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

Apple 상자와 Orange 상자를

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

구현의 편의만 놓고 보면, FruitBox 클래스가 더 좋아 보인다. 하지만 FruitBox 클래스는 자료형에 안전하 지 못하다는 단점이 있다.

물론 AppleBox와 OrangeBox는 구현의 불편함이 따르는 단점이 있다. 그러나 자료형에 안전하다는 장점이 있다.

```
Class FruitBox 보았어든 담을 수 있는 상자 클래스 {
Object item;
public void store(Object item) { this.item=item; }
public Object pullOut() { return item; }
}
```

AppleBox, OrangeBox의 장점 인 자료형의 안전성과 FruitBox 의 장점인 구현의 편의성을 한데 모은것이 제네릭이다!

```
public static void main(String[] args)
{
    FruitBox fBox1=new FruitBox();
    fBox1.store(new Orange(10));
    Orange org1=(Orange)fBox1.pullOut();
    org1.showSugarContent();

    FruitBox fBox2=new FruitBox();
    fBox2.store("오렌지");
    Orange org2=(Orange)fBox2.pullOut();
    org2.showSugarContent(); 어머니 바다지점
}
```

실행중간에 Class Casting Exception이 발생한다! 그런데 실행중간에 발생하는 예외는 컴파일 과정에서 발견되는 오류상황보다 발견 및 정정이 어렵다! 즉, 이는 자료형에 안전한 형태가 아니다.

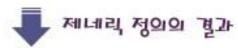
```
Public static void main(String[] args)
{
    OrangeBox fBox1=new OrangeBox();
    fBox1.store(new Orange(10));
    Orange org1=fBox1.pullOut();
    org1.showSugarContent();

    OrangeBox fBox2=new OrangeBox();
    fBox2.store("오렌지"); 에러 발생지점
    Orange org2=fBox2.pullOut();
    org2.showSugarContent();
}
```

컴파일 과정에서, 메소드 store에 문자열 인스턴스가 전달될 수 없음이 발견된다. 이렇듯 컴파일 과정에서 발견된 자료형 관련 문제는 발견 및 정정이 간단하다. 즉, 이는 자료형에 안전한 형태이다.

■ 제네릭 (Generics) 클래스 설계

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



```
T라는 이름이 매개변수화
된 자료형임을 나타냄
class FruitBox <T>
{
  T item;
  public void store( T item ) {this.item=item;}
  public T pullOut() {return item;}
```

T에 해당하는 자료형의 이름은 인스턴스를 생성하는 순간에 결정이 된다!

■ 제네릭 클래스 인스턴스 생성

FruitBox<Orange> orBox=new FruitBox<Orange>();

T를 Orange로 결정해서 FruitBox의 인스턴스를 생성하고 이를 참조할 수 있는 참조변수를 선언해서 참조 값을 저장한다!

FruitBox<Apple> apBox=new FruitBox<Apple>();

T를 Apple로 결정해서 FruitBox의 인스턴스를 생성하고 이를 참조할 수 있는 참조변수를 선언해서 참조 값을 저장한다!

```
public static void main(String[] args)
{
    FruitBox<Orange> orBox=new FruitBox<Orange>();
    orBox.store(new Orange(10));
    Orange org=orBox.pullOut();
    org.showSugarContent();

    FruitBox<Apple> apBox=new FruitBox<Apple>();
    apBox.store(new Apple(20));
    Apple app=apBox.pullOut();
    app.showAppleWeight();
}
```

인스턴스 생성시 결정된 T의 자료형에 일치하지 않으면 컴파일 에러가 발생 하므로 자료형에 안전한 구조임!

■ 제네릭을 이용한 List 생성 (1 / 2)

```
public class MyList<E> {
   private int size = 0;
   private Object[] data = new Object[10];
   public void add(E e) {
      if(size >= data.length) {
         data = Arrays.copyOf(data, data.length + 10);
      this.data[size] = e;
      size++;
   E get(int idx) {
      return (E) this.data[idx];
```

■ 제네릭을 이용한 List 생성 (2 / 2)

```
public class MyListMain {
   public static void main(String[] args) {
      MyList < String > list = new MyList < String > ();
      list.add("1");
                         list.add("2");
      list.add("3");
                         list.add("4");
      list.add("5");
                         list.add("6");
      list.add("7");
                         list.add("8");
      list.add("9");
                         list.add("10");
      System.out.println(list);
      list.add("11");
      System.out.println(list);
```

■ 타입 매개변수 표기

亜 기	설명
E	Element
K	Key
N	Number
Т	Туре
V	Value
•••	