# 웹 개발자 부트캠프 과정

CodingOn

# Wrapper 클래스

# Wrapper 클래스

- 8가지 기본 자료형(primitive type)을 객체로 표현하기 위해 제공되는 클래스
- 객체로서 다양한 메서드와 속성을 사용

- 포장하고 있는 기본 타입은 외부에서 변경할 수 없으며, 객체로 생성하는데 목적이 있음
- 포장 객체를 생성하기 위한 클래스는 java.lang 패키지에 포함되어 있음

| 기본 타입 | 포장 클래스 |
|---|---|
| byte | Byte |
| char | Charcter |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |

# 왜? Wrapper?

- 컬렉션 저장
  - 자바의 컬렉션(예: ArrayList)은 기본 데이터 타입을 직접 저장할 수 없음
  - 기본 데이터 타입을 저장하고 싶을 때 Wrapper 클래스를 사용
- null 값 허용
  - 기본 데이터 타입은 null 값을 가질 수 없음
  - 그러나 어떤 값이 없거나 알 수 없는 경우를 표현하고 싶을 때, Wrapper 클래스는 null 값을 가질 수 있어 유용
- 메서드와 유틸리티
  - Wrapper 클래스는 문자열 변환, 값 비교와 같은 유용한 메서드들을 제공
- 메서드의 매개변수
  - 메서드에 객체를 매개변수로 전달하거나 반환해야 할 때 Wrapper 클래스가 유용

# 오토박싱과 오토언박싱

- auto-boxing
  - 기본 데이터 타입을 Wrapper 클래스 객체로 자동 변환

- auto-unboxing
  - Wrapper 클래스 *객체를* 기본 데이터 타입으로 자동 변환

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(1);  // 오토박싱으로 int가 Integer로 변환되어 저장
numbers.add(2);
int sum = numbers.get(0) + numbers.get(1);  // 오토언박싱으로 Integer가 int로 변환
System.out.println("합계: " + sum);
```

# 포장 값 비교

- 포장 객체는 값 비교를 위해 ==, != 연산자 사용 불가능
- 왜? 참조값을 비교하기 때문!
- equals() 메소드로 내부 값을 비교

- 단, 포장 객체의 효율적인 사용을 위해 아래 범위의 값을 갖는다면 포장 객체는 공유됨
- 이 외의 값은 내부 값이 아닌 참조값을 비교함을 주의할 것

| 타입 | 값 범위 |
|---|---|
| boolean | true, false |
| char | \u0000 ~ \u007f |
| byte, short, int | -128 ~ 127 |

# 제네릭

# 제네릭이란

- 제네릭은 자바에서 형 안전성(type safety)을 높이기 위해 도입된 프로그래밍
- 제네릭을 사용하면, 컴파일 시간에 타입 오류를 더욱 효과적으로 찾아낼 수 있으며 클래스, 인터페이스, 메서드에 대한 타입을 파라미터로 전달할 수 있게 해 줍니다.

- "결정되지 않은 타입을 파라미터로 처리하고
**실제 사용할 때 파라미터를 구체적인 타입으로 대체**시키는 기능"

```
public class Container <T> {
  public T something;
}
```

Container 클래스에서
something 필드의 타입은 아직 결정되지 않음

```
Container<Integer> c1 = new Container<Integer>;
c1.something = 100;
```

Container 클래스를 사용할 때 구체적인 타입(Integer) 으로 대체

# 제네릭 장점

- 타입 안전성: 잘못된 타입의 객체가 저장되는 것을 컴파일 시간에 방지
- 형 변환 필요성 감소: 제네릭을 사용하면 명시적인 형 변환이 필요 없어짐
- 코드 재사용성: 일반 클래스나 메서드로 다양한 타입에 대해 동작하는 코드를 작성할 수 있음.

# 제네릭 타입

- 결정되지 않은 타입을 파라미터로 가지는 클래스와 인터페이스
- <> 괄호 안에는 일반적으로 대문자와 알파벳으로 작성

```
public class 클래스명<T>
public interface 인터페이스명<E>
```

# 제네릭 클래스 예시 코드

```java
public class MyCustomList {
    ArrayList<String> list = new ArrayList<>();

    public void addElement(String element) {
        list.add(element);
    }

    public void removeElement(String element) {
        list.remove(element);
    }
}
```

일반 클래스

addElement()에는 항상 String만 가능

```java
public class MyCustomList<T> {
    ArrayList<T> list = new ArrayList<>();

    public void addElement(T element) {
        list.add(element);
    }

    public void removeElement(T element) {
        list.remove(element);
    }
}
```

제네릭 클래스

addElement()에는 원하는 타입의 데이터 삽입 가능

# 제네릭 메서드 예시 코드

```
class Utility {
    public static void printPair(String first, String second) {
        System.out.println("(" + first + ", " + second + ")");
    }

    public static void main(String[] args) {
        printPair("apple", "banana");
    }
}
```

일반 메서드

두 개의 문자열을 출력하는 메서드

```
class Utility {
    public static <T> void printPair(T first, T second) {
        System.out.println("(" + first + ", " + second + ")");
    }

    public static void main(String[] args) {
        printPair("apple", "banana");
        printPair(1, 2);
        printPair(1.5, 2.5);
    }
}
```

제네릭 메서드

두 개의 아이템을 출력하는 메서드

# 제한된 타입 파라미터 (bounded type parameter)

- 모든 타입으로 대체할 수 없고, 특정 타입과 자식 or 구현 관계에 있는 타입만 대체할 수 있는 타입 파라미터
- extends 키워드를 제네릭에서 사용하면, 해당 타입 파라미터에 대한 상한을 지정할 수 있음.
- 이를 통해 타입 파라미터가 특정 클래스의 서브 클래스, 또는 특정 인터페이스의 구현 클래스만 가능하도록 제한할 수 있다.

- 사용법
  <T extends 제한 타입>
  ex)  <T extends Number>

# 와일드 카드 (?)

- 제네릭 타입을 매개값 or 리턴 타입으로 사용할 때 타입 파라미터로 ? 사용 가능
- 제네릭에서 와일드카드(?)는 "알 수 없는 타입"을 의미함
- 와일드카드는 제네릭 코드에서 더 큰 유연성을 얻기 위해 사용되며 특히 제네릭 메서드나 제네릭 클래스에서 다양한 제네릭 타입을 처리할 때 유용하게 사용됨

- **?** (Unbounded Wildcard): 어떠한 타입도 될 수 있습니다.
- **? extends T** (Upper Bounded Wildcard): T 타입 또는 T의 서브타입을 의미
- **? super T** (Lower Bounded Wildcard): T 타입 또는 T의 슈퍼타입을 의미합니다.

# 실습. 제네릭 실습

- Pair 클래스는 두 개의 제네릭 타입 K와 V를 가진다.
- Pair 클래스는 두 개의 프라이빗 멤버 변수, key와 value를 갖는다.
- Pair 클래스는 생성자를 통해 key와 value를 초기화
- 각 변수에 대해 getter 메서드를 제공
- 아래와 같이 출력하세요

```
Key: One, Value: 1
Key: 2, Value: Two
```

# 실습. 제네릭 실습2

- Calculator 클래스는 제네릭 타입 T를 사용한다.
- Calculator 클래스는 두 개의 프라이빗 멤버 변수, num1과 num2를 갖는다.
- Calculator 클래스는 생성자를 통해 두 숫자를 초기화
- add 메소드는 두 숫자를 더하는 동작을 하며, 반환값은 double 형이다.
  - 힌트. 제네릭 메소드 .doubleValue() 활용
- main 메소드에서 정수형과 실수형에 대한 Calculator 객체를 생성하고 각각의 결과를 출력한다.

```
Integer Sum: 15.0
Double Sum: 5.640000000000001
```

# Collection

# Collection?

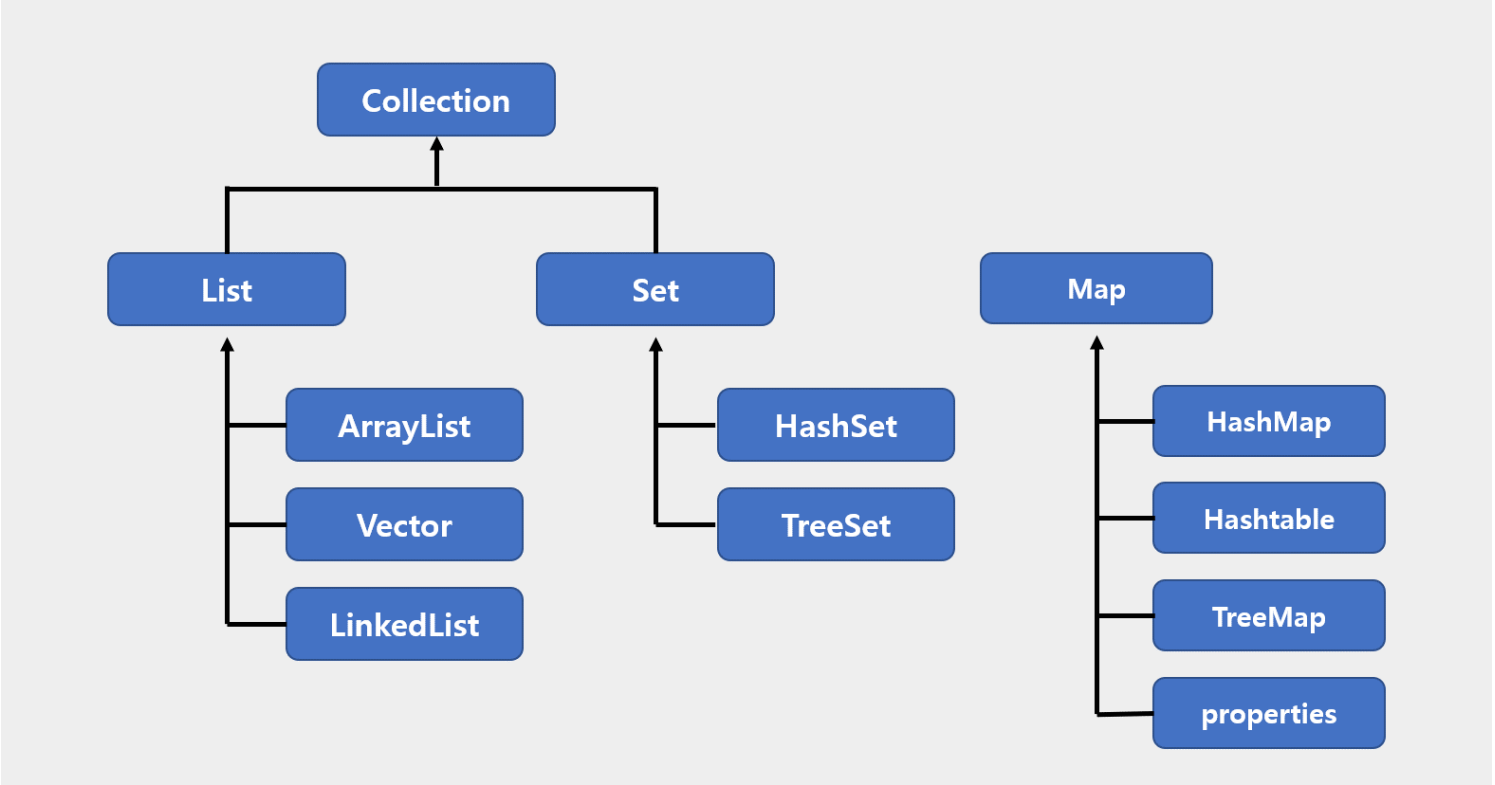
- 데이터 구조와 알고리즘을 제공하는 프레임워크
- 객체의 그룹을 효율적으로 관리하기 위한 다양한 클래스와 인터페이스를 제공
  - 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 인터페이스와 클래스들을 java.util 패키지에 포함

- 컬렉션 프레임워크의 주요 인터페이스
  - Collection: 가장 기본적인 인터페이스로, 모든 컬렉션 클래스가 이를 구현
  - List: 순서가 있는 데이터의 집합을 다룰 때 사용. 데이터 중복을 허용
  - Set: 순서가 없는 데이터의 집합을 다룰 때 사용. 데이터 중복을 허용하지 않음
  - Map: 키와 값의 쌍으로 데이터를 저장합니다. 키는 중복될 수 없다

# Collection Framework 상속 구조

- List, Set
  객체를 추가, 삭제, 검색하는 방법에 공통점이 있어 공통된 메소드만 따로 모아 Collection 인터페이스로 정의해두고 이를 상속 받음

- Map
  키와 값을 하나의 쌍으로 묶어서 관리하는 구조로 List, Set 과 사용법이 다름



# List

- 요소의 순서를 유지하고 저장하며 중복된 요소를 허용하는 컬렉션
- List 컬렉션에서 공통적으로 사용 가능한 List 인터페이스 메소드

| 기능 | 메소드 | 설명 |
|---|---|---|
| 객체 추가 | boolean add(E e) | 주어진 객체를 맨 끝에 추가 |
| | void add(int index, E element) | 주어진 인덱스에 객체를 추가 |
| | set(int index, E element) | 주어진 인덱스의 객체를 새로운 객체로 변경 |
| 객체 검색 | boolean contains(Object o) | 주어진 객체가 저장되었는지 여부 |
| | E get(int index) | 주어진 인덱스에 저장된 객체 리턴 |
| | isEmpty() | 컬렉션이 비어 있는지 여부 |
| | int size() | 컬렉션에 저장된 전체 객체 수 리턴 |
| 객체 삭제 | void clear() | 저장된 모든 객체 삭제 |
| | E remove(int index) | 주어진 인덱스에 저장된 객체 삭제 |
| | boolean remove(Object o) | 주어진 객체를 삭제 |

# List

## ArrayList

- ArrayList는 List 인터페이스의 동적 배열 구현
- 초기 크기가 있지만, 요소가 추가됨에 따라 자동으로 크기가 확장됨
- 배열 기반이므로 인덱스를 사용한 요소 접근이 빠르지만, 중간에 요소를 삽입하거나 삭제하는 연산은 느린 편이다

## LinkedList

- LinkedList는 List와 Deque 인터페이스의 양방향 연결 리스트 구현
- 연결 리스트 기반이므로 중간에 요소를 삽입하거나 삭제하는 연산이 빠르지만, 인덱스를 사용한 요소 접근은 느린 편이다

# Set

- List와 달리 중복된 요소를 저장할 수 없으며 순서를 보장하지 않는 컬렉션
- Set 컬렉션에서 공통적으로 사용 가능한 Set 인터페이스 메소드

| 기능 | 메소드 | 설명 |
|---|---|---|
| 객체 추가 | boolean add(E e) | 주어진 객체를 저장 (중복된 요소 없다면 true, 중복된 요소 있으면 false 반환) |
| 객체 검색 | boolean contains(Object o) | 주어진 객체가 저장되었는지 여부 |
| | isEmpty() | 컬렉션이 비어 있는지 여부 |
| | Iterator<E> iterator () | 저장된 객체를 한 번 씩 가져오는 반복자 리턴 |
| | int size() | 컬렉션에 저장된 전체 객체 수 리턴 |
| 객체 삭제 | void clear() | 저장된 모든 객체 삭제 |
| | boolean remove(Object o) | 주어진 객체를 삭제 |

# Set

## HashSet

- Set의 대표적인 클래스로, 해시 테이블을 사용하여 요소를 저장
- 순서를 보장하지 않으며, 동일한 객체는 중복 저장하지 않음

## LinkedHashSet

- LinkedHashSet은 HashSet을 확장하여 요소의 삽입 순서를 기억
- 이로 인해 요소의 삽입 순서대로 반복

## TreeSet

- 자동으로 정렬된 순서로 요소를 저장하며, 사용자 정의 정렬도 가능합니다.

# Map

- Map은 키와 값으로 구성된 엔트리(Entry) 객체를 저장하는 데이터 구조
- 각 키는 고유해야 하며, 값은 중복될 수 있음
- Map 컬렉션에서 공통적으로 사용 가능한 Map 인터페이스

| 기능 | 메소드 | 설명 |
|---|---|---|
| 객체 추가 | V put(K key, V value) | 주어진 키와 값을 추가, 저장이 되면 값을 리턴 |
| 객체 검색 | boolean containsKey(Object key) | 주어진 키가 있는지 여부 |
| | boolean containsValue(Object value) | 주어진 값이 있는지 여부 |
| | Set<Map.Entry<K, V>> entrySet() | 키와 값을 쌍으로 구성된 모든 Map.Entry 객체를 Set에 담아 리턴 |
| | V get(Object key) | 주어진 키의 값을 리턴 |
| | boolean isEmpty() | 컬렉션이 비어 있는지 여부 |
| | Set<K> keySet() | 모든 키를 Set 객체에 담아서 리턴 |
| | int size() | 저장된 키의 총 개수를 리턴 |
| | Collection<V> values() | 저장된 모든 값 Collection에 담아서 리턴 |
| 객체 삭제 | void clear() | 모든 Map.Entry(키와 값)를 삭제 |
| | V remove(Object key) | 주어진 키와 일치하는 Map.Entry 삭제, 삭제 되면 값을 리턴 |

# Map

## HashMap

- 해시 테이블을 사용하여 키-값 쌍을 저장
- 순서를 보장하지 않음

## LinkedHashMap

- LinkedHashMap은 HashMap을 확장하여 키-값 쌍의 삽입 순서나 접근 순서를 기억
- 순서가 중요한 경우에 유용

## TreeMap

- 키에 따라 자동으로 정렬됩니다.

# 실습. 콜렉션 연습문제

- 사용자로부터 정수를 반복적으로 입력 받습니다.
- 사용자가 -1을 입력하면 입력을 종료합니다.
- 모든 입력된 정수 중 중복을 제거하고 정수만 출력합니다.
- 조건:
  - 중복 체크를 위해 HashSet을 사용해야 합니다.

```
정수를 입력하세요. -1을 입력하면 종료됩니다.
정수 입력: 10
정수 입력: 10
정수 입력: 14
정수 입력: 2
정수 입력: 8
정수 입력: -1
중복 제거된 정수 목록: [2, 8, 10, 14]
```

# 실습2. 콜렉션 연습문제

- 프로그램은 사용자로부터 반복적으로 이름과 나이를 입력 받습니다. 이 때, 사용자가 "종료"라는 이름을 입력하면 입력을 종료하고, 입력 받은 모든 이름과 나이를 출력
- 만약 동일한 이름이 두 번 이상 입력될 경우, 가장 마지막에 입력된 나이로 이름의 나이를 갱신해야 합니다.
- 조건:
  - 이름과 나이 관리를 위해 Map<String, Integer>을 사용해야 합니다.
- 힌트
  - Map.Entry는 Java의 Map 인터페이스 내부에 정의된 인터페이스로, 맵에 저장된 키-값 쌍을 나타냄

```
이름과 나이를 입력하세요. '종료'를 입력하면 종료됩니다.
이름 입력: 김새싹
나이 입력: 20
이름 입력: 홍길동
나이 입력: 21
이름 입력: 종료

== 입력 받은 이름과 나이 목록 ==
이름: 홍길동, 나이: 21
이름: 김새싹, 나이: 20
```