2017 September 28

SeokRae Kim

[회사 이름]

[회사 주소]

자바의 정석

chapter 12 제네릭, 열거형, 어노테이션

내용

[I. 제네릭 1](#_Toc495745077)

[1. 제네릭스(Generics)란? 1](#_Toc495745078)

[2. 제네릭 클래스의 선언 1](#_Toc495745079)

[1) 제네릭스의 용어 1](#_Toc495745080)

[2) 제네릭스의 제한 2](#_Toc495745081)

[3. 제네릭 클래스의 객체 생성과 사용 3](#_Toc495745082)

[4. 제한된 제네릭 클래스 4](#_Toc495745083)

[5. 와일드 카드 5](#_Toc495745084)

[6. 제네릭 메서드 \*\* 8](#_Toc495745085)

[7. 제네릭 타입의 형변환 8](#_Toc495745086)

[8. 제네릭 타입의 제거 8](#_Toc495745087)

[**II.** **열거형(enums)** 8](#_Toc495745088)

[**1.** **열거형이란?** 8](#_Toc495745089)

[**2.** **열거형의 정의와 사용** 8](#_Toc495745090)

[1) 모든 열거형의 조상 – java.lang.Enum 8](#_Toc495745091)

[**3.** **열거형에 멤버 추가하기** 10](#_Toc495745092)

[**4.** **열거형에 추상 메서드 추가하기** 11](#_Toc495745093)

[5. 열거형의 이해 11](#_Toc495745094)

[III. 어노테이션(annotation) 13](#_Toc495745095)

[1. 어노테이션이란? 13](#_Toc495745096)

[2. 표준 어노테이션 13](#_Toc495745097)

[**1)** **@Override** 13](#_Toc495745098)

[**2)** **@Deprecated** 13](#_Toc495745099)

[3) @FunctionalInterface 14](#_Toc495745100)

[4) @SuppressWarnings 14](#_Toc495745101)

[5) @SafeVarargs 14](#_Toc495745102)

[3. 메타 어노테이션 14](#_Toc495745103)

[1) @Target 14](#_Toc495745104)

[2) @Retention 14](#_Toc495745105)

[3) @Documented 15](#_Toc495745106)

[4) @Inherited 15](#_Toc495745107)

[5) @Repeatable 15](#_Toc495745108)

[6) @Native 15](#_Toc495745109)

[4. 어노테이션 타입 정의하기 15](#_Toc495745110)

[1) 어노테이션의 요소 15](#_Toc495745111)

[2) Java.lang.annotation.Annotation 15](#_Toc495745112)

[3) 마커 어노테이션 Marker Annotation 15](#_Toc495745113)

[4) 어노테이션 요소의 규칙 15](#_Toc495745114)

1. 제네릭
   1. 제네릭스(Generics)란?

* 다양한 타입의 객체들을 다루는 메서드나 컬렉션 클래스에 컴파일 시의 타입체크(compile-time type check)를 해주는 기능이다.
* 객체 타입을 컴파일 시에 체크하기 때문에 객체의 타입 안정성을 높이고 형변환의 번거로움이 줄어든다.
* 제네릭의 장점
  + 타입의 안정성을 제공
  + 타입체크와 형변환을 생략할 수 있으므로 코드가 간결해진다.
  1. 제네릭 클래스의 선언

|  |
| --- |
|  |

* + 1. 제네릭스의 용어

|  |
| --- |
| Class Box<T> {}  원시타입  제네릭 클래스 |

* Box<T> 제네릭 클래스, ‘T의 Box’ 또는 ‘T Box’라고 읽는다.
* T 타입변수 또는 타입 매개변수(T는 타입 문자)
* Box 원시 타입(raw type)

|  |
| --- |
| Box<String> b = new Box<String>();  제네릭 타입 호출  대입된 타입(매개변수화된 타입) |

* + 1. 제네릭스의 제한
* Box의 객체를 생성할 때, 객체 별로 다른 타입을 지정하는 것이 적절하다.
* 모든 객체에 대해 동일하게 동작해야 하는 stataic멤버에 타입 변수 T를 사용할 수 없다. -> T는 인스턴스변수로 간주되기 때문이다.
* 제네릭 타입의 배열을 생성하는 것을 허용하지 않는다. -> new 연산자를 사용하여 컴파일 했을 때 정확한 타입이 무엇인지 알아야 한다.
  1. 제네릭 클래스의 객체 생성과 사용

|  |
| --- |
| 제네릭으로 인해 객체의 타입을 동일하게 해야 한다. |
|  |

* 1. 제한된 제네릭 클래스
* 타입 문자(T)로 사용할 타입을 명시하는 것 보다 타입의 종류를 제한 하는 방법
* 제네릭 타입에 ‘extends’를 사용하기 (인터페이스를 구현에도 동일)

|  |
| --- |
| 제네릭 타입에도 상속(인터페이스구현)이 가능하다. |
|  |

* 1. 와일드 카드
* 메서드의 파라미터로 클래스의 제네릭만 달리 하여 오버로딩을 가능하게 하기 위해 사용

|  |
| --- |
| **<? extends T>** 와일드 카드의 상한 제한. T와 그 자손들만 가능  **<? super T>** 와일드 카드의 하한 제한. T와 그 조상들만 가능  <?> 제한 없음. 모든 타입이 가능. <? extends Object> 와 동일 |
|  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 상위 클래스 | 하위 클래스 | | Fruit | Apple | | Grape |  |  |  | | --- | --- | | 인터페이스 | 구현 클래스 | | Comparator<T> | AppleComp | | GrapeComp | | FruitComp | |
|  |
|  |

* 1. 제네릭 메서드 \*\*
  2. 제네릭 타입의 형변환
* 제네릭 타입과 원시 타입간의 형변환이 가능한 방법
  1. 제네릭 타입의 제거
     1. 제네릭 타입의 경계(bound)를 제거한다.
     2. 제네릭 타입을 제거한 후에 타입이 일치하지 않으면, 형변환을 추가한다.

1. **열거형(enums)**
   1. **열거형이란?**
      1. 클래스처럼 보이게 하는 상수
      2. 서로 관련 있는 상수들을 모아 심볼릭한 명칭의 집합으로 정의한 것
      3. Enum 클래스형을 기반으로 한 클래스형 선언
      4. 새로운 열거형을 선언하면, 내부적으로 Enum 클래스형 기반의 새로운 클래스형이 만들어짐
   2. **열거형의 정의와 사용**
      1. 모든 열거형의 조상 – java.lang.Enum

|  |  |
| --- | --- |
| **메서드** | **설명** |
| Class<E> getDeclaringClass() | 열거형의 Class객체를 반환한다. |
| String name() | 열거형 상수의 이름을 문자열로 반환한다. |
| Int ordinal() | 열거형 상수가 정의된 순서를 반환한다. (0부터 시작) |
| T valueOf(Class<T> enumType, String name) | 지정된 열거형에서 name과 일치하는 열거형 상수를 반환한다. |

|  |
| --- |
| **열거형을 직접 정의하고 사용하는 예제** |
|  |

* 1. **열거형에 멤버 추가하기**

|  |
| --- |
|  |
|  |

* 1. **열거형에 추상 메서드 추가하기**

|  |
| --- |
| 열거형의 메서드 추가 방식 |
|  |

* 1. 열거형의 이해

|  |
| --- |
| 열거형의 추상클래스를 상속받은 추상클래스를 구현하는 예제 |
|  |
|  |

1. 어노테이션(annotation)
   1. 어노테이션이란?

* 어노테이션의 뜻은 주석, 주해, 메모이다.
* 어노테이션은 JDK에서 기본적으로 제공하는 것과 다른 프로그램에서 제공하는 것들이 있는데, 어느 것이든 그저 약속된 형식으로 정보를 제공하기만 하면 될 뿐이다.
  1. 표준 어노테이션
* 자바에서 기본적으로 제공하는 어노테이션은 몇 개 없다.
* 일부는 ‘메타 어노테이션(meta annotation)’으로 어노테이션을 정의하는데 사용되는 어노테이션의 어노테이션이다.

|  |  |
| --- | --- |
| **어노테이션** | **설명** |
| @Override | 컴파일러에게 오버라이딩하는 메서드라는 것을 알린다. |
| @Deprecated | 앞으로 사용하지 않을 것을 권장하는 대상에 붙인다. |
| @SuppressWarnings | 컴파일러의 특정 경고메시지가 나타나지 않게 해준다. |
| @SafeVarargs | 제네릭스 타입의 가변인자에 사용한다. (JDK1.7) |
| @FunctionalInterface | 함수형 인터페이스라는 것을 알린다. (JDK1.8) |
| @Native | native메서드에서 참조되는 상수 앞에 붙인다. (JDK1.8) |
| @Target | 어노테이션이 적용가능한 대상을 지정하는데 사용한다. |
| @Dcumented | 어노테이션 정보가 javadoc으로 작성된 문서에 포함되게 한다. |
| @Inherited | 어노테이션이 자손 클래스에 상속되도록 한다. |
| @Retention | 어노테이션이 유지되는 범위를 지정하는데 사용된다. |
| @Repeatable | 어노테이션을 반복해서 적용할 수 있게 한다. (JDK1.8) |

* + 1. **@Override**
* 메서드 앞에 붙일 수 있는 어노테이션
* 조상의 메서드를 오버라이딩하는 것이라는걸 컴파일러에게 알려주는 역할을 한다.
  + 1. **@Deprecated**
* 새로운 버전의 JDK가 소개될 때, 기존의 기능을 대체할 것들이 추가 되어 더 이상 사용하지 않을 것을 권한다는 의미
  + 1. @FunctionalInterface
* 함수형 인터페이스를 올바르게 선언했는지 확인하는 어노테이션
  + 1. @SuppressWarnings
* 컴파일러가 보여주는 경고메시지가 나타나지 않게 억제해주는 어노테이션
  + 1. @SafeVarargs
* 메서드에 선언된 가변인자의 타입이 non-reifiable타입일 경우, 해당 메서드를 선언하는 부분과 호출하는 부분에서 “unchecked”경고가 발생한다. 이 경고를 억제하기 위한 어노테이션
  1. 메타 어노테이션
* ‘어노테이션을 위한 어노테이션’, 즉 어노테이션에 붙이는 어노테이션으로 어노테이션을 정의할 때 어노테이션의 적용대상(target)이나 유지기간(retention)등을 지정하는데 사용한다.
  + 1. @Target
* 어노테이션이 적용 가능한 대상을 지정하는데 사용

|  |  |
| --- | --- |
| **대상 타입** | **의미** |
| ANNOTATION\_TYPE | 어노테이션 |
| CONSTRUCTOR | 생성자 |
| FILED | 필드(멤버변수, enum상수) |
| LOCAL\_VARIABLE | 지역변수 |
| METHOD | 메서드 |
| PACKAGE | 패키지 |
| PARAMETER | 매개변수 |
| TYPE | 타입(클래스, 인터페이스, enum) |
| TYPE\_PARAMETER | 타입 매개변수(JDK1.8) |
| TYPE\_USE | 타입이 사용되는 모든 곳(JDK1.8) |

* + 1. @Retention
* 어노테이션이 유지(retention)되는 기간을 지정하는데 사용된다.

|  |  |
| --- | --- |
| **유지 정책** | **의미** |
| SOURCE | 소스 파일에만 존재. 클래스파일에는 존재하지 않음. |
| CLASS | 클래스 파일에 존재. 실행 시에 사용불가. 기본값 |
| RUNTIME | 클래스 파일에 존재. 실행 시에 사용가능 |

* + 1. @Documented
* 어노테이션에 대한 정보가 javadoc으로 작성한 문서에 포함되도록 한다.
  + 1. @Inherited
* 어노테이션이 자손 클래스에 상속되도록 한다.
  + 1. @Repeatable
* 여러 종류의 어노테이션을 붙이기 위한 어노테이션
  + 1. @Native
* 네이티브 메서드(native method)에 의해 참조되는 ‘상수 필드(constant field)’에 붙이는 어노테이션
  1. 어노테이션 타입 정의하기
     1. 어노테이션의 요소
* 어노테이션 내에 선언된 메서드를 ‘어노테이션의 요소(element)’라고 한다.
  + 1. Java.lang.annotation.Annotation
* 모든 어노테이션의 조상은 Annotation이다.
* 하지만 어노테이션은 상속이 허용되지 않는다.
* Annotation은 어노테이션이 아니라 인터페이스로 정의되어 있다.
  + 1. 마커 어노테이션 Marker Annotation
* 값을 지정할 필요가 없는 경우, 어노테이션의 요소를 하나도 정의하지 않을 수 있다.
  + 1. 어노테이션 요소의 규칙
* 요소의 타입은 기본형, String, enum, 어노테이션, class만 허용된다.
* ()안에 매개변수를 선언할 수 없다.
* 예외를 선언할 수 없다.
* 요소를 타입 매개변수로 정의할 수 없다.

|  |
| --- |
| 어노테이션을 직접 정의하고, 어노테이션의 요소의 값을 출력하는 방법 |
|  |