Java

Wrapper 객체

Primitive Types (기본 자료 형)

정의

- 자바의 기본 데이터 타입은 값을 직접 저장하는 방식
  - 메모리 효율성이 좋고, 성능이 빠르지만, 객체로 취급되지 않아 객체 관련 메서드나 컬렉션에서 사용할 수 없다.
  - int, double, char, boolean 등.

- Primitive Type 사용
  - 기본 데이터 타입인 int, double 변수를 직접 선언하고 출력
  - int primitiveInt = 10;
  - double primitiveDouble = 20.5;

Wrapper Classes

정의

- 기본 자료형을 객체로 감싸는 클래스
  - 기본 타입을 감싸 객체로 변환할 수 있는 클래스
  - 객체로 사용되므로 메서드를 호출하거나, 컬렉션과 같은 객체를 다루는 데이터 구조에 넣을 수 있다.
  - 래퍼 클래스는 불변(immutable)
  - Integer, Double, Character, Boolean, 등.

- Wrapper Class 사용
  - 기본 타입을 Integer와 Double로 래핑하여 객체로 변환

  - Integer wrapperInt = Integer.valueOf(primitiveInt); // int를 Integer로 래핑
  - Double wrapperDouble = Double.valueOf(primitiveDouble); // double을 Double로 래핑

Wrapper Classes

정의

- Autoboxing
  - 기본 타입을 자동으로 래퍼 클래스로 변환하는 기능
  - primitive -> wrapper 자동 변환

  - Integer autoBoxedInt = primitiveInt;
  - Double autoBoxedDouble = primitiveDouble;

- Unboxing
  - 래퍼 클래스 객체를 다시 기본 타입으로 자동 변환하는 기능
  - wrapper -> primitive 자동 변환

  - int unboxedInt = wrapperInt;
  - double unboxedDouble = wrapperDouble;

Wrapper Classes

parseXXX()

- 문자열을 해당 기본 타입으로 변환
  - parseInt(), parseDouble() 등과 같은 메소드들은 문자열을 정수나 실수 같은 기본 타입으로 변환

- 문자열 "100"을 정수 100으로 변환
  - String numberStr = "100";
  - int number = Integer.parseInt(numberStr);

- 문자열 "99.99"을 실수 99.99로 변환
  - String decimalStr = "99.99";
  - double decimal = Double.parseDouble(decimalStr);

Wrapper Classes

valueOf()

- 문자열 또는 기본 타입을 래퍼 객체로 변환
  - 주어진 값을 래퍼 클래스 객체로 변환
  - valueOf()는 캐싱을 사용하므로 작은 범위의 정수는 새로운 객체를 생성하지 않고, 미리 생성된 객체를 반환한다.

- 문자열 "200"을 Integer 객체로 변환
  - String numberStr = "200";
  - Integer wrapperInteger = Integer.valueOf(numberStr);

- 기본 타입 300을 Integer 객체로 변환
  - int num = 300;
  - Integer wrapperInteger2 = Integer.valueOf(num);

Wrapper Classes

toString()

- 객체를 문자열로 변환
  - 래퍼 클래스의 toString() 메소드는 객체 값을 문자열로 변환한다.

- Integer 객체의 값을 문자열 "150"으로 변환
  - Integer number = 150;
  - String numberStr = number.toString();

Wrapper Classes

compareTo()

- 두 래퍼 객체 비교
  - 두 래퍼 객체를 비교하고, 비교 결과에 따라 음수, 0, 양수를 반환
  - 음수 : 호출한 객체가 비교 대상보다 작음.
  - 0 : 두 값이 동일함.
  - 양수 : 호출한 객체가 비교 대상보다 큼.

- 100과 200을 비교하여 -1을 반환
  - Integer num1 = 100;
  - Integer num2 = 200;

  - int result = num1.compareTo(num2);

Wrapper Classes

equals()

- 객체 값 비교
  - 두 래퍼 객체의 값을 비교할 때 equals()를 사용

- 객체의 실제 값을 비교
  - 두 객체의 값이 동일하므로 true를 반환

  - Integer num1 = 300;
  - Integer num2 = 300;

  - boolean isEqual = num1.equals(num2);

Wrapper Classes

hashCode()

- 해시 코드
  - 객체를 식별하는 정수 값
  - 객체가 HashMap이나 HashSet과 같은 해시 기반 컬렉션에 저장될 때 사용 된다.

- hashcode() 메서드
  - 주로 해시 기반의 컬렉션에서 객체를 빠르게 검색하고 저장할 때 사용

- 객체의 해시 코드 반환
  - 객체의 고유한 해시 코드를 반환하며, 이 해시 코드는 equals() 메소드와 함께 많이 사용 된다
  - Integer number = 400;
  - int hashCode = number.hashCode();

- equals() 메서드를 재정의할 때는 반드시 hashCode() 메서드도 재정의해야 한다.
  - 두 객체가 equals()로 같다면, 두 객체의 hashCode() 값도 동일해야 한다.
  - 두 객체가 equals()로 다르더라도, hashCode() 값은 같을 수 있다

Wrapper Classes

Collection

- 컬렉션 프레임워크에서는 기본 데이터 타입(예: int, char) 대신 래퍼 객체를 사용해야 한다.
  - 컬렉션 클래스(예: ArrayList, HashSet)는 객체만 저장할 수 있기 때문에, 기본 타입을 사용할 수 없다

- 컬렉션에서 래퍼 객체를 사용하는 이유
  - 컬렉션 클래스는 객체만 저장
  - null을 가질 수 있다
  - 자동 박싱과 언박싱

Object 객체

정의

- 모든 Java 클래스의 최상위 슈퍼클래스
  - 모든 객체는 Object 클래스를 상속받기 때문에, Object 클래스에 정의된 메서드는 모든 Java 객체에서 사용할 수 있다.

Object 객체

메서드

- toString()
  - 객체의 정보나 상태를 문자열 형식으로 반환
- equals(Object obj)
  - 두 객체가 같은지를 비교
- hashCode()
  - 객체의 해시 코드를 반환
- clone()
  - 객체의 복사본을 생성
  - Cloneable 인터페이스를 구현한 클래스에서 사용 된다.

Object 객체

메서드

- getClass()
  - 클래스 정보를 동적으로 얻는 데 사용
  - Class<?> clazz = obj.getClass();
- getClass().getName()
  - 객체의 클래스 이름을 문자열로 반환
  - String className = obj.getClass().getName();
- getClass().getSimpleName()
  - 객체의 클래스 이름을 간단한 이름 형식으로 반환
  - String simpleName = obj.getClass().getSimpleName();

Arrays 객체

정의

- Arrays 클래스
  - 배열을 다루기 위한 다양한 메서드를 제공
  - 모든 메서드가 정적(static)으로 정의되어 있어 인스턴스를 생성하지 않고도 사용할 수 있다.

Arrays 객체

배열 정렬 및 검색

- sort(int[] a)
  - 주어진 int 배열을 오름차순으로 정렬

- sort(T[] a)
  - 객체 배열을 오름차순으로 정렬

- sort(T[] a, Comparator<? super T> c)
  - 주어진 배열을 사용자 정의 Comparator에 따라 정렬

Arrays 객체

배열 정렬 및 검색

- binarySearch(int[] a, int key)
  - 정렬된 int 배열에서 주어진 키를 이진 검색하여 인덱스를 반환
  - 배열이 정렬되어 있어야 한다.

- binarySearch(T[] a, T key, Comparator<? super T> c)
  - 정렬된 배열에서 주어진 키를 Comparator를 사용하여 이진 검색
  - 배열이 정렬되어 있어야 한다.

Arrays 객체

배열 복사 및 변환

- copyOf(T[] original, int newLength)
  - 원본 배열을 지정된 길이만큼 복사하여 새 배열을 생성
  - 새 배열의 길이가 원본 배열보다 짧으면 일부 요소가 잘린다.
- copyOfRange(T[] original, int from, int to)
  - 원본 배열의 지정된 범위(from 포함, to 제외)를 복사하여 새 배열을 생성
- copyOf(int[] original, int newLength)
  - int 배열을 지정된 길이만큼 복사하여 새 배열을 생성
  - 새 배열의 길이가 원본 배열보다 짧으면 일부 요소가 잘린다.
- copyOfRange(int[] original, int from, int to)
  - int 배열의 지정된 범위(from 포함, to 제외)를 복사하여 새 배열을 생성
- asList(T... a)
  - 배열을 변환하여 List 인터페이스를 구현하는 객체를 반환한다.

Arrays 객체

배열 채우기 및 비교

- fill(int[] a, int val)
  - 주어진 int 배열의 모든 요소를 지정된 값으로 채운다.
- fill(T[] a, T val)
  - 주어진 객체 배열의 모든 요소를 지정된 객체 값으로 채운다.
- setAll(int[] array, IntUnaryOperator generator)
  - 주어진 int 배열의 모든 요소를 IntUnaryOperator에 의해 생성된 값으로 설정
- equals(int[] a, int[] a2)
  - 두 int 배열이 동일한지를 비교
- equals(T[] a, T[] a2)
  - 두 객체 배열이 동일한지를 비교

String 객체

정의

- 문자열을 다루기 위한 불변(immutable) 클래스
  - 문자열 조작을 위한 다양한 메서드를 제공한다.

- 불변성 (Immutability):
  - String 객체는 한 번 생성되면 변경할 수 없다.

- 메모리 관리
  - 문자열 상수 풀을 사용하여 동일한 문자열 리터럴을 공유 한다.

- 자동 변환
  - 문자열과 기본 데이터 타입 간의 자동 변환을 지원

String 객체

String 객체 생성

- 리터럴 사용
  - String str1 = "Hello";

- new 키워드 사용
  - new 키워드는 항상 새로운 객체를 생성
  - String str2 = new String("World");

String 객체

메서드

- length()
  - 문자열의 길이를 반환
- charAt(int index)
  - 문자열에서 지정된 인덱스의 문자를 반환
- substring(int beginIndex, int endIndex)
  - 문자열의 지정된 범위에 해당하는 부분 문자열을 반환
- indexOf(String str)
  - 문자열 내에서 지정된 문자열의 첫 번째 발생 위치의 인덱스를 반환
  - 문자열이 포함되어 있지 않으면 -1을 반환
- lastIndexOf(String str)
  - 문자열 내에서 지정된 문자열의 마지막 발생 위치의 인덱스를 반환

String 객체

메서드

- toLowerCase()
  - 문자열의 모든 문자를 소문자로 변환하여 새로운 문자열을 반환
- toUpperCase()
  - 문자열의 모든 문자를 대문자로 변환하여 새로운 문자열을 반환
- trim()
  - 문자열의 시작과 끝에서 공백 문자를 제거한 새로운 문자열을 반환
- replace(CharSequence target, CharSequence replacement)
  - 문자열 내의 모든 지정된 문자열을 다른 문자열로 교체
- split(String regex)
  - 지정된 정규 표현식에 따라 문자열을 분리하여 문자열 배열을 반환
- valueOf(int i)
  - 정수 값을 문자열로 변환하여 반환

String 객체

메서드

- concat(String str)
  - 지정된 문자열을 현재 문자열의 끝에 추가하여 새로운 문자열을 반환
- contains(CharSequence sequence)
  - 문자열이 지정된 시퀀스를 포함하고 있는지를 검사하여 true 또는 false를 반환
- startsWith(String prefix)
  - 문자열이 지정된 접두사로 시작하는지를 검사하여 true 또는 false를 반환
- endsWith(String suffix)
  - 문자열이 지정된 접미사로 끝나는지를 검사하여 true 또는 false를 반환
- equals(Object another)
  - 두 문자열이 같은지 비교

Java

컬렉션 프레임웍

wuridl@naver.com

컬렉션 프레임 워크(Collection Framework)

정의

- 컬렉션 프레임 워크
  - 데이터 집합을 저장하는 클래스들을 표준화한 설계
  - 데이터 그룹을 다루고 표현하기 위한 단일화된 구조

- 컬렉션
  - 데이터의 그룹

- 프레임 워크
  - 표준화된 프로그래밍 방식

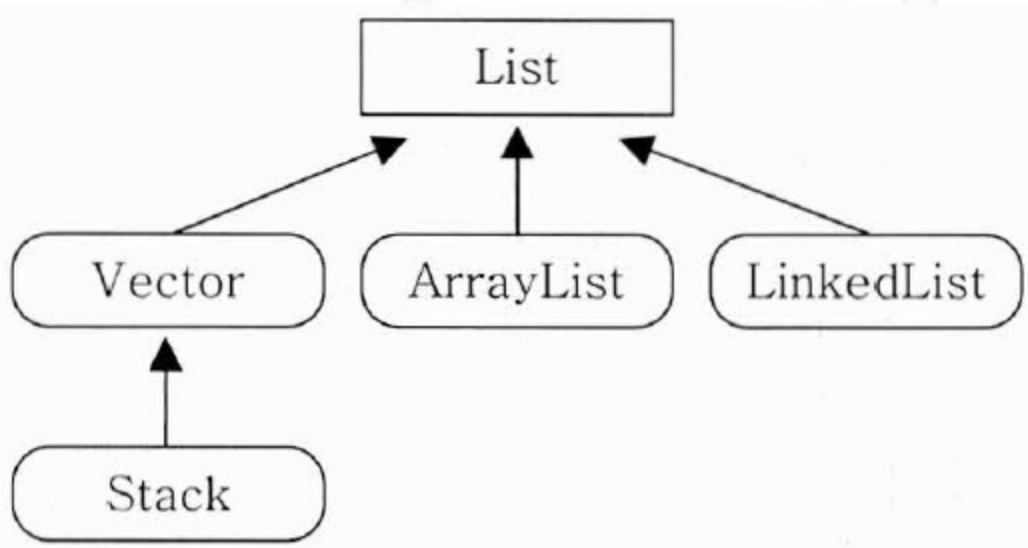
- 컬렉션 프레임워크의 핵심 인터페이스
  - List : 순서가 있는 데이터의 집합, 중복 허용
  - Set : 순서가 없는 데이터의 집합, 중복을 허용하지 않는다.
  - Map : 키와 값의 쌍으로 이루어진 데이터 집합

List Interface

정의

- List interface
  - 중복을 허용하면서,저장 순서가 유지되는 컬렉션을 구현하는데 사용

- List interface를 구현한 클래스
  - ArrayList
  - LinkedList
  - Vector
  - Stack



제네릭스(Generics)

정의

- 제네릭스(Generics)
  - 다양한 타입의 객체들을 다루는 메서드나 컬렉션 클래스에 컴파일 시의 타입 체크를 해 주는 기능

- 컬렉션 클래스는 다양한 종류의 객체를 취급 할 수 있기 때문에 사용할 때 마다 타입 체크를 하고 형 변환을 수행 해야 한다.
  - 객체의 타입을 컴파일 시에 체크하기 때문에 객체의 타입 안정성을 높이고 형변환의 번거로움이 줄어든다

- 제네릭스를 사용하면 의도하지 않은 타입의 객체를 저장하는 것을 방지할 수 있다
  - 객체의 타입을 미리 명시해서 형 변환을 하지 않아도 되게 한다

Iterator Interface

Iterator

- Iterator 인터페이스
  - 컬렉션에 저장된 요소를 읽어 오기 위해 사용하는 표준화된 인터페이스
  - Collecton 인터페이스에는 Iterator를 반환하는 iterator()를 정의 하고 있다
  - Collection인터페이스의 자손인 List와 Set에도 포함 되어 있다

- 메서드
  - Boolean hasNext() : 읽어올 요소가 남아 있는지 확인한다
  - Object next() : 다음 요소를 읽어 온다
  - Void remove() : next()로 읽어온 요소를 삭제 한다.

Iterator Interface

Iterator

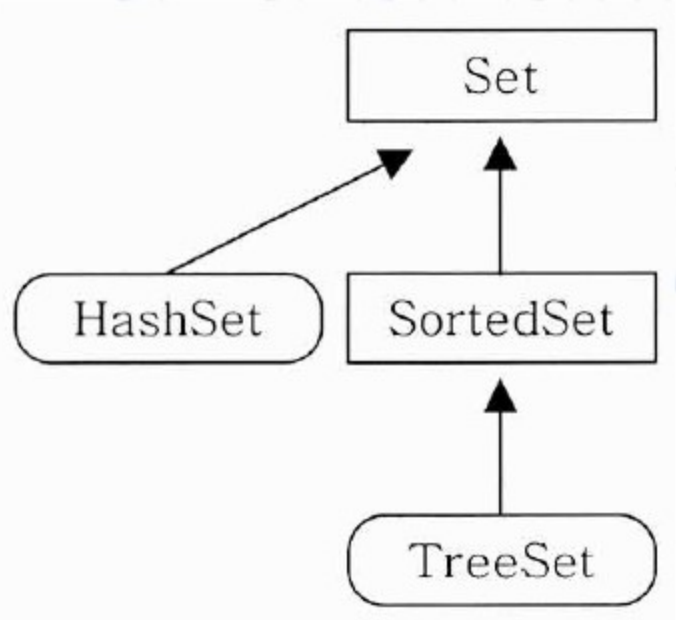
- Map 인터페이스를 구현한 컬렉션 클래스는 키와 값을 쌍으로 저장하고 있기 때문에 iterator()을 직접 호출 할 수 없다
  - keySet() 또는 entrySet()메서드를 통해 키와 값을 Set의 형태로 얻어 온 후에 다시 iterator()를 호출

Set Interface

정의

- Set interface
  - 중복을 허용하지 않고, 저장 순서가 유지되지 않는 컬렉션을 구현하는데 사용

- Set interfac를 구현한 클래스
  - HashSet
  - TreeSet : SortedSet interface를 구현한 클래스



Set Interface

HashSet

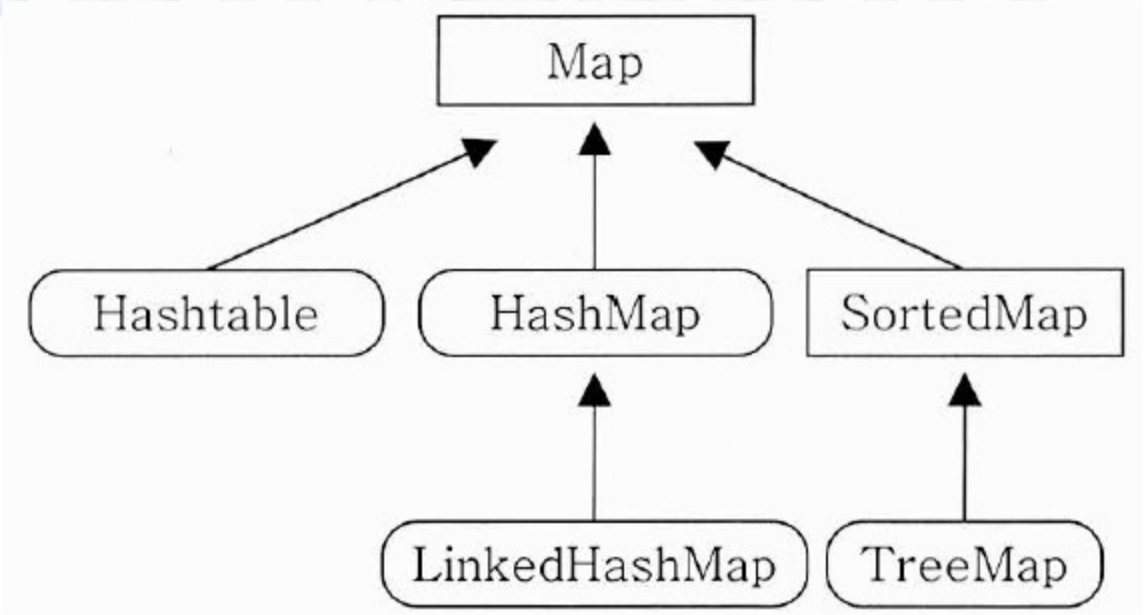
- HashSet
  - Set인터페이스를 구현한 가장 대표적인 컬렉션
  - Add, addAll메서드는 중복된 요소를 추가 하려고 하면 false를 반환한다
  - 저장 순서를 유지하기위해서 LinkedHashSet를 사용한다

Map Interface

정의

- Map interface
  - Key와 value를 하나의 쌍으로 묶어서 저장하는 컬렉션 클래스를 구현하는데 사용
  - 키는 중복될 수 없지만 값은 중복을 허용한다
  - 중복된 키와 값을 저장하면 마지막에 입력한 값이 저장된다

- Map interfac를 구현한 클래스
  - Hashtable
  - HashMap
  - LinkedHashMap
  - SortedMap
  - TreeMap



Map Interface

정의

- Map interface / Entry interface

```
public interface Map{
        ...
        interface Entry{
                Object getKey();
                Object getValue();
                Object getValue(Object  value);
                boolean equals(Object o);
                int hashCode();
        }
}
```

Map Interface

HashMap

- HashMap
  - Entry라는 내부 클래스를 정의 하고 Entry타입의 배열을 선언하고 있다
  - 키와 값을 각각 Object타입으로 저장한다.

- Entry
  - 키와 값을 하나의 클래스로 정의해서 접근한다

```
Entry [] table;

Class Entry{
        Object key;
        Object value;
}
```