2주차(1)

상속 - 248

api보기 - http://www.oracle.com 접속 - DownLoads마우스 오버 - 왼쪽에 Java for Developers클릭 -
우측 탭에 Java Resources 아래 APIs - Java SE7클릭

상단의 Tree는 계층도이고, 중간에 See Also는 해당 클래스를 상속받는 객체들이다.

계층도를 보면 Object가 최상위(단군 할아버지)인걸 알 수가 있다.

일단, 상속이라 함은 부모가 보유하고 있는 재산 중 일부를 자식이 물려받는 것.

1. extends(상속)

클래스 상속은 객체의 재사용이라는 장점뿐 아니라, 코드의 간결성을 제공해주는 객체지향 언어의 최고 장점이라 할 수 있다.
그러므로 잘 정의된 부모클래스(코드 진행하면서 설명)가 있다면 자식 클래스의 작성이 간편해진다는 장점이 있지렁

Parent 클래스 정의

```
public class Parent {
	private int money = 2000000000; //부모의 재산
	private String str = "신촌"; //부모의 부동산

	public int getMoney() {
		return money;
	}

	public void setMoney(int money) {
		this.money = money; //부모의 재산
	}

	public String getStr() {
		return str;
	}

	public void setStr(String str) {
		this.str = str; //부모의 부동산
	}
}
```

Child클래스 정의(Parent를 상속받는다)

```
public class Child extends Parent{
	private String car = "아우디";

	public String getCar() {
		return car;
	}

	public void setCar(String car) {
		this.car = car;
	}
}
```

Main클래스 정의

```
public class Main {
	public static void main(String[] args) {
		Child c1 = new Child();

		//Child클래스에 있는 getCar호출
		System.out.println(c1.getCar());

		//자식이 부모의 재산을 모두 상속받았다고 가정하자.
		//Child클래스에는 getMoney()라는 함수가 없지만
		//부모로 부터 상속받은 getMoney()를 호출 가능하다
		System.out.println(c1.getMoney());

		// Child는 Parent로부터 상속 받았으므로
		// Child는 곧 Parent다.

		//왼쪽의 객체가 오른쪽의 클래스의 인스턴스(즉, 그 클래스로 만든 객체)인
			지를 확인해서(instanceof연산자)
		//맞으면 true, 틀리면 false를 반환
		if(c1 instanceof Parent)
			System.out.println("c1은 Parent의 자식");

		System.out.println("----------------------------");

		//예제 2 함께 보시죠
		Child ch = new Child();
		Parent p1 = new Parent();
```

```
		//자식에 부모를 대입해보자.
		//자식을 부모로 캐스팅 하고
		//부모를 다시 자식으로 캐스팅해서 대입
		//이렇게 쓰는경우는 많지않다. 테스트를 위해 작성함
		p1 = (Parent)ch;
		ch = (Child)p1;

		//자식에 부모를 대입해도 자식은 자신의 기능을 포함해 부모의 모든 기능
			을 사용할 수 있다.
		System.out.println(ch.getCar());
		System.out.println(ch.getMoney());
		System.out.println(ch.getStr());
		System.out.println("----------------------------");

		//반면
		p1 = ch; //부모에 자식을 대입해도
		//부모는 부모의 기능만 사용할 수 있다.
		System.out.println(p1.getMoney());
		System.out.println(p1.getStr());

		//부모가 자식의 메서드를 쓰고 싶다면 캐스팅 해야하는데
		//좋은 사용법이 아니다. 안쓴다고 보면 됨
		//System.out.println(((Child)p1).getCar());
	}
}
```

----------예제 1

Animal클래스 정의

```
public class Animal {

	private int eye = 2; //일반적인 동물들의 눈 갯수라고 가정
	private int leg = 4; //일반적인 동물들의 다리 개수라고 가정

	public int getEye() {
		return eye;
	}
	public int getLeg() {
		return leg;
	}
```

```
}
```

Bear클래스 정의(Animal상속)

```
public class Bear extends Animal{
	//곰은 눈이 2개, 다리가 4개 이므로
	//eye, leg는 부모의 것을 가져다 쓰면 된다
	String woong = "웅담";
}
```

Lion클래스 정의(Animal상속)

```
public class Lion extends Animal{
	//사자도 눈이 2개, 다리가 4개 이므로
	//eye, leg는 부모의 것을 가져다 쓰면 된다
	String galgi = "어휴 풍성~";
}
```

Spider클래스 정의(Animal상속)

```
public class Spider extends Animal{

	String web = "슉슉";

	//메서드 오버라이딩. 일전에 배웠던 오버로딩과는 다름.
	//'메서드 재 정의'의 의미를 가지며, 상속관계의 객체에서
	//부모의 함수를 가져와 자식의 사정에 맞게 재정의 하는것이다.
	//오버라이딩 메서드는 내용을 제외하고는 부모의것과 완전히 동일해야 한다.
	//(접근제한은 부모것 보다 넓은 범위에서 변경 가능)
	@Override
	public int getEye() {
		return 6;
	}

	@Override
	public int getLeg() {
		return 8;
	}

}
```

AnimalMain클래스 정의

```
public class AnimalMain {
	public static void main(String[] args) {
```

```
                Bear b = new Bear();
                System.out.println( "---곰---" );
                System.out.println( "눈 : " + b.getEye() );
                System.out.println( "다리 : " + b.getLeg() );

                Lion l = new Lion();
                System.out.println("---사자---");
                System.out.println("눈 : " + l.getEye());
                System.out.println("다리 : " + l.getLeg());

                Spider s = new Spider();
                System.out.println("---거미---");
                System.out.println("눈 : " + s.getEye());
                System.out.println("다리 : " + s.getLeg());

        }//main
}
```

예제2

```
//아주 간단한 오버라이딩 예제를 만들어 봅시다.

//Calculator클래스를 만들고
//getResult()함수를 정의하세요. 반환형은 정수.
//인자 두개(n1, n2)를 받고 -1로 리턴하게 만듭니다.

//CalPlus클래스를 만들어 Calculator클래스를 상속받으세요.
//오버라이딩을 이용하여 Calculator의 getResult()함수를
//인자로 받은 n1과 n2를 더해주는 함수로 만듭니다.
//물론 리턴값도 -1이면 안되겠죠??

//CalMinus클래스를 만들어 Calculator클래스를 상속받으세요.
//오버라이딩을 이용하여 Calculator의 getResult()함수를
//인자로 받은 n1과 n2를 빼주는 함수로 만듭니다.

//Main에서 실행하여 아래와 같은 결과가 나오면 성공
//CalPlus : 30
//CalMinus : 15
```

풀이

Calculator클래스 정의

```
public class Calculator {
	public int getResult(int n1, int n2){
		return -1;
	}
}
```

CalPlus클래스 정의

```
public class CalPlus extends Calculator{
	@Override
	public int getResult(int n1, int n2) {
		// TODO Auto-generated method stub
		return n1 + n2;
	}
}
```

CalMinus클래스 정의

```
public class CalMinus extends Calculator{
	@Override
	public int getResult(int n1, int n2) {
		// TODO Auto-generated method stub
		return n1 - n2;
	}
}
```

CalMain클래스 정의

```
public class CalMain {
	public static void main(String[] args) {
		CalPlus cp = new CalPlus();
		System.out.println("CalPlus : " + cp.getResult(10, 20));

		CalMinus cm = new CalMinus();
		System.out.println("CalMinus : " + cm.getResult(30, 15));
	}
}
```

예제 3

super()의 활용- page 258

이전에 이야기 했듯이 super는 부모클래스를 의미한다.
super()는 부모클래스의 생성자를 호출하는 것. 간단하게 super()를 알아보자.

Parent클래스 정의

```
public class Parent {
```

```
	public Parent(int n) {
		System.out.println("부모(Parent)클래스" + n);
	}//생성자
}
```

Child클래스 정의

```
public class Child extends Parent{
	public Child() {
		super(1);
		System.out.println("자식(Child)클래스");
	}//생성자
}
```

SuperMain클래스 정의

```
public class SuperMain {
	public static void main(String[] args) {
		Child ch = new Child();
		//자식클래스를 생성하면
		//자식의 생성자가 호출되는데, 여기서는 자식클래스에서 super()로
		//부모의 생성자를 먼저 호출했으므로,
		//부모클래스의 생성자에 먼저 접근하게 된다.
	}
}
```

2. Object상속관계

ExtendsEx1클래스 정의

```
public class ExtendsEx1 {
	// 현재 클래스의 특정 메서드가 어떤때는 String을 인자로 받고,
	// 어떤때는 int형을 인자로 받는등, 상황에 따라 다른 인자값을
	// 받아야 한다면....
	// 멤버변수를 어떻게 선언해야 할까?
	// 오버로딩 하는 방법도 있지만 이런 방법도 있다규요

	Object value; //자바 객체의 최상위인 Object형으로 변수생성

	public Object getValue() {
		return value;
	}

	public void setValue(Object value) {
```

```
                this.value = value;
        }
}
```

ExtendsEx1_Main클래스 정의

```
public class ExtendsEx1_Main {
        public static void main(String[] args) {

                ExtendsEx1 v1 = new ExtendsEx1();
                v1.setValue("TEST");
                // 인자가 Object이지만 String이 Object를
                // 상속받았으므로 인자로 가능하다.

                System.out.println(v1.getValue()); // TEST

                //이번에는 정수(int)를 인자로 넣어보자!!
                ExtendsEx1 v2 = new ExtendsEx1();
                v2.setValue(100); // AutoBoxing(자동 형변환 : 기본자료형->객체)

                //int i = (Integer)v2.getValue();//Object를 Integer로 형변환 -강제 형변환
                //예전엔 이렇게 객체로 캐스팅하여 썼어야 했다.↑↑↑↑

                //지금은 이렇게 기본자료형으로 캐스팅해도 사용할수 있도록 바뀐 듯
                //UnBoxing - 객체 -> 기본자료형
                int i = (int)v2.getValue();
                System.out.println(i+1);
        }
}
```

3. work / 0415 / src / ex2

제네릭(Generic) 타입 (클래스명<T>)클래스

제네릭 클래스란?
제네릭 프로그래밍이란 일반적인 코드를 작성하고 이 코드를 다양한 타입의 객체에 대하여 재사용하는 객체 지향 기법이다.

원하는 타입의 객체만 받아들이기 위한 방법이라고 생각해도 좋다.

결국 하나의 메소드에서 여러가지 타입을 한번에 다 지원 하는 클래스가 제네릭 클래스라 할 수 있다.

클래스를 하나 만들었는데, 내부 메서드에서 int만이 아니라 원하는 자료형(객체) 타입 으로 (String, char) 다양하게 지원하고 싶을 때 사용.

설명이 어렵죠? 예제보자

GenEx<T>클래스 정의
<T>부분은 GenEx로 먼저 클래스를 만든 후에 추가. 이름짓는 과정에서는 <T>가 안들어감 ㅋ

```
public class GenEx<T>{
	T value;

	public T getValue() {
		return value;
	}

	public void setValue(T value) {
		this.value = value;
	}
}
```

GenEx_Main클래스 정의

```
public class GenEx_Main {
	public static void main(String[] args) {
		// 사용자가 원하는 형태로 객체 생성
		GenEx<String> v1 = new GenEx<String>();
		v1.setValue("100");

		System.out.println(v1.getValue());

		// 정수를 가지는 GenEx객체를 생성하자!
		// 제네릭 타입은 기본자료형을 인식하지 않음
		// 따라서 int, double등의 기본자료형을 제네릭타입으로 이용하고자 할 때
		// 는 Integer, Double등의 클래스를 이용해야 한다.
		GenEx<Integer> v2 = new GenEx<Integer>();
		v2.setValue(1000);
		System.out.println(v2.getValue()+10);

		GenEx<Character> v3 = new GenEx<Character>();
		v3.setValue('A');
		System.out.println(v3.getValue());
```

```
                GenEx<Double> v4 = new GenEx<Double>();
                v4.setValue(3.14);
                System.out.println(v4.getValue());
        }
}
```
---------- 예제1

다른거 한번 해볼까요(문제로 내도되고 같이해봐도 됨)

Gen클래스를 만들어 제네릭 타입T를 갖는 printArr메서드를 생성한다.
printArr메서드 내부에서 배열을 순차적으로 보여줄수 있게 하는 코드를 작성.

Main클래스를 만들어 Integer[], Double[], Character[]을 각각 만든 뒤
Gen클래스의 printArr메서드를 각각 호출하여 배열의 내용을 출력하도록 해보자.

결과 :
```
 1 2 3 4 5  //정수배열 출력
 1.1 2.2 3.3 4.4 5.5 //실수배열 출력
 A B C D E //문자배열 출력
```

Gen클래스 정의
아래의 초록색 <T>는 둘 중에 한 개만 넣어주면 되지만,
메서드에서 제네릭 타입을 사용할 경우 메서드쪽에 넣어주는 것이 더 좋다.
```
public class Gen <T> {
        public <T> void printArr(T[] arr){

                for(int i = 0; i < arr.length; i++){
                        System.out.print(" " + arr[i]);
                }
                System.out.println();
        }
}
```

GenEx클래스 정의
```
public class GenEx{
        public static void main(String[] args) {
                Gen gen = new Gen();

                // 정수형
                Integer[] iArr = {1, 2, 3, 4, 5};
```

```
                // 더블형
                Double[] dArr = {1.1, 2.2, 3.3, 4.4, 5.5};

                // Character
                Character[] cArr = {'A', 'B', 'C', 'D', 'E'};

                //제네릭 이용
                //위의 배열들을 int, double, char와 같은 기본자료형으로 만들었다면
                //아래의 메서드에 대입할수 없다.
                //제네릭 타입은 반드시 객체를 처리하록 되어있다.
                gen.printArr(iArr);
                gen.printArr(dArr);
                gen.printArr(cArr);
        }
}
```

예제2