제네릭 심화

컴퓨터공학전공
박요한

수업내용

- 제네릭 메소드
- 와일드카드
- 제네릭 인터페이스

제네릭의 이해

- 제네릭이란?
  - ✓사전적 의미 : 포괄적인, 회사 이름이 붙지 않은, 일반 명칭으로 판매되는

- 제네릭 in OOP?
  - ✓타입을 매개변수(파라미터)화 해서 컴파일시 구체적인 타입이 결정되도록 하는 것
  - ✓즉, 여러 자료형이 대체되도록 프로그래밍 하는 것
  - ✓변수가 하나의 자료형에 국한되지 않고 여러 자료형에 쓰일 수 있도록 프로그래밍 하는 방식

제네릭의 이해

- 제네릭을 사용하는 코드의 이점
  - ✓컴파일 시 강한 타입 체크 가능
    - 실행 시 타입 에러가 나는 것 방지
    - 컴파일 시에 미리 타입을 강하게 체크해서 에러 사전 방지
  - ✓타입변환 제거 가능

```
List list = new ArrayList();
list.add("hello");
String str = (String) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
String str = list.get(0);
```

호출

```
class AAA
{
	public String toString()
	{
		return "Class AAA";
	}
}

class BBB
{
	public String toString()
	{
		return "Class BBB";
	}
}

class InstanceTypeShower
{
	int showCnt=0;

	public <T> void showInstType(T inst)
	{
		System.out.println(inst);
		showCnt++;
	}

	void showPrintCnt()
	{
		System.out.println("Show count: "+showCnt);
	}
}
```

```
class IntroGenericMethod
{
	public static void main(String[] args)
	{
		AAA aaa=new AAA();
		BBB bbb=new BBB();

		InstanceTypeShower shower=new
InstanceTypeShower();
		shower.<AAA>showInstType(aaa);
		shower.<BBB>showInstType(bbb);
		shower.showPrintCnt();
	}
}
```

제네릭 메소드의 정의와 호출

```
class AAA
{
	public String toString()
	{
		return "Class AAA";
	}
}

class BBB
{
	public String toString()
	{
		return "Class BBB";
	}
}

class InstanceTypeShower2
{
	public <T, U> void showInstType(T inst1, U inst2)
	{
		System.out.println(inst1);
		System.out.println(inst2);
	}
}
```

```
class IntroGenericMethod2
{
	public static void main(String[] args)
	{
		AAA aaa=new AAA();
		BBB bbb=new BBB();

		InstanceTypeShower2 shower=new
InstanceTypeShower2();
		shower.<AAA, BBB>showInstType(aaa,
bbb);
		shower.showInstType(aaa, bbb);
	}
}
```

제네릭 메소드와 배열

- 배열도 인스턴스이므로 제네릭 매개변수에 전달이 가능

- 다음과 같이 매개변수를 선언하면, 매개변수에 전달되는 참조값을 배열 인스턴스의 참조 값으로 제한할 수 있다.
  - ✓ T[ ] arr
  - ✓ 그리고 이렇게 되면 참조 값은 배열 인스턴스의 참조 값임이 100% 보장 되므로 [ ] 연산을 허용한다.

제네릭 메소드와 배열

```
class IntroGenericArray
{
	public static <T> void showArrayData(T[] arr)
	{
		for(int i=0; i<arr.length; i++)
			System.out.println(arr[i]);
	}

	public static void main(String[] args)
	{
		String[] stArr=new String[]{
					"Hi!",
					"I'm so happy",
					"Java Generic Programming"
		};
		Integer[] arr = {1,2,3,4,5};
		showArrayData(stArr);
	}		showArrayData(arr);
}
```

매개변수의 자료형 제한

```
class Apple {
    public String toString() {
        return "I am an apple.";
    }
}

class Orange {
    public String toString() {
        return "I am an orange.";
    }
}

class Durian {
    public String toString() {
        return "I am a durian.";
    }
}
```

```
class Box<T> {
        private T ob;

        public void set(T o) {
        ob = o;
        }

        public T get() {
        return ob;
        }
}
```

매개변수의 자료형 제한

```
public abstract class Eatable {
        public abstract void eat();
}
```

```
class Apple {
    public String toString() {
        return "I am an apple.";
    }
}

class Orange {
    public String toString() {
        return “I am an orange.”;
    }
}

class Durian {
    public String toString() {
        return "I am a durian.";
    }
}
```

```
class FruitAndBox2_Generic {
    public static void main(String[] args) {
        Box<Apple> aBox = new Box<Apple>();
        Box<Orange> oBox = new Box<Orange>();



        aBox.set(new Apple());
        oBox.set(new Orange());




        Apple ap = aBox.get();
        Orange og = oBox.get();

        System.out.println(ap);
        System.out.println(og);
    }
}
```

와일드카드

- Eatable 클래스와 Apple, Orange 클래스와의 관계는?
- Box<Apple>, Box<Eatable>의 관계는?
- Box<Eatable>과 Box<Apple>의 인스턴스를 매개변수로 선언할 수 없을까?

- 자바는 와일드카드를 이용한 자료형의 명시를 허용
- 와일드카드?
  - ✓이름 또는 문자열에 제한을 가하지 않음을 명시하는 용도로 사용되는 특별한 기호(?)

와일드카드

```
class FruitAndBox2_Generic {
    public static void main(String[] args) {
        Box<Apple> aBox = new Box<Apple>();
        Box<Eatable> eBox = new Box<Eatable>();

        aBox.set(new Apple());
        eBox.set(new Eatable());

        openAndShowBox(aBox);
        }
}
```

```
public static void openAndShowBox(Box<? extends Eatable> box) {

            System.out.println(box.get());
            box.get.eat();
}
```

와일드카드 타입

- 와일드카드 타입의 세가지 형태
  - 제네릭타입<?> : Unbounded Wildcards (제한없음)
    타입 파라미터를 대치하는 구체적인 타입으로 모든 클래스나 인터페이스 타입이 올 수 있다.
  - 제네릭타입<? extends 상위타입> : Upper Bounded Wildcards (상위 클래스 제한)
    타입 파라미터를 대치하는 구체적인 타입으로 상위 타입이나 하위 타입만 올 수 있다.
  - 제네릭타입<? super 하위타입> : Lower Bounded Wildcards (하위 클래스 제한)
    타입 파라미터를 대치하는 구체적인 타입으로 하위 타입이나 상위 타입이 올 수 있다.

클래스를 제한하는 용도의 와일드 카드

```
FruitBox<? extends Apple> boundedBox;
```

- ~을 상속하는 클래스라면 무엇이든지
- Apple을 상속하는 클래스의 인스턴스라면 무엇이든지 참조 가능한 참조변수 선언

```
FruitBox<? super Apple> boundedBox;
```

- ~이 상속하는 클래스라면 무엇이든지
- Apple이 상속하는 클래스의 인스턴스라면 무엇이든지 참조 가능한 참조변수 선언

제네릭 타입의 상속과 구현

- 제네릭 타입을 부모 클래스로 사용할 경우
  - ✓타입 파라미터는 자식 클래스에도 기술해야 !!!

```
public class ChildProduct<T, M> extends Product<T, M> { ... }
```

  - ✓추가적인 타입 파라미터 가질 수 있음

```
public class ChildProduct<T, M, C> extends Product<T, M> { ... }
```

- 제네릭 인터페이스를 구현할 경우
  - ✓제네릭 인터페이스를 구현한 클래스도 제네릭 타입

제네릭 타입의 상속과 구현

```
interface Getable<T> {
    public T get();
}

class Box<T> implements Getable<T> {
    private T ob;
    public void set(T o) { ob = o; }

    @Override
    public T get() {
        return ob;
    }
}
```

```
class Toy {
    @Override
    public String toString() {
        return "I am a Toy";
    }
}

class GetableGenericInterface {
    public static void main(String[] args) {
        Box<Toy> box = new Box<>();
        box.set(new Toy());

        //System.out.println(box.get());

        Getable<Toy> gt = box;
        System.out.println(gt.get());
    }
}
```