제18장
제네릭(Generic)-2

5. 제네릭 메서드

▣ 매개변수 타입과 리턴 타입으로 타입 파라메터를 갖는 메서드를 칭한다.

```
public <T> Student<T> changing(T t)
```

※리턴 타입 : Student<T>, 매개변수 타입 : T타입

```
public static<T> Student<T> changing(T t)
```

▣ **제네릭 메서드는 리턴 타입 앞에 꺽쇠 기호를 추가하고, 타입 파라메터를 기술하며, 타입 파라메터를 리턴 타입과 매개변수에 사용한다.**

▣ 제네릭 메서드를 호출하는 두 가지 방법

```
리턴타입 변수 = <구체적타입> 메소드명(매개값);      //명시적으로 구체적 타입 지정
리턴타입 변수 = 메소드명(매개값);                  //매개값을 보고 구체적 타입을 추정
```

```
Box<Integer> box = <Integer>boxing(100);    //타입 파라미터를 명시적으로 Integer 로 지정
Box<Integer> box = boxing(100);             //타입 파라미터를 Integer 으로 추정
```

* 컴파일러에 의해 매개 변수 타입에 의해 타입 파라메터의 타입이 유추 된다.

6. 타입 파라메터의 제한

▣ 메서드의 타입 파라메터를 구체적으로 타입을 제한하고자 할 때 사용한다.

– 상속이나 구현 관계를 이용하여 타입 파라메터를 제한할 수 있다.

```
public static<T extends Number> Student<T> changing(T t)
```

– 상위 타입은 클래스 뿐 아니라 인터페이스도 가능하지만, 인터페이스라고 하여 implements를 사용하지 않고 동일하게 extends를 사용한다.

– 위와 같이 extends로 메서드를 선언했다면, 메서드의 매개값으로 상위타입은 제한된다.

```
public <T extends Number> int compare(T t1, T t2) {

    double v1 = t1.doubleValue();    //Number클래스의 doubleValue()추상 메서드를
                                     //오버라이딩한 T타입의 doubleValue()가 호출됨.
    double v2 = t2.doubleValue();    //위와 동일
    return Double.compare(v1, v2)
}
```

7. 와일드 카드 타입

- 이미 선언되어 있는 제네릭 타입을 매개변수나 리턴 타입으로 사용할 때,
  타입 파라메터를 제한할 목적으로 사용한다.
  ★ [비교]
  – 〈T extends 상위 또는 인터페이스〉는 제네릭 타입과 메서드를 선언할 때 제한함

```
public static void registerCourse(Course<?> course) {}
public static void registerCourse(Course<? extends Student> course) {}
public static void registerCourse(Course<? super Student> course) {}
```

- 와일드 카드 타입의 세가지 형태(중요함)

  - 제네릭타입<?> : Unbounded Wildcards (제한없음) **A,B,C,D,E 다 허용**
    타입 파라미터를 대치하는 구체적인 타입으로 모든 클래스나 인터페이스 타입이 올 수 있다.
  - 제네릭타입<? extends 상위타입> : Upper Bounded Wildcards (상위 클래스 제한) **C,D,E만 허용**
    타입 파라미터를 대치하는 구체적인 타입으로 상위 타입이나 하위 타입만 올 수 있다.
  - 제네릭타입<? super 하위타입> : Lower Bounded Wildcards (하위 클래스 제한) **A,B,C만 허용**
    타입 파라미터를 대치하는 구체적인 타입으로 하위 타입이나 상위 타입이 올 수 있다.

8. 제네릭 타입의 상속과 구현

- 제네릭 타입을 조상클래스로 사용할 경우에, 자손클래스에도 반드시 타입 파라메터를 기술해야 한다.
  - 다시 말해, 조상이 제네릭이면 자손도 제네릭이 된다는 것!

```
class Student<T,M> extends Person<T, M> { }
```

  - 또한, 얼마든지 추가적 타입 파라메터를 가질수 있다.

```
class Student<T,M,C> extends Person<T, M> { }
```

- 제네릭 타입의 인터페이스를 구현할 경우에도 역시 타입파라메터를 **구현클래스에서도 반드시 기술해야 한다.(인터페이스도 일종의 조상**이다.)

```
class Student<T> implements Comparable<T>
```

감사합니다.