본 메소드는 사전에 인터페이스에 미리 구현을 해두는 것이다.
그 후, 새로운 d 클래스가 A 인터페이스를 상속하여 새 메소드를 사용하고 싶을 때, 다음과 같이 사용할 수 있게 되는 것이다.

```
public class d클래스 implements A인터페이스 {
	A인터페이스의 기존 존재하던 메소드들 { ... }
	@Override
	public void 새메소드 { ... }
}
```

단, 기본 메소드는 구현체가 모르게 추가된 기능이기 때문에 리스크가 있다. 구현체에 따라 런타임 에러가 발생할 수도 있고 반드시 문서화가 필요하다. (@implSpec 태그 사용)

- **인터페이스의 static 메소드, 자바 8**

static 메소드는 오라클 공식문서에서는 인터페이스에서 helper 메소드로 사용하는 것을 예시로 들고 있다. (helper 메소드란 주목적으로서 사용되는 것이 아닌, 다른 객체를 돕기 위한 특정 목적으로 만들어진 메소드)
클래스의 static 메소드처럼 사용하기보다는, 인터페이스 내부적으로 필요한 것을 정의해두고 사용하는데 목적이 있다고 추측된다. 기본 메소드와는 달리 Override가 불가능하다.

- **인터페이스의 private 메소드, 자바 9**

private 메소드는 뭘까... 아까 위에서 public 이 기본이라고 했는데 Java9 부터는 private 접근제어자를 가능하도록 하였다. 도대체 왜 필요한걸까???
결과만 말해보자면 private 메소드를 사용하면 코드의 수정과 유지보수가 쉬워지고 재사용성이 올라간다는 장점이 있기 때문이다.

만약 private 메소드가 없는 상태에서 디폴트 메소드 5개, 스태틱 메소드 5개를 가진 인터페이스가 있다고 생각해보자. 이 10개의 메소드가 동일한 작업을 해야하는 로직이 포함된다고 할 때, 이 10개의 메소드를 일일이 구현하는 것은 매우 비효율적이다.
 이때 private 메소드를 사용하여 하나를 제외한 나머지 9개의 메소드를 다른 클래스에서 상속받을 수 없도록하면 코드 구현도 편하고, 효율적이다.

뭔가.. Java 8, 9에 이어 저런 기능들이 구현되니 기존 클래스와 차이점이 점점 희미해지는 것 같다... 그래도 역시 인터페이스는 다중 상속이 된다는 점과(인터페이스의 기본 메소드들은 public 이고, 예외들이 default, static 과 같은 구현 메소드들이니) 상속 관계에서의 IS-A 관계인지 HAS-A 관계인지가 가장 큰 것 같다.

참고 링크

- 자바의 추상 클래스와 인터페이스
- 자바의 인터페이스(interface)란?
- https://seeminglyjs.tistory.com/211
- Live Study_Week 08. 인터페이스
- JAVA 8(2) - 기본 메소드와 스태틱 메소드 - Dev Blog
- [JAVA] 인터페이스(Interface)의 기본 개념과 Java 8 이후의 변화
- 8주자 과제: 인터페이스 · Issue #8 · whiteship/live-study