# 제네릭이란

## ▣ 제네릭이란

- 제너릭이란 결정되지 않은 타입을 파라미터로 처리하고,
  실제 사용할 때 파라미터를 구체적인 타입으로 대체시키는 기능이다.

- 타입 파라미터를 대체할수 있는 타입은 클래스 와 인터페이스만 가능하다. <예> Box<int> - X, Box<Integer> - O

- 변수를 선언할 때와 동일한 타입으로 호출하고 싶다면
  생성자 호출 시 생성자에는 타입을 명시하지 않고 <>만 붙일 수 있다.

```
public class Box <T> {
  public T content;
}
```

```
Box<String> box = new Box<String>();
```
→
```
Box<String> box = new Box<>();
```

```
Box<Integer> box = new Box<Integer>();
```
→
```
Box<Integer> box = new Box<>();
```

- **컴파일시 타입을 체크해 주는 기능**(compile-time type check) - JDK1.5
- **객체의 타입 안정성을 높이고 형변환의 번거로움을 줄여줌**
  **(하나의 컬렉션에는 대부분 한 종류의 객체만 저장)**

클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법을 의미. 인스턴스화할때 정의

각각의 인스턴스를 생성할 때 사용한 < > 사이에 어떤 데이터 타입을 사용하느냐에 따라 내부의 변수의 데이터 타입을 설정할 수 있다.

```
Box<String> box = new Box<String>();
box.set("hello");
String str = box.get();
```

```
public class Box<String> {
    private String t;
    public void set(String t) { this.t = t; }
    public String get() { return t; }
}
```

```
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
```

```
Box<Integer> box = new Box<Integer>();
box.set(6);
int value = box.get();
```

```
public class Box<Integer> {
    private Integer t;
    public void set(Integer t) { this.t = t; }
    public Integer get() { return t; }
}
```

| 타입인자 | 설명 |
|---|---|
| <T> | Type |
| <E> | Element |
| <K> | Key |
| <N> | Number |
| <V> | Value |
| <R> | Result |