Part 4. Collection

2017년 12월 27일 수요일 오후 12:05

Collection

- Java Collection에는 List, Map, Set 인터페이스를 기준으로 여러 구현체가 존재한다.
- 다수의 데이터를 다루는데 표준화된 클래스들을 제공해주기 때문에 편하게 Data Structure를 직접 구현하지 않고 사용할 수 있다.
- 배열과 다르게 객체를 보관하기 위한 공간을 미리 정하지 않아도 되므로, 상황에 따라 객체의 수를 동적으로 정할 수 있고 프로그램의 공간적인 효율성을 향상시킨다.
- List
    - 대표적으로 ArrayList와 LinkedList가 있다.
    - ArrayList의 경우, 기존의 Vector를 개선한 것이며, 내부적으로는 RedBlack Tree로 구성되어 잇다.
- Map
    - 대표적으로 HashMap이 있다.
    - HashMap의 경우, Key-Value의 구조를 가지며, Key를 기준으로 중복된 값을 저장하지 않으며 순서를 보장하지 않는다.
    - 순서를 보장하는 LinkedHashMap도 지원한다.
- Set
    - 대표적으로 HashSet이 있다.
    - HashSet의 경우, Value에 대해 중복된 값을 저장하지 않으며, 내부적으로 Value를 Key로 하는 자료구조이다.
    - 순서를 보장하는 LinkedHashSet도 지원한다.
- Stack과 Queue

Annotation

- 본래의 주석이랑 뜻으로, 인터페이스를 기반으로 한 문법이다.
- 주석과는 다르지만, 어노테이션을 코드에 달아 클래스에 특별한 의미를 부여하거나 기능을 주입하는 역할을 하며, 이 의미는 컴파일 타임 또는 런타임에 해석될 수 있다.
- 프로그램의 소스코드 안에 다른 프로그램을 위한 정보를 미리 약속된 형식으로 포함하기 위해 생겨났다.
- 어노테이션에는 크게 3 가지 종류가 있다.
    - Built-in Annotation
        - JDK에 내장된 어노테이션
        - 주로 컴파일러를 위한 것으로, 컴파일러에게 유용한 정보를 제공한다.
        - @Override
    - Meta Annotation
        - 어노테이션에 대한 정보를 나타내기 위한 어노테이션
        - 해당 어노테이션의 동작 대상을 결정한다.
    - Custom Annotation

        - 개발자가 직접 만들어내는 어노테이션

- 기존에는 XML 설정 파일에 명시하는 방법으로 프로그래밍 되었다. 변경될 수 있는 데이터를 코드가 아닌 외부 설정 파일에 분리하기 때문에,컴파일없이 쉽게 변경할 수 있었지만, 매번 많은 설정을 작성해야 하는 불편함이 생겼고, 이는 웹 어플리케이션의 규모가 커지면서 부각되었다. 이를 해결하기위해 어노테이션이 등장했다.

Generic

- 제네릭은 자바에서 안정성을 담당한다.
- 다양한 타입의 객체를 다루는 메서드나 컬렉션 클래스에서 사용하는 것으로, 컴파일 과정에서 타입 체크 기능을 지원한다.
- 객체의 타입 안정성을 높이고 형 변환의 불편함을 감소시키며, 이는 자연스럽게 코드를 간결하도록 도와준다.
- 클래스를 정의 할 때, 데이터 타입을 확정하지 않고 인스턴스를 생성할 때 데이터 타입을 지정하는 기능을 제네릭이라 한다.

Final Keyword

- final class
    - 다른 클래스에서 상속하지 못하도록 한다.
- final method
    - 다른 메소드에서 오버라이딩하지 못하도록 한다.
- final variable
    - 변하지 않는 상수 값이 되어 새로 할당할 수 없는 변수가 된다.

- finally
    - try-catch 또는 try-catch-resource 구문을 사용할 경우, 정상적으로 작업을 한 경우와 에러가 발행했을 경우를 포함하여 마무리하는 작업이 존재하는 경우에 해당하는 코드를 작성해주는 코드 블록이다.
- finalize()
    - 메소드로, GC에 의해 호출되는 함수이다.
    - Object 클래스에 정의되어 있으며 GC가 발생하는 시점이 불분명하기 때문에 해당 메소드가 실행된다는 보장이 없다.

Overriding vs Overloading

- 오버라이딩 (Overriding)
    - 상속받은 클래스에 존재하는 메소드를 하위 클래스에서 필요에 맞게 재정의하는 것을 의미
- 오버로딩 (Overloading)
    - 같은 클래스 내에 retrun value와 메소드 명이 동일한 메소드를 매개 변수만 다르게 만들어 다양한 상황에 메소드가 호출될 수 있도록 하는 것

Access Modifier

- 변수 또는 메서드의 접근 범위를 설정해주기 위해서 사용하는 Java의 예약어

public	어떤 클래스에서도 접근 가능
protected	클래스가 정의되어 있는 해당 패키지 내 그리고 헤당 클래스를 상속받은 외부 패키지의 클래스에서 접근 가능
(default)	클래스가 정의되어 있는 해당 패키지 내에서만 접근이 가능하도록 접근 범위 제한
private	정의된 해당 클래스에서만 접근이 가능하도록 접근 범위 제한

Wrapper Class

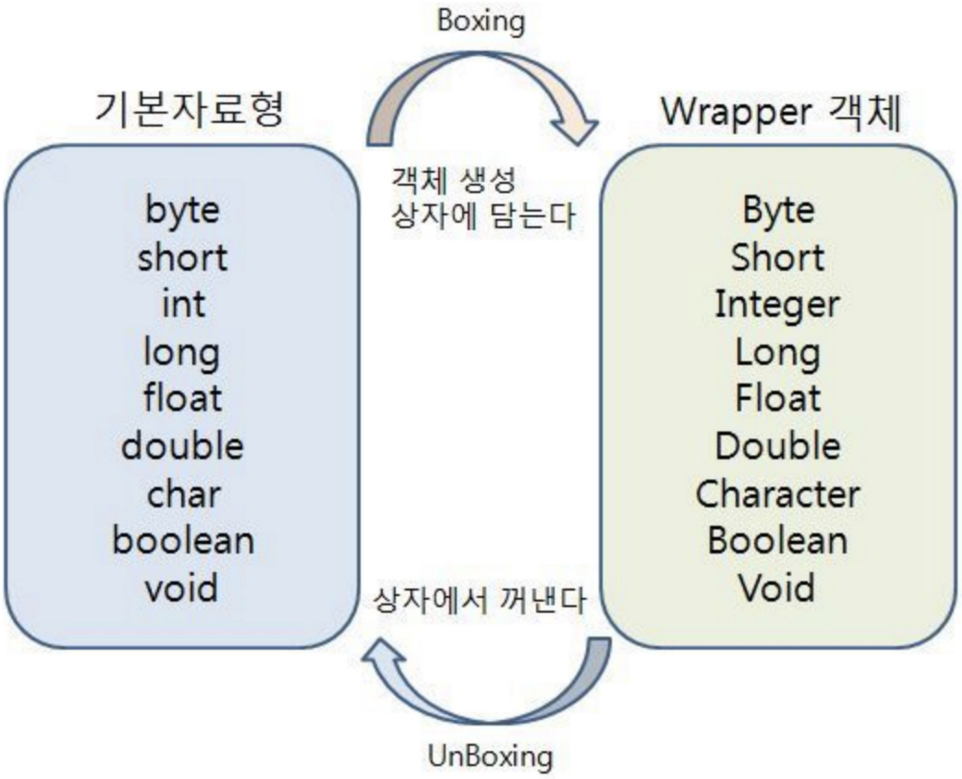
- 자바는 기본형(Primitive Type)과 참조형(Reference Type)으로 나뉜다.
- 데이터를 저장할 때 기본형 타입의 변수에 저장할 수 있고, 다양한 객체를 저장할 수 있는 컨테이너 역할을 하는 객체를 생성할 수 있다.
- 어떤 상황에서는 기본형 타입을 객체로 사용해야 하는 경우, 기본형 타입 값을 객체로 포장할 필요가 있다.
    - 매개 변수로 객체가 요구될 때
    - 기본형 값이 아닌 객체로 저장해야 할 때
    - 객체간의 비교가 필요할 때
- 자바의 각 기초 타입에 대해 Java 클래스 라이브러리 내에 상응되는 Wrapper Class가 존재한다. 모든 Wrapper Class는 java.lang 패키지에 정의된다.

wrapper class

기본형	래퍼클래스	생성자
boolean	Boolean	Boolean(boolean value) Boolean(String s)
char	Character	Character(char value)
byte	Byte	Byte(byte value) Byte(String s)
short	Short	Short(short value) Short(String s)
int	Integer	Integer(int value) Integer(String s)
long	Long	Long(long value) Long(String s)
float	Float	Float(double value) Float(float value) Float(String s)
double	Double	Double(double value) Double(String s)

Boxing과 UnBoxing

- Wrapper 클래스는 산술 연산을 위해 정의된 클래스가 아니기 때문에, 이 클래스의 인스턴스에 저장된 값을 변경하는 것은 불가능하며 값을 저장하는 새로운 객체의 생성 및 참조만 가능하다.
- ex) Integer age = new Integer(30);
- ex) int age2 = age.intValue();

- 

AutoBoxing과 AutoUnBoxing

- JDK 1.5 버전 이후에는 자동으로 Boxing과 UnBoxing을 처리하도록 AutoBoxing과 AutoUnBoxing을 제공한다.
- ex) Integer obj = 28;
- ex) Integer obj = new Integer(28);
  int num = obj;
- AutoBoxing과 AutoUnBoxing은 단지 기본형 타입과 사응하는 Wrapper class에만 일어난다. 다른 경우에 컴파일 에러가 발생한다.
- Integer는 intValue(), Double은 doubleValue() 등만 AutoBoxing과 AutoUnBoxing이 발생한다.
- ex) Double obj = 3.14;
  int num = obj.intValue();

*** Wrapper 클래스의 기본 메소드들**

-

메소드	반환값	설명
booleanValue()	boolean	기본형 데이터를 문자열로 바꾼 뒤에 반환
byteValue()	byte	객체의 값을 byte 값으로 변환하여 반환
doubleValue()	double	객체의 값을 double 값으로 변환하여 반환
floatValue()	float	객체의 값을 float 값으로 변환하여 반환
intValue()	int	객체의 값을 int 값으로 변환하여 반환
longValue()	long	객체의 값을 long 값으로 변환하여 반환
shortValue()	short	객체의 값을 short 값으로 변환하여 반환