NCS - Java 010 Generic

1. Generic(제네릭) 이란?

- java 에서 Generic 이라 함은 어떤 속성에 대한 정의를 하는 것이라고 할 수 있다.

- 사용이유

이유	
타입안정성	제네릭을 사용함으로써 단일 타입의 오브젝트만 저장할 수 있게 해준다
타입캐스팅 제거	저장된 오브젝트에 대한 타입캐스팅을 필요로 하지 않는다
컴파일시에 에러에 대한 선확인	타입캐스팅 등의 에러에 대한 예측이 가능해 지므로 실행하기 전 컴파일타임에 에러를 발생시킬 수 있다

2. J2SE 5 이전 버전에서의 object collection

- J2SE 5 이하에서는 ArrayList 라는 object collection 사용시에 아래와 같은 형태로 사용 되었다

```
import java.util.ArrayList;
import java.util.List;

public class Generic {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add("hello");
        String s = (String) list.get(0);
                    // 여기서 타입 캐스팅을 해주지 않으면
                    // collection 에 입력되었다가 다시 사용할 시에
                    // object 로 return 되기 때문에 오류가 발생한다
    }
}
```


3. J2SE 5 이후 버전에서의 처리

- J2SE 5 부터는 Generic 을 사용해서 클래스 타입을 선언시에 지정할 수 있다

```
import java.util.ArrayList;
import java.util.List;

public class Generic {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        list.add("hello");
        String s = list.get(0);
                    // 타입 캐스팅이 필요 없음
    }
}
```


4. 클래서에서의 Generic

- 클래스도 아래와 같이 Generic 형태로 정의 할 수 있다

```
public class Generic {
    public static void main(String[] args) {
        Person<String> p1 = new Person<String>();
        p1.info = "이영후";
        System.out.println(p1.info);
    }
}

class Person<T> {
    public T info;
}
```

Person 클래스에서 위와 같이 Generic으로 정의했으므로
다른 method 에서 사용시에 타입을 지정해 줄 수 있다
Person<T> 에서 T 는 Type을 의미한다

5. Generic 에서 Parameter 정의 방법

- Generic 에서 파라미터들을 정의하기 위해서 일반적으로 아래와 같은 약어를 사용한다

약어	파라미터 종류
T	Type
E	Element
K	Key
N	Number
V	Value

* 위와 같은 형태를 많이 사용하고 있으며, 절대적인 규칙은 아니지만 임의로 생성해서 사용하는것은 바람직 하지 않다

6. Method(함수)에서의 Generic

- 함수에서는 아래와 같이 Generic 형태로 정의 할 수 있다

```
public class generic {
    public static void main(String[] args) {
        Integer[] intArray = {10, 20, 30, 40};
        String[] stringArray = {"가", "나", "다", "라"};
        printArray(intArray);
        printArray(stringArray);
    }

    // 함수에서 사용되는 형태
    public static <E> void printArray(E[] elements) {
        for (E element : elements) {
            System.out.println(element);
        }
    }
}
```


7. Wildcard 와 상속을 이용한 Generic

- 와일드카드(?)로 제네릭을 선언함으로 여러가지 타입의 클래스를 입력 받아도 동일하게 동작하도록 할 수있다.
  단, extends로 상속을 받는 형태이므로 와일드 카드인 부모객체를 상속 받은 클래스만 가능하다

```
public class Generic {
    public static void drawShapes(List<? extends shape> lists) {
        for (shape s : lists) {
            s.draw();
        }
    }
}
```

* 와일드카드의 경우 타입을 직접 지정할 수는 없고 Generic 을 이용한 매개변수 형태로 지정되어야 한다
  예) Vector<? extends Integer>

8. ? Wildcard

- Generic 사용 시 타입 호환성을 위해 도입되었고 super를 사용하여 하위 호환성 문제를 해결할 수 있다

- 와일드 카드를 사용하기 위해서는 아래와 같은 제약사항이 있다

경계 지정 와일드카드는 타입 매개변수로만 사용할 수 있으며, 자신을 타입으로 사용할 수는 없다

와일드 카드 ? 로 받은 제네릭 요소 타입은 타 제네릭 요소 타입과 어떤 식으로든 결합될 때 컴파일러는 각 제네릭 요소 타입의 호환성을 알 수 없다

제네릭 요소 사이에는 타입 호환성(다형성)이 존재하지 않는다. 예를 들어 Integer 가 Number 와 같다고 해도 Class<Integer> 가 Class<Number> 는 아니다

