제네릭

주효진

CONTENTS

Chapter 01 왜 제네릭을 사용해야 하는가?

Chapter 02 제네릭 타입

Chapter 03 멀티 타입 파라미터

Chapter 04 제네릭 메소드

Chapter 05 제한된 타입 파라미터

Chapter 06 와일드카드 타입

Chapter 07 제네릭 타이브이 상속과 구현

Chapter 01

왜 제네릭을 사용해야 하는가?

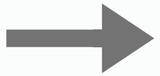
왜 제네릭을 사용해야 하는가?

- 제네릭은 Java 5에서 부터 추가 되었고 제네릭 타입을 이용함으로써 잘못된 타입이 사용될 수 있는 문제를 컴파일 과정에서 제거할 수 있게 되었다.
  제네릭의 중요성 : 컬렉션, 람다식, 스트림, NIO에서 널리 사용됨, API 도큐먼트를 보면 제네릭 표현이 많아 API 도큐먼트를 정확히 이해할 수 없음

- 제네릭의 장점
  1. 컴파일 시 강한 타입 체크를 할 수 있음
     - 실행 시 타입 에러가 나는 것을 방지
     - 컴파일 시 미리 타입을 강하게 체크해 에러를 사전 방지

  2. 타입 변환(casting)을 제거

```
List list = new ArrayList();
list.add("hello");
String str = (String) list.get(0);
```



```
List<String> list = new ArrayList<String>();
list.add("hello");
String str = list.get(0);
```

Chapter 02

제네릭 타입
class<T>
interfase<T>

제네릭 타입

- 제네릭 타입 : 타입을 파라미터로 가지는 클래스와 인터페이스
  클래스 또는 인터페이스 이름 뒤에 "<>" 부호가 붙고, 사이에 타입 파라미터가 위치
  ex)
  public class 클래스명<T> { ... }
  public interfase 인터페이스명<T> { ... }

- 제네릭 타입 사용 비교
  Object 타입을 사용하면 빈번한 타입 변환이 발생하여 프로그램 성능 저하

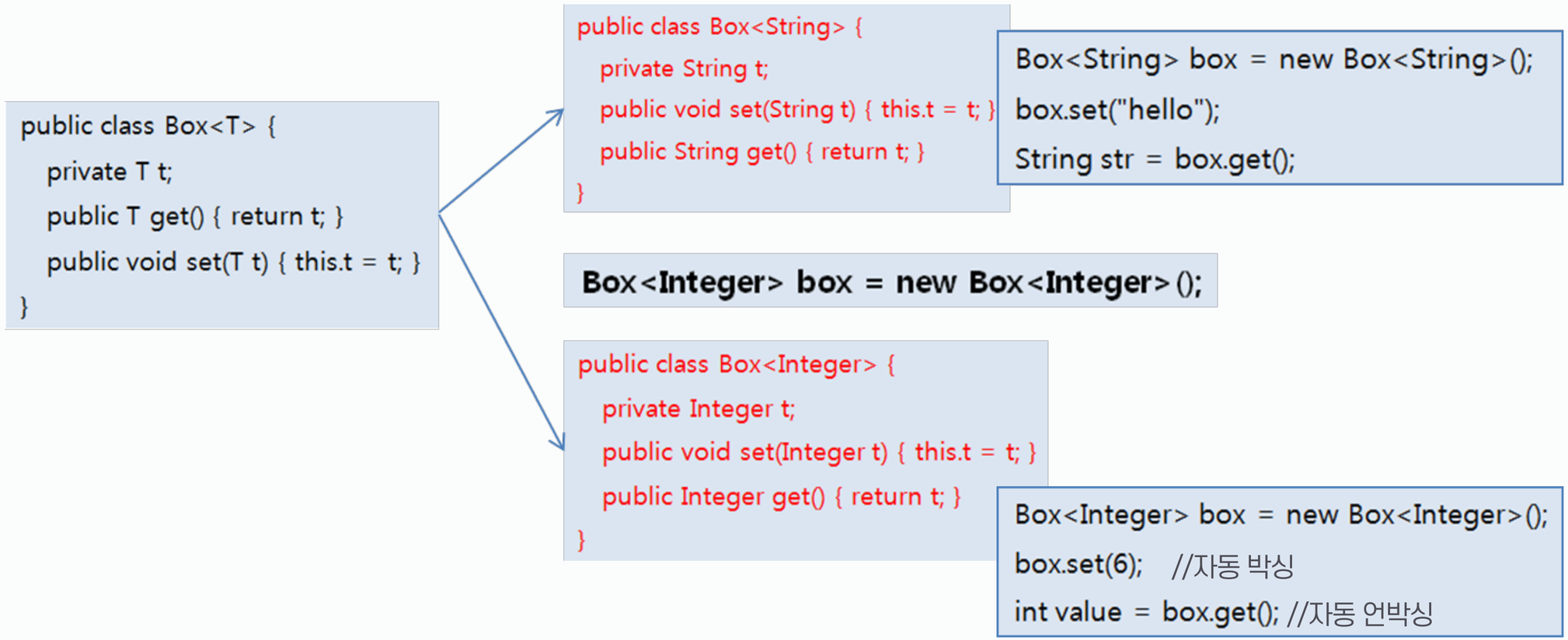
```
public class Box {
   private Object object;
   public void set(Object object) { this.object = object; }
   public Object get() { return object; }
}
```

```
Box box = new Box();
box.set("hello");                  //String 타입을 Object 타입으로 자동 타입 변환해서 저장
String str = (String) box.get();   //Object 타입을 String 타입으로 강제 타입 변환해서 얻음
```

제네릭 타입

- 제네릭 타입 사용 비교

  Box<String> box = new Box<String>();을 실행하여 Box 객체를 생성하면 타입 파라미터 T가 String으로 변경되어 재구성 됨



Chapter 03

멀티 타입 파라미터

class<K, V, ...>

interfase<K, V, ...>

멀티 타입 파라미터

- 제네릭 타입은 두 개 이상의 멀티 타입 파라미터를 사용할 수 있음
  각 파라미터는 콤마로 구분

```
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() { return this.kind; }
    public M getModel() { return this.model; }

    public void setKind(T kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}
```

```
Product<Tv, String> product = new Product<Tv, String>();
```

```
Product<Tv, String> product = new Product<>();
// 자바 7부터 다이아몬트 연산자 (<>)를 사용해서 간단히 작성 가능
```

Chapter 04

제네릭 메소드

<T,R> R method(T t)

제네릭 메소드

- 매개 타입과 리턴 타입으로 타입 파라미터를 갖는 메소드
  선언하는 방법은 리턴 타입 앞에 <> 기호를 추가하고 타입 파라미터를 기술한 후 리턴 타입과 매개 타입으로 타입 파라미터를 사용

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

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

1. <> 기호 안에 타입 파라미터 T를 기술
2. 리턴 타입을 제네릭 타입인 Box<T> 사용
3. boxing 메소드의 매개변수 타입으로 T를 사용

- 제네릭 메소드 호출
  타입을 지정해주거나 컴파일러가 매개 값을 보고 타입을 추정하는 방법이 있음

```
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
```

Chapter 05

제한된 타입 파라미터
<T extends 최상위타입>

제한된 타입 파라미터

- 타입 파라미터에 지정되는 구체적인 타입을 제한할 필요가 있는 경우 제한된 타입 파라미터가 필요함
  ex) 숫자 연산을 하는 제네릭 메소드의 매개 값으로 Number 타입 또는 하위 클래스 타입(Byte, Short, Integer, Long, Double)이 필요한 경우
  타입 파라미터 뒤에 extends 키워드를 붙여 상위 타입을 명시, 인터페이스도 extends 키워드를 사용(implements X)

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

Chapter 06

와일드카드 타입

<?>

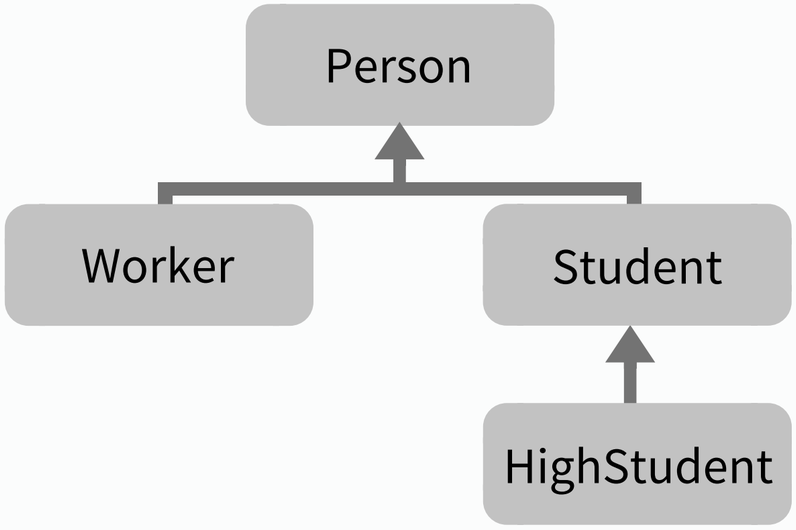
<? extends ...>

<? super ...>

와일드카드 타입

- 코드에서 ?를 일반적으로 와일드카드라 함.
  제네릭 타입을 매개값이나 리턴 타입으로 사용할 때 구체적인 타입 대신 와일드카드를 다음과 같이 사용

  - 제네릭타입<?> : Unbounded Wildcards (제한 없음), 타입 파라미터를 대치하는 구체적인 타입으로 모든 클래스나 인터페이스가 올 수 있음
  - 제네릭타입<? extends 상위타입> : Upper Bounded Wildcards (상위 클래스 제한), 상위타입이나 하위타입만 올 수 있음
  - 제네릭타입<? super 하위타입> : Lower Bounded Wildcards (하위 클래스 제한), 하위타입이나 상위타입이 올 수 있음



Course<T>

- Course<?>
  -> Person, Worker, Student, HighStudent
- Course<? extneds Student
  >-> Student, HighStudent
- Course<? super Worker>
  -> Person, Worker

Chapter 07

제네릭 타입의 상속과 구현

제네릭 타입의 상속과 구현

- 제네릭 타입도 다른 타입과 마찬가지로 부모 클래스가 될 수 있음
  Product<T, M> 제네릭 타입을 상속한 ChildProduct<T, M> 타입을 정의, 추가적으로 타입 파라미터를 가질 수 있음

```
public class ChildProduct<T, M> extends Product<T, M> { ... }
```

```
public class ChildProduct<T, M, C> extends Product<T, M> { ... }
```

감사합니다.