객체지향 프로그래밍

Object-oriented Programming

Chapter 09.
예외 처리와 제네릭 프로그래밍

- 예외
- 예외 처리 방법
- 제네릭 클래스와 인터페이스
- 제네릭 상속과 타입 한정
- 제네릭 메서드

제네릭 클래스와 인터페이스

제네릭 타입의 필요성

- 자바는 다양한 종류의 객체를 관리하는 컬렉션(Collection)이라는 자료구조를 제공
- 초기에는 Object 타입의 컬렉션을 사용
- 모든 타입의 객체가 컬렉션 자료구조에 저장될 수 있었음

제네릭 클래스와 인터페이스

제네릭 타입

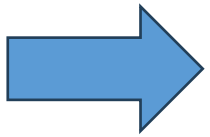
- 하나의 코드를 다양한 타입의 객체에 재사용하는 객체 지향 기법
- 클래스, 인터페이스, 메서드를 정의할 때 **타입을 변수로 사용**

제네릭 클래스와 인터페이스

제네릭 타입의 장점

- 컴파일할 때 타입을 점검하기 때문에 실행 도중 발생할 오류 사전 방지
- 불필요한 타입 변환이 없어 프로그램 성능 향상
- 코드 재사용성 및 가독성 향상

```
List intList;
List stringList;
```



```
List<Integer> intList;
List<String> stringList;
```

제네릭 클래스와 인터페이스

제네릭 타입 선언

- 클래스, 인터페이스, 메서드 선언부에 <>기호를 사용해 타입 변수를 선언
- 여러 개의 타입 변수를 선언할 때 ', (comma)'로 각 이름을 구분

```
class 클래스이름<타입매개변수> {
    필드;
    메서드;
}
```

메서드나 필드에 필요한 타입을 타입 매개변수로 나타낸다.

```
public class Box<T, P> {
}
```

제네릭 클래스와 인터페이스

제네릭 객체 생성

- <> 기호 내에 해당 클래스에 선언된 타입 매개변수에 대응되는 타입 이름을 표기

```
제네릭클래스 <적용할타입> 변수 = new 제네릭클래스<적용할타입>();
```

생략할 수 있다.

```
Box<Integer, String> bos = new Box<Integer, String>();
```

Practice

제네릭 클래스 실습

- 예제 9-14: 제네릭 클래스 (p.368)
- 예제 9-15: 제네릭 클래스의 테스트 (p.369)
- 예제 9-17, 9-18: 다중 타입 매개변수 제네릭 (p.370)

제네릭 상속 및 타입 한정

제네릭 타입의 상속 관계

- 자식 객체를 부모 타입 변수에 저장할 수 있음
- 제네릭 객체를 다룰 때도 타입 매개변수의 자식 타입을 사용 가능

```
ArrayList<Beverage> list = new ArrayList<>();
list.add(new Beer());                    // OK
list.add(new Boricha());                 // OK
```

제네릭 상속 및 타입 한정

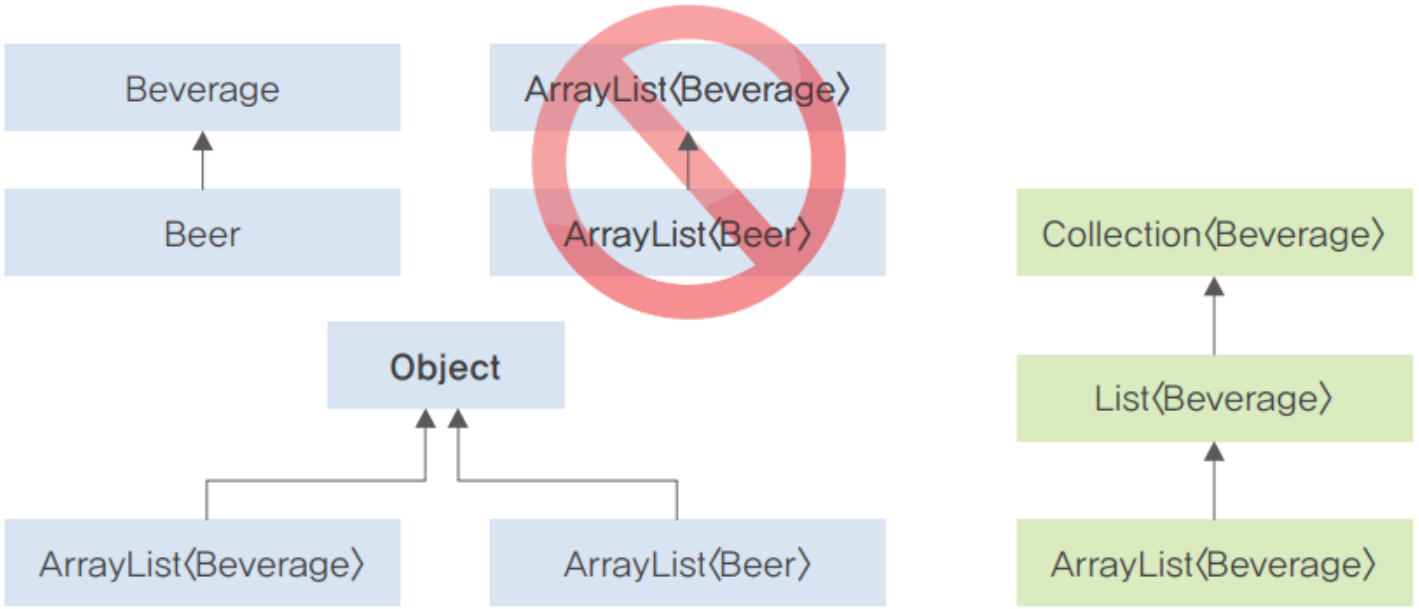
제네릭에 관한 제약 사항

- 실행 중에 제네릭 타입 점검 금지. Ex) a instanceof ArrayList<String>
- 기초 타입을 제네릭 인수로 사용 불가
- 정적 제네릭 타입 금지
- 제네릭 타입의 인스턴스화 금지. Ex) new T( )
- 제네릭 타입의 배열 생성 금지
- 제네릭 클래스의 객체는 예외로 던지거나 잡을 수 없다
- 서브타입 제약

제네릭 상속 및 타입 한정

서브타입 제약

- **타입 매개변수 간 상속 관계**와 **제네릭 객체 간 상속 관계**는 별도로 취급



제네릭 상속 및 타입 한정

타입 한정

- 타입 매개변수의 범위를 **extends** 키워드를 사용해 한정지을 수 있음

- 특정 타입의 하위 타입으로 한정:

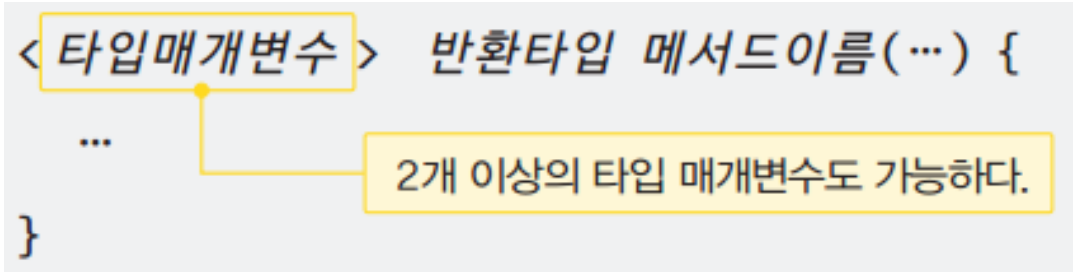
```
<T extends 특정클래스> 반환타입 메서드이름(...) { ... }
<T extends 인터페이스> 반환타입 메서드이름(...) { ... }
```

부모가 인터페이스라도 extends를 사용한다.

제네릭 메서드

제네릭 메서드

- **제네릭 메서드**: 타입 매개변수를 사용하는 메서드
- 제네릭 메서드를 정의할 때는 타입 매개변수를 반환 타입 앞에 위치



제네릭 메서드

제네릭 메서드의 타입 한정

- 제네릭 클래스, 인터페이스의 타입 한정 방식과 동일

```
<T extends 특정클래스> 반환타입 메서드이름(…) { … }
<T extends 인터페이스> 반환타입 메서드이름(…) { … }
```

부모가 인터페이스라도 extends를 사용한다.

Practice

제네릭 메서드 실습

- 예제 9-21: 제네릭 메서드의 테스트 (p.374)
- 예제 9-22, 9-23: 제한된 타입의 제네릭 메서드 (p.375)

Practice

제네릭 실습

요구사항:

- 주어진 실행 결과를 참고해 Box 클래스를 작성하라.
- Box 클래스는 어떤 객체라도 보관할 수 있고 꼬집어낼 수 있어야 한다.

```
Box<Integer> i = new Box<>();
i.set(new Integer(100));
System.out.println(i.get()); // 결과: 100

Box<String> s = new Box<>();
s.set("만능이네!");
System.out.println(s.get()); // 결과: 만능이네!
```

Practice

제네릭 실습

요구사항:

- 주어진 실행 결과를 참고해 Max 클래스를 작성하라.
- Max 클래스는 **제네릭 클래스**로, 인수가 **숫자라면** 더 큰 숫자를 반환하고,
  숫자가 아니라면 더 긴 **문자열**을 반환하는 max() 메서드를 포함해야 한다.

제약 조건:

- Max 클래스의 타입 매개변수는 **숫자(Number) 혹은 문자열(String)**이라고 가정
- 객체를 비교할 때 <와 같은 비교 연산자는 사용 불가
- Number 객체에서 정수 값이나 실수 값을 얻으려면 intValue()나 doubleValue() 메서드를 호출

```
Max<Number> n = new Max<>();
System.out.println(n.max(10.0, 8.0)); // 결과: 10.0
System.out.println(n.max(5, 8.0));     // 결과: 8.0

Max<String> s = new Max<>();
System.out.println(s.max("Hello", "Hi"));     // 결과: Hello
System.out.println(s.max("Good", "morning")); // 결과: morning
```