SECTION 1. 상속

1. 자바 클래스의 상속은 단일 상속이다. > 다중 상속은 객체의 명확성이 떨어진다!
2. 특정 클래스가 가지는 일부 속성과 기능을 다른 새로운 클래스에게 제공하기 위해 맺는 클래스간의 관계를 말한다.

Base class = super class = parent class

* 상속을 주기 위해 준비된 특정 클래스

Derivation class = sub class = child class

* 특정 클래스로부터 상속을 받아 새롭게 정의되는 클래스

1. 클래스 상속은 객체의 재사용과 코드의 간결성을 제공해준다.
2. Overriding: 메서드 재정의라고 불리며 메서드 명이 같아야 하고 인자가 같아야 한다.

리턴형도 같아야 한다. super 클래스의 메서드보다 sub 클래스의 메서드 접근제한이 동일하거나 더 넓어야한다.

: 부모 클래스가 가지는 메서드를 자식 클래스에서 똑같이 재정의하는 것.

: 부모 클래스의 기능을 변경하거나 필요에 따라 확장을 필요할 때 사용한다.

super와 super()

1. 개념

: 특정 객체 내에서 자기 자신의 객체를 참조할 수 있는 유일한 참조변수인 this와 함께 객체를 참조할 수 있는 참조변수이다.

: 현재 객체의 바로 위의 super 클래스(부모 클래스)를 참조한다.

: 오버라이딩으로 멤버 은폐가 되어 super 클래스의 일부를 사용할 수 없느 ㄴ상태가 되었을 때 은폐된 것을 sub 클래스에서 호출하고자 할때 또는 super 클래스 자체를 참조하고자 할 때 사용하는 것이 super라는 예약어이다.

SECTION 2. final 예약어

1. 개념

: 프로그램이 끝날 때까지 값을 변경하지 못하는 Constant화가 된다.

: 상수는 static 예약어와 함께 정적화하여 특정 클래스의 객체가 여러 개 생성되더라도 오로지 1개만 만들어지도록 하는 것이 보통이다.

: 메서드에 final을적용하면 상속관계에 따른 오버라이딩으로의 확장을 할 수 없게 된다.

: 클래스에 final을 적용하면 더 이상 상속을 허용하지 않은 종단 클래스가 된다.

SECTION 3. 추상화

1. 개념

: 변수와 메소드를 encapsulation하고 information hidihng 까지 한 것

: 구체적인 개념으로부터 공통된 부분들만 추려내어 일반화 할 수 있도록 하는 것

|  |  |
| --- | --- |
| 구성 | 사용 예 |
| [접근제한] abstract void [메서드 명](); | public abstract void absTest(); |
| [접근제한] abstract class [클래스 명](); | public abstract class AbsEx(); |

2. 추상 클래스의 상속 관계

: 일반 클래스간의 상속과 유사하지만 추상 클래스간의 상속에서는 상속받은 추상 메서드들을 꼭 재정의 할 필요는 없다.

: 특정 클래스에서 일부 기능은 공통된 기능이지만일부 기능이 상속받는 각 클래스들마다 특징적인 변화가 있을 때 상속에 의해 다른 부분만 오버라이딩하여 사용한다.

3. 정리

: 추상 크래스는 공통된 부분만 구현하고 공통되지 않은 부분들은 abstract라는 예약어를 사용한 미완성된 클래스이므로 다른 클래스로 상속되어 미완성된 부분들을 재정의하여 완성시킨 뒤 사용할 수 있다.

: 추상클래스의 필요성이라 함은 여러 개의 클래스들이 상당수의 공통된 점을 가지고 있고 소수의 다른 점들이 있을 때 효율적인 것이다.

SECTION 7. 제네릭

자바에서 제네릭이란 데이터의 타입을 일반화(Generalize)하는 것을 의미한다.

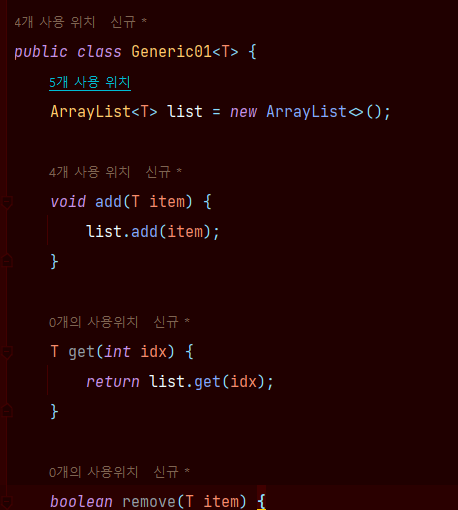
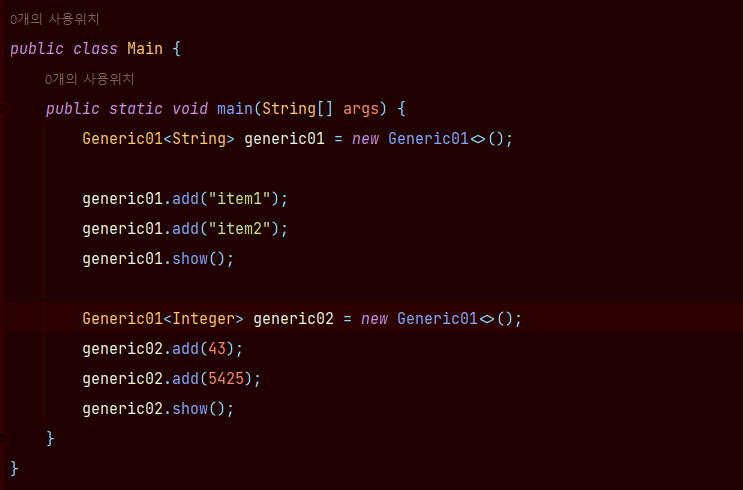
제네릭은 클래스나 메소드에서 사용할 내부 데이터 타입을 컴파일 시에 미리 지정하는 방법이다.

이렇게 컴파일 시에 미리 type check를 수행하면 아래와 같은 장점을 갖는다.

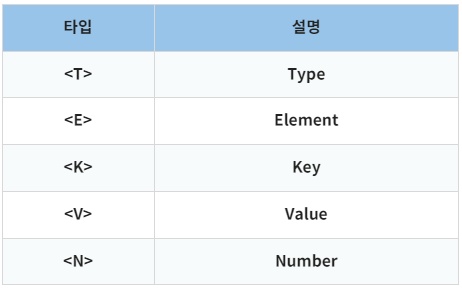
* 클래스나 메소드 내부에서 사용되는 객체의 타입 안정성을 높일 수 있다.
* 반환값에 대한 타입 변환 및 타입 검사에 들어가는 노력을 줄일 수 있다.

JDK 1.5 이전에서는 여러 타입을 사용하는 대부분의 클래스나 메소드에서 인수나 반환값으로 Object타입을 사용했다. 하지만 이 경우에는 반환된 Object 객체를 다시 원하는 타입으로 타입 변환해야 하며, 이때 오류가 발생할 가능성도 존재한다. 하지만 JDK 1.5 부터 도입된 제네릭을 사용하면 컴파일 시에 미리 타입이 정해지므로 타입 검사나 타입 변환과 같은 번거로운 작업을 생략할 수 있다.

1. 제네릭 선언 및 생성



* 제네릭 타입 변수



임의의 참조형 타입이라고 하고 위 코드에서 <T>와 같은 것을 말한다. 반드시 위 설명대로 쓰지 않아도 되고 여러 개의 타입 변수는 쉼표로 구분하여 명시할 수 있다. 타입 변수는 클래스에서 뿐만 아니라 메소드의 매개변수나 반환 값으로도 사용할 수 있다.

위와 같이 선언된 제네릭 클래스를 생성할 때에는 타입 변수 자리에 사용할 실제 타입을 명시해야 한다. JAVA에서 타입 변수 자리에 사용할 실제 타입을 명시할 때 기본 타입을 바로 사용할 수는 없다. 이때는 Wrapper class를 사용해야 한다.

1. 제네릭의 제거 시기

자바 코드에서 선언되고 사용된 제네릭 타입은 컴파일 시 컴파일러에 의해 자동으로 검사되어 타입 변환된다. 그리고서 코드 내의 모든 제네릭 타입은 제거되어 컴파일된 class 파일에는 어떤 제네릭 타입도 포함되지 않게 된다.

이런 식으로 동작하는 이유는 제네릭을 사용하지 않는 코드와의 호환성을 유지하기 위해서이다.

1. 타입 변수의 제한

제네릭은 타입 변수를 사용하여 타입을 제한하는데 이때 extends 키워드를 사용하면 타입 변수에 특정 타입만 사용하도록 제한할 수 있다.

class list<T extends Aclas> { … }

위와 같이 클래스의 타입 변수에 제한을 걸어 놓으면 클래스 내부에서 사용된 모든 타입 변수에 제한이 걸린다. 이때에는 클래스가 아닌 인터페이스를 구현할 경우에도 implements가 아닌 extends 키워드를 사용해야 한다.

Class list<T extends Aclas & Ainter> { … }

1. 제네릭 메소드

제네릭 메소드란 메소드의 선언부에 타입 변수를 사용한 메소드를 의미한다.

Public static <T> void sort(…) {…}

이때 타입 변수의 선언은 메소드 선언부에서 반환 타입 바로 앞에 위치한다.

class list<T> {

…

public static <T> void sort(List<T> list, Comparator<? super T comp) {

… }

…

}

위 코드에서 제네릭 클래스에 정의된 타입변수 T와 제네릭 메소드에 정의된 타입변수 T는 별개의 것이다.

1. 와일드카드

이름에 제한을 두지 않음을 표현하는데 사용되는 기호이다.

자바의 제네릭에서는 물음표(?) 기호를 사용하여 이러한 와일드카드를 사용할 수 있다.

<?> // 타입 변수에 모든 타입을 사용할 수 있음.

<? extends T> // T 타입과 T 타입을 상속받는 자손 클래스 타입만을 사용할 수 있음.

<? super T> // T 타입과 T 타입이 상속받은 조상 클래스 타입만을 사용할 수 있음