제네릭스
(Generics)

▶ 제네릭스(Generics)

✓ 제네릭스

JDK1.5부터 제공되는 기능
클래스나 메소드 내부에서 다룰 데이터의 클래스 타입을 지정하거나,
컬렉션이 다룰 클래스 타입을 제한하여 한 가지 종류의 클래스만 저장할 수 있게 한 기능

✓ 제네릭스를 사용하는 이유

- 컴파일 단계에서 '잘 못 된 타입을 사용할 수 있는 문제' 제거
- 컬렉션에 저장된 여러 종류의 객체를 꺼내서 사용할 때,
  객체의 종류에 따라 매번 형 변환을 해야하기 때문에 코드가 복잡해짐
- 컬렉션, 람다식(함수적 인터페이스), 스트림, NIO에서 널리 사용
- API Document 해석에 어려움을 겪어 학습에 제한

```
default BiConsumer<T,U> andThen(BiConsumer<? super T,? super U> after)
```

▶ 제네릭스(Generics)

✓ 제네릭스 이점

- 컴파일 시 강한 타입 체크 가능(실행 시, 컴파일 시 에러 방지)
- 타입 변환 제거 가능

```
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)

✓ 제네릭스 타입과 표현식

타입을 파라미터로 가지는 클래스와 인터페이스 선언 시 클래스 또는 인터페이스 이름 뒤에 "< >" 부호를 붙이고 사이에는 타입 파라미터 위치

[표현식]

클래스명<**클래스타입**> 레퍼런스 = new 생성자<**클래스타입**>();

클래스명<**클래스타입**> 레퍼런스 = new 생성자<>(); // JDK 1.7 부터 적용, 타입 추론

[사용예시]

```
ArrayList<Book> list1 = new ArrayList<Book>();
ArrayList<Book> list2 = new ArrayList<>();
```

▶ 제네릭스(Generics)

✓ 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)

✓ 제네릭스가 설정된 레퍼런스를 리턴하는 경우

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

메소드의 반환형에도 제네릭스가 적용되어야 한다.

▶ 제네릭스(Generics)

✓ 클래스에서 제네릭스 사용

예) class 클래스명**<영문자>** {

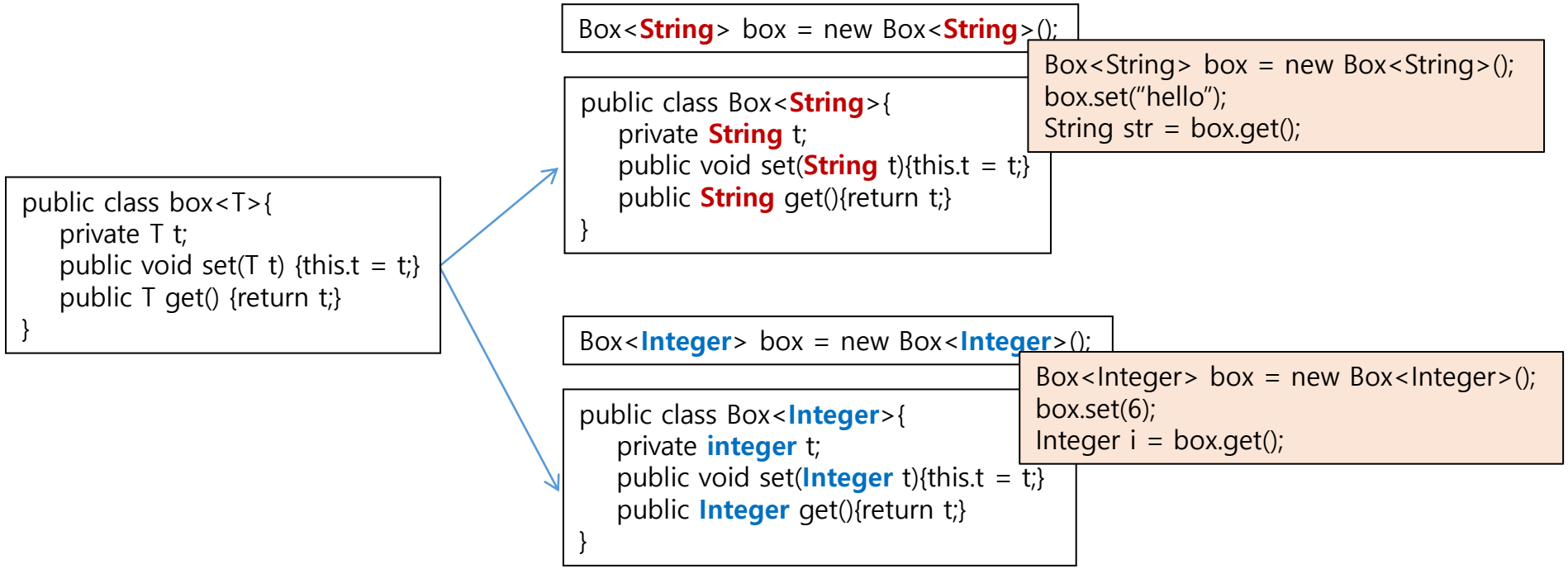
    영문자 레퍼런스;

}

* 영문자
  - 일반적으로 대문자 사용(ex. T, E, K, V)
  - 클래스 타입이 미정인 경우, 객체 생성 시 정해지는 클래스 타입을 영문자가 받아 영문자 사용 위치에 적용

▶ 제네릭스(Generics)

✓ 클래스에서 제네릭스 사용



컴파일 시 타입 파라미터가 구체적인 클래스로 변경된다.

▶ 제네릭스(Generics)

✓ 멀티 타입 파라미터

제네릭스 타입은 두 개 이상의 타입 파라미터 사용 가능하며 각 타입 파라미터를 콤마로 구분

```
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<>();  // 타입 추론
```

▶ 제네릭스(Generics)

✓ 제네릭스 타입의 상속과 구현

제네릭스 타입을 부모 클래스로 사용해야 할 경우 타입 파라미터는 자식 클래스에도 기술

추가적인 타입 파라미터를 가질 수 있음

ex)

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

▶ 제네릭스(Generics)

✓ 구체적인 타입 제한

상속 및 구현 관계를 이용해 타입 제한이 가능하며,
상위 타입은 클래스 뿐 아니라 인터페이스도 가능

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

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