JAVA 기초입문과정

CHAPTER12 제네릭

Contents

제네릭


PART
01
“
제네릭
”

PART 01

제네릭

포괄적인;
타입을 정하지 않고 클래스 등의 설계를 한다.

1. 제네릭이란?

결정되지 않은 타입을 파라메터로 처리하고
실제 사용할 때 파라메터를 구체적인 타입으로 대체시키는 기능

1.

```
public class Box {
        public ? content
}
```

- Box는 다양한 내용물을 저장해야 한다.

2.

```
public class Box {
        public Object content
}
```

- 특정 클래스 타입을 선언할 수 가 없어서 Object로 선언.

3.

```
Box box = new Box();
box.content = 모든 객체;
```

- content에는 어떤 객체라도 대입 가능하다.

4.

```
String content = (String) box.content;
```

- 강제 타입 변환으로 내용물을 얻을 수 있다.
- 하지만 어떤 내용물이 저장되어 있는지
  모른다면 모든 클래스를 조사하는 건 한계가 있다.

Box를 생성할 때 저장할 내용물의 타입을 미리 알려주면 content에 무엇이 대입되고, 읽을 때 어떤 타입으로 제공할지를 안다.

2. 제네릭 정의

T라는 타입은 없다.
A ~ Z까지 어떤 것이든 사용 가능하다.

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

*T: 타입 파라메터, 타입이 필요한 자리에 T를 사용할 수 있다.

```
Box<String> box = new Box<String>();
box.content = “안녕하세요”;
String content = box.content;
```

- 강제 타입 변환이 필요 없이 바로 ‘안녕하세요’를 얻을 수 있다.

```
Box<Integer> box = new Box<Integer>();
box.content = 100;
int content = box.content;
```

Box(int) 라고 하지 않는 이유:
타입파라미터를 대체하는 타입은 클래스나 인터페이스다.

2. 제네릭 정의

생략해서 쓸 수도 있다.

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

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

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

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

3. 제네릭 타입 결정되지 않은 타입을 파라미터로 가지는 클래스와 인터페이스

```
public class 클래스명<A, B, ...> { ... }

public interface 클래스명<A, B, ...> { ... }
```

지정하지 않으면
암묵적으로 Object가 된다.

- 외부에서 제네릭 타입을 사용하려면 타입 파라메터에 구체적인 타입을 지정해야 한다.

4. 제네릭 메소드 결정되지 않은 타입을 파라미터로 가지는 메소드

```
public <A, B, ... > 리턴타입 메소드명(매개변수, ... ) { ... }
```

- 리턴타입과 매개변수에서 사용한다.

리턴 타입, 메소드를 동시에
적용하므로
두 개의 타입은
일치가 되어야 한다.

```
public <T> Box<T> boxing(T t) { ··· }

① Box<Integer> box1 = boxing(100);
② Box<String> box2 = boxing("안녕하세요");
```

5. 제한된 타입 파라메터 특정 타입과 자식 관계에 있는 타입만 대체할 수 있는 타입 파라미터

```
public <T extends 상위타입> 리턴타입 메소드(매개변수, ...) { ... }
```

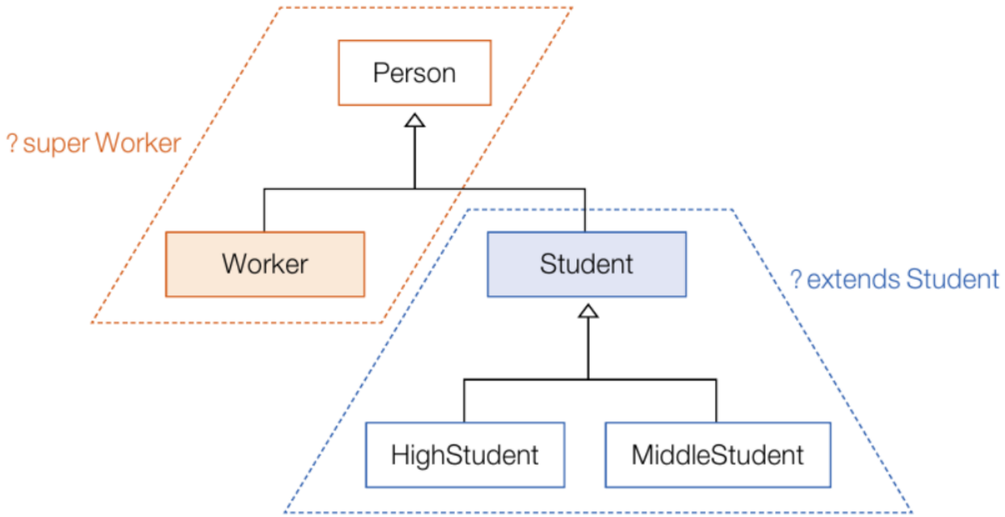
상위타입과 같거나 상위 타입의 자식 객체여야 한다.

타입제한으로 Number가 가지고 있는 메소드도 사용 가능.

```
public <T extends Number> boolean compare(T t1, T t2) {
  double v1 = t1.doubleValue();   //Number의 doubleValue() 메소드 사용
  double v2 = t2.doubleValue();   //Number의 doubleValue() 메소드 사용
  return (v1 == v2);
}
```

6. 와일드 카드 타입 파라메터 ?는 범위에 있는 모든 타입으로 대체할 수 있다.

```
리턴타입 메소드명(제네릭타입<?> 변수) { ... }
```



- Student의 자식클래스와 자기 자신만 사용 가능

```
리턴타입 메소드명(제네릭타입<? extends Student> 변수) { … }
```

- Worker의 부모클래스와 자기 자신만 사용 가능

```
리턴타입 메소드명(제네릭타입<? super Worker> 변수) { … }
```

- 어떤 타입이든 가능

```
리턴타입 메소드명(제네릭타입<?> 변수) { … }
```

Thank You!