Generics <> I

제네릭 클래스

- 자료형에 안전하고, 구현의 편의를 위한 클래스 설계 문법요소

제네릭 클래스의 등장배경

```
// 사과를 담을 수 있는 사과박스
class AppleBox {
     Apple item;

     public void store(Apple item){ this.item = item; }
     public Apple pullOut(){ return item; }
}

// 오렌지를 담을 수 있는 오렌지박스
class OrangeBox {
     Orange item;

     public void store(Orange item){ this.item = item; }
     public Orange pullOut(){ return item; }
}
```

- 장점 : 자료형에 안전하다.
- 단점 : 구현이 조금 번거롭다.

```
// 무엇이든 담을 수 있는 박스
class FruitBox {
     Object item;

     public void store(Object item){ this.item = item; }
     public Object pullOut(){ return item; }
}
```

- 장점 : 구현의 편의성이 좋다.
- 단점 : 자료형에 안전하지 못하다.

논리적 측면에서의 자료형의 안전성?

```
class DataTypeSafe {
     public static void main(String[] args) {
          FruitBox fBox1 = new FruitBox();
          fBox1.store(new Orange());

          Orange org1 = (Orange)fBox1.pullOut();
          org1.showSugarContent();

          FruitBox fBox2 = new FruitBox();

          fBox2.store("오렌지");
```

- 자료형에 안전하지 않다.
- 잘못된 자료형을 기반으로 인스턴스를 저장했음에도 불구하고 오류가 발생하지 않았기 때문에

```
          Orange org2 = (Orange)fBox2.pullOut();
```

- 설계 상 Orange 객체를 담아야 했는데, 사용자가 String 객체를 담았다.
- 그렇다고 해서 컴파일 오류는 발생하지 않는다. (문법적 오류가 아닌 논리적 오류)
- 추 후에 예외처럼 예상치 못한 논리적 오류가 발생할 수 있고, 아예 발견조차 되지 않을 수 있다.

```
<terminated> DataTypeSafe [Java Application] C:₩Program Files₩Java₩jre1.8.0_102₩bin₩javaw.exe
Exception in thread "main" 당도
java.lang.ClassCastException: java.lang.String cannot be cast to Orange
        at DataTypeSafe.main(DataTypeSafe.java:12)
```

```
        org2.showSugarContent();
    }
}
```

```
class DataTypeSafe {
    public static void main(String[] args) {
        OrangeBox fBox1 = new OrangeBox();
        fBox1.store(new Orange());

        Orange org1 = (Orange)fBox1.pullOut();
        org1.showSugarContent();

        OrangeBox fBox2 = new OrangeBox();

        fBox2.store("오렌지"); // ERROR
```

- 자료형에 안전하다.
- 잘못된 자료형을 기반으로 인스턴스를 저장하면 바로 컴파일 ERROR를 알려주기 때문에

```
        Orange org2 = (Orange)fBox2.pullOut();
        org2.showSugarContent();
    }
}
```

제네릭 클래스의 설계 <T>

- **자료형에 안전하고, 구현의 편의를 위한 클래스 설계 문법요소**

```
class FruitBox {
    Object item;

    public void store(Object item){ this.item = item; }
    public Object pullOut(){ return item; }
}
```

```
class FruitBox <T> {
    T item;

    public void store(T item){ this.item = item; }
    public T pullOut(){ return item; }
}
```

<T>

- **Class가 T 라는 이름 기반의 제네릭 클래스라는 것을 명시**
- **제네릭(Generic)은 클래스 설계시점에서 클래스 내부에서 사용할 자료형을 결정하지 않는 문법이다.**
- **외부에서 해당 Class의 인스턴스를 생성할 때, 자료형을 결정한다. (자료형에 안전!)**

```
public class Generics {
    public static void main(String[] args) {
        FruitBox<Orange> orBox = new FruitBox<Orange>();
        orBox.store(new Orange());
        Orange org = orBox.pullOut();
        org.showSugarContent();

        FruitBox<Apple> apBox = new FruitBox<Apple>();
        apBox.store(new Apple());
        Apple app = apBox.pullOut();
```

```
        app.showSugarContent();
    }
}
```

인스턴스 생성 시 자료형 결정하기

Class<T> name	=	new Class<T>();
제네릭 참조변수		제네릭 클래스의 인스턴스

- **FruitBox<Orange>** orBox = **new FruitBox<Orange>()**;
  <T>를 <Orange>로 대체해서 인스턴스를 생성하겠다.

```
class FruitBox <Orange> {
    Orange item;

    public void store(Orange item){ this.item = item; }
    public Orange pullOut(){ return item; }
}
```

- **FruitBox<Apple>** apBox = **new FruitBox<Apple>()**;
  <T>를 <Apple>로 대체해서 인스턴스를 생성하겠다.

```
class FruitBox <Apple> {
    Apple item;

    public void store(Apple item){ this.item = item; }
    public Apple pullOut(){ return item; }
}
```
