KH 정보교육원

제네릭스
(Generics)

제네릭스(Generics)

KH 정보교육원

제네릭스란?

jdk 1.5 부터 제공되는 기능으로, 컬렉션 클래스를 이용해서 객체를 저장할 때, 저장할 객체(클래스타입)을 제한하는 기능으로, 한 가지 종류의 클래스만 저장할 수 있게 해 놓은 기능이다.

제네릭스를 사용하는 이유

- '컴파일 단계 ' 에서 '잘못된 타입을 사용할 수 있는 가능성' 제거함
- 컬렉션에 저장된 여러 종류의 객체를 꺼내서 사용할 때, 객체의 종류에 따라 매번 형변환하는 복잡한 코드를 제거함.
- 컬렉션, 람다식(함수적 인터페이스), 스트림, NIO에서 널리 사용한다.
- 제네릭스를 모르면 API Document 해석이 어렵기 때문에 학습에 제한

```
public class ArrayList<E> extends AbstractList<E>
```

제네릭스(Generics)

KH정보교육원

제네릭스의 이점

· 컴파일 시 강한 타입 체크가 가능하다. (실행시, 컴파일시 에러 방지)

· Object타입의 요소를 지정타입으로 형변환할 필요가 없다.

```
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);
```

제네릭스(Generics)

KH 정보교육원

제네릭 타입

클래스와 인터페이스 선언시 클래스 또는 인터페이스 이름 뒤에 “< >”괄호안에 타입 파라미터로써 특정클래스를 기입한다.

표현식

클래스명<클래스타입> 레퍼런스 = new 생성자<클래스타입>();

사용예시

ArrayList<Book> list = new ArrayList<Book>();

제네릭스(Generics)

KH 정보교육원

함수에서 제네릭스 사용

제네릭스가 설정된 레퍼런스를 인자로 넘기는 경우

예) ArrayList<Book> list = new ArrayList<Book>();
BookManager bm = new BookManager();
bm.printInformation(list);

```
//BookManager Class
public void printInformation(ArrayList<Book> list){
    ....
}
```

메소드 쪽에서 받아주는 매개 변수도 제네릭스가 적용되어야 한다.

제네릭스(Generics)

KH 정보교육원

함수에서 제네릭스 사용

제네릭스가 적용된 레퍼런스를 리턴하는 경우

```
예) public ArrayList<Book> getInformation(){
        ArrayList<Book> list = ArrayList<Book>();
        return list;
    }
```

메소드의 반환형에도 제네릭스가 적용되어야 한다.

제네릭스(Generics)

KH 정보교육원

클래스에서 제네릭스 사용

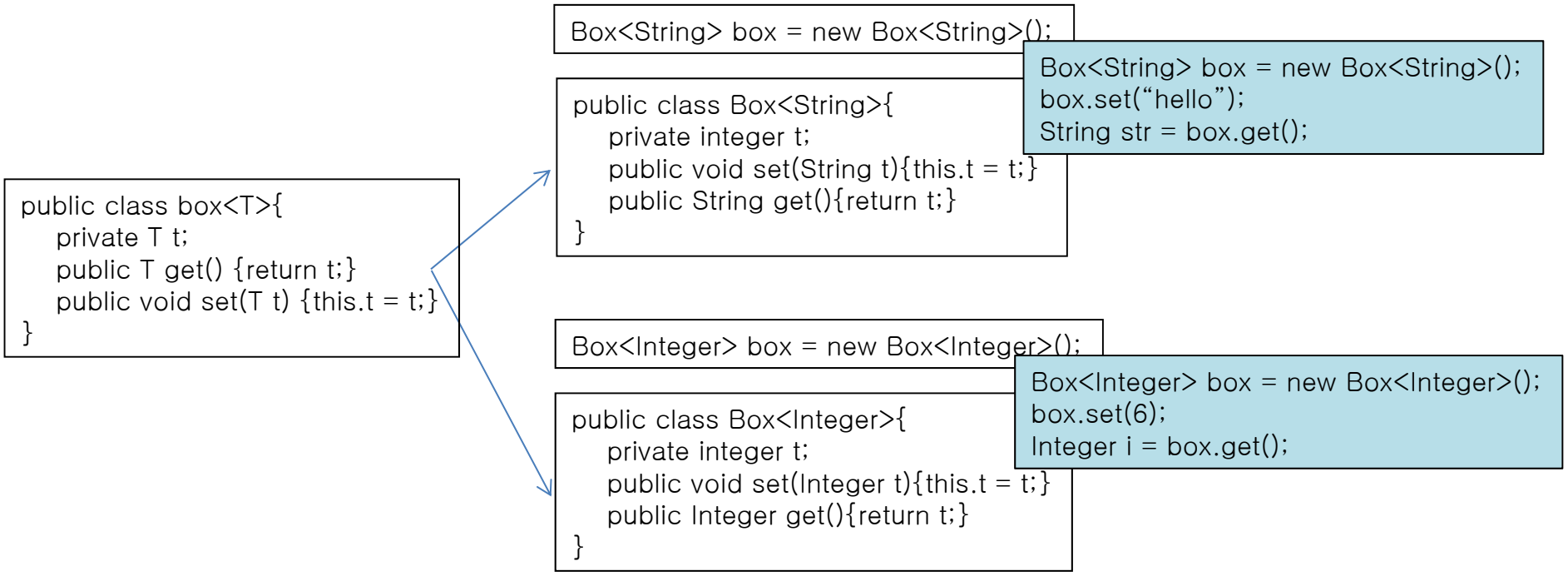
예) class 클래스명<영문자>{
        영문자 레퍼런스;
    }

영문자 : 일반적으로 대문자를 사용한다.
        클래스 타입이 미정인 경우, 객체 생성시 정해지는 클래스
        타입을 영문자가 받아서, 영문자 사용 위치에 적용한다.

제네릭스(Generics)

KH정보교육원

클래스에서 제네릭스 사용



객체생성시 타입 파라미터가 구체적인 클래스로 변경된다.

제네릭스(Generics)

KH 정보교육원

멀티 타입 파라미터

제네릭 타입은 두 개 이상의 타입 파라미터를 사용 가능하다. 각 타입 파라미터는 콤마로 구분하여 사용하면 된다.

```
public class Employee<D, P>{
    private D dept;
    private P person;

    public D getDept(){return this.dept;}
    public P getPerson(){return this.person;}

    public void setDept(D dept){this.kind = kind;}
    public void setPerson(P person){this.person = person;}
}
```

```
Employee<Dept, Person> = new Employee<Dept, Person>();
Employee<Dept, Person> = new Employee<>();
```

제네릭스(Generics)

KH 정보교육원

제네릭스 타입의 상속과 구현

제네릭스 타입을 부모 클래스로 사용해야 할 경우에는, 타입 파라미터는 자식 클래스에도 기술해야 하며, 추가적인 타입 파라미터를 가질 수 있다.

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

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

제네릭스(Generics)

KH 정보교육원

구체적 타입 제한

상속 및 구현관계를 이용해 타입을 제한할 수 있으며, 상위 타입으로 클래스, 인터페이스 모두 가능하다.

```
class 클래스명<T extend 상위타입>{}
```

이제 클래스에 전달할 수 있는 타입은 상위타입을 상속한 자식클래스만 가능하게 된다.

단, 중괄호 {}안에서 타입 파라미터 변수로 사용 가능한 것은 상위타입의 멤버(필드, 메소드)로 제한되어, 하위 타입에만 있는 필드와 메소드는 사용할 수 없다.

제네릭스(Generics)

KH 정보교육원

와일드카드를 이용한 상/하한 제한.

와일드카드 ? 는 extends와 super 키워드와 함께 타입의 상한제한(지정클래스를 상속한 자식클래스)와 하한제한(지정클래스의 부모클래스)을 가능하게 한다.

단독으로 쓰인 ?와일드 카드는 Object와 같다.

```
<? extends T> //와일드카드의 상한 제한. T와 그 자손들만 가능
<? super T> //와일드카등의 하한제한. T와 그 조상들만 가능
<?> //제한 없음. 모든 타입이 가능. <? extends Object>와 동일
```