SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

Java

SAMSUNG SW ACADEMY FOR YOUTH

Programming Language

객체지향 프로그래밍

- Collection Framework

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

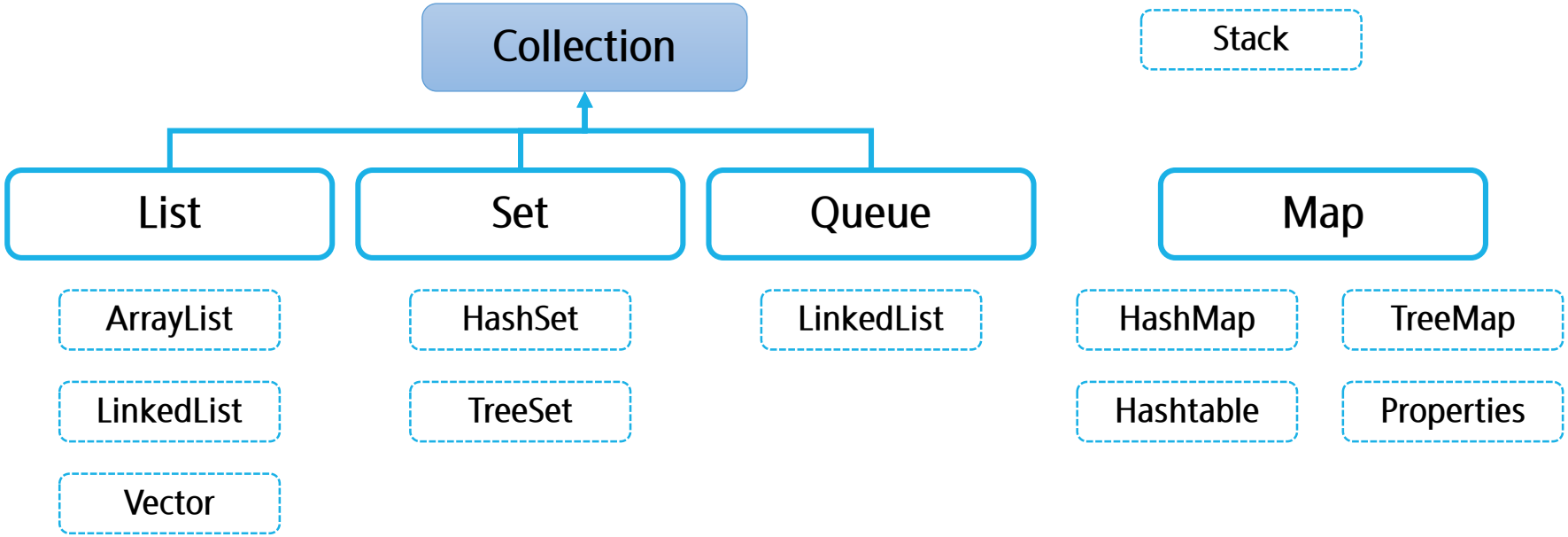
Collection Framework

Collection Framework

- 객체들을 한곳에 모아 놓고 편리하게 사용할 수 있는 환경을 제공
- 정적 자료구조 (Static data structure)
  - 고정된 크기의 자료구조
  - 배열이 대표적인 정적 자료구조
  - 선언 시 크기를 명시하면 바꿀 수 없음
- 동적 자료구조 (Dynamic data structure)
  - 요소의 개수에 따라 자료구조의 크기가 동적으로 증가하거나 감소
  - 리스트, 스택, 큐 등

Collection Framework

- 자료구조들의 종류는 결국은 어떤 구조에서 얼마나 빨리 원하는 데이터를 찾는가에 따라 결정된다.
  - 순서를 유지할 것인가?
  - 중복을 허용할 것인가?
  - 다른 자료구조들에 비해서 어떤 단점과 장점을 가지고 있는가?



Collection Framework

- java.util 패키지
  - 다수의 데이터를 쉽게 처리하는 방법 제공
- Collection Framework 핵심 interface

interface	특징
List	순서가 있는 데이터의 집합. 순서가 있으니까 데이터의 중복을 허락 ex) 일렬로 줄 서기 ArrayList, LinkedList, Vector
Set	순서를 유지하지 않는 데이터의 집합. 순서가 없어서 같은 데이터를 구별할 수 없음 → 중복 허락 하지 않음 ex) 알파벳이 한 종류 씩 있는 주머니 HashSet, TreeSet…
Map	key와 value의 쌍으로 데이터를 관리하는 집합. 순서는 없고 key의 중복 불가, value는 중복 가능 ex) 속성 – 값, 지역번호-지역 HashMap, TreeMap
Queue	지하철, 버스를 탈 때 대기줄과 같이 들어온 순서대로 나가는 자료구조 LinkedList

Collection interface

분류	Collection
추가	add(E e), addAll(Collection<? extends E> c)
조회	contains(Object o), containsAll(Collection<?> c), equals(), isEmpty(), iterator(), size(),
삭제	clear(), removeAll(Collection<?> c) retainAll(Collection<?> c),
수정	
기타	toArray()

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

List

Collection Framework - List

- 특징 : 순서가 있고, 중복을 허용 (배열과 유사)
- 구현 클래스
  - ArrayList
  - LinkedList
  - Vector

Collection Framework - List

- 내부적으로 배열을 이용하여 데이터를 관리
- 배열과 다르게 크기가 유동적으로 변함 (동적 자료구조)
- 배열을 다루는 것과 유사하게 사용 할 수 있음
- 주요 메서드

분류	Collection	List
추가	add(E e), addAll(Collection<? extends E> c)	add(int index, E element), addAll(int index, Collection<? extends E> c)
조회	contains(Object o), containsAll(Collection<?> c), equals(), isEmpty(), iterator(), size(),	 get(int index), indexOf(Object o), lastIndexOf(Object o), listIterator(),
삭제	clear(), removeAll(Collection<?> c) retainAll(Collection<?> c),	remove(int index)
수정		set(int index, E element)
기타	toArray()	subList(int fromIndex, int toIndex)

주요 메서드

```
// 문자열을 저장할 List, 구현체는 ArrayList
List<String> names = new ArrayList<>();

//추가
names.add("양만춘");
names.add("홍길동");
names.add("양만춘");
names.add("이순신");
names.add(0, "임꺽정");

System.out.println(names);
```

배열의 크기는 어떻게 되는 걸까?

구현 코드

Constructor

```
public ArrayList(int initialCapacity) {
  if (initialCapacity > 0) {
    this.elementData = new Object[initialCapacity];
  } else if (initialCapacity == 0) {
    this.elementData = EMPTY_ELEMENTDATA;
  } else {
    throw new IllegalArgumentException("Illegal Capacity: "+ initialCapacity);
  }
}
public ArrayList() {
  this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA; // {}
}
```

내부적으로 Object []에 저장

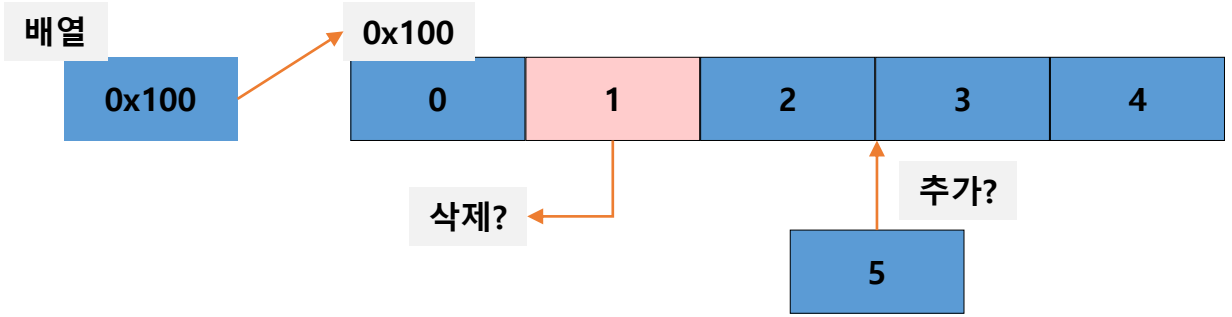
add → ensureCapacityInternal → ensureExplicitCapacity → grow

```
private void grow(int minCapacity) {
  // overflow-conscious code
  int oldCapacity = elementData.length;
  int newCapacity = oldCapacity + (oldCapacity >> 1);
  if (newCapacity - minCapacity < 0)
    newCapacity = minCapacity;
  if (newCapacity - MAX_ARRAY_SIZE > 0)
    newCapacity = hugeCapacity(minCapacity);
  // minCapacity is usually close to size, so this is a win:
  elementData = Arrays.copyOf(elementData, newCapacity);
}
```

칸이 부족하면 알아서 늘여주고

배열과 ArrayList

- 배열의 장점
  - 가장 기본적인 형태의 자료 구조, 간단하며 사용이 쉬움
  - 접근 속도가 빠름
- 배열의 단점
  - 크기를 변경할 수 없어 추가 데이터를 위해 새로운 배열을 만들고 복사 해야함.
  - 비 순차적 데이터의 추가, 삭제에 많은 시간이 걸림



- 배열을 사용하는 ArrayList도 태생적으로 배열의 장/단점을 그대로 가져감

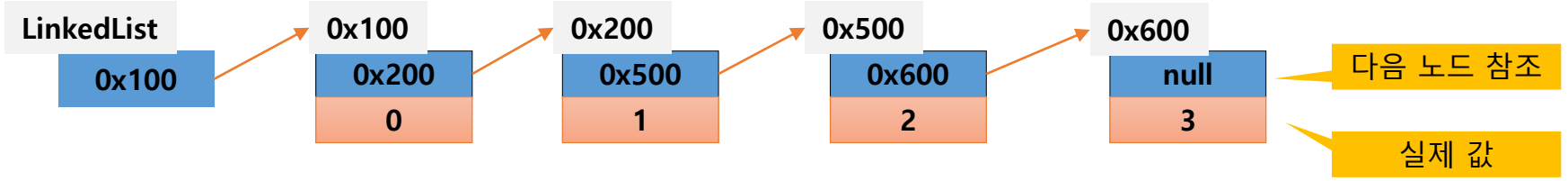
Collection Framework - ArrayList

- add(E e) : 데이터 입력
- get(int index) : 데이터 추출
- size() : 입력된 데이터의 크기 반환
- remove(int i) : 특정한 데이터를 삭제
- remove(Object o) : 특정한 데이터를 삭제
- clear() : 모든 데이터 삭제
- contains(Object o) : 특정 객체가 포함되어 있는지 체크
- isEmpty() : 비어있는지 체크(true, false)
- addAll(Collection c) : 기존 등록된 콜렉션 데이터 입력
- iterator() : iterator 인터페이스 객체 반환

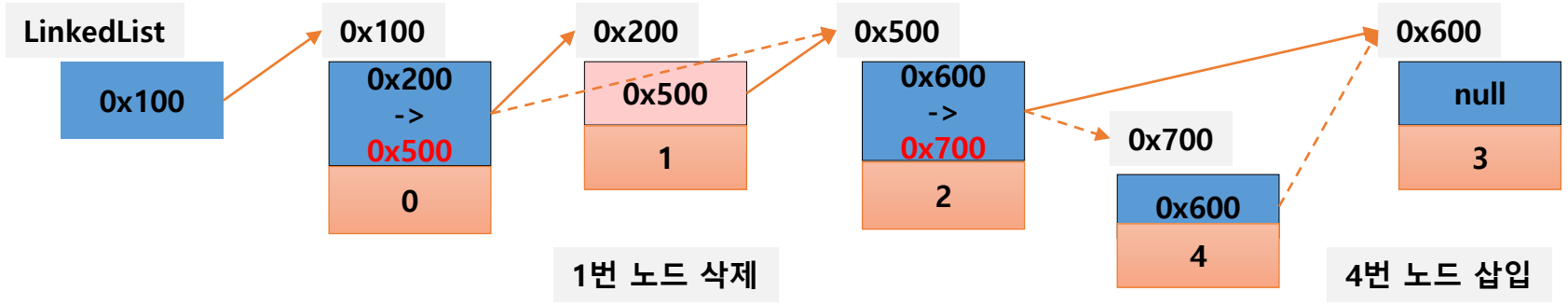
Collection Framework - 참고

LinkedList

- 각 요소를 Node로 정의하고 Node는 다음 요소의 참조 값과 데이터로 구성됨
- 각 요소가 다음 요소의 링크 정보를 가지며 연속적으로 구성될 필요가 없음



- 데이터 삭제 및 추가



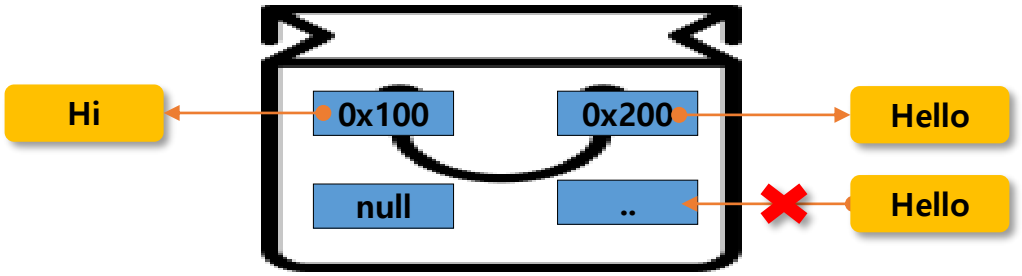
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Set & Map

Collection Framework - Set

- 특징 : 순서가 없고, 중복을 허용하지 않음
- 장점 : 빠른 속도, 효율적인 중복 데이터 제거 수단
- 단점 : 단순 집합의 개념으로 정렬하려면 별도의 처리가 필요하다.
- 구현 클래스
  - HashSet
  - TreeSet

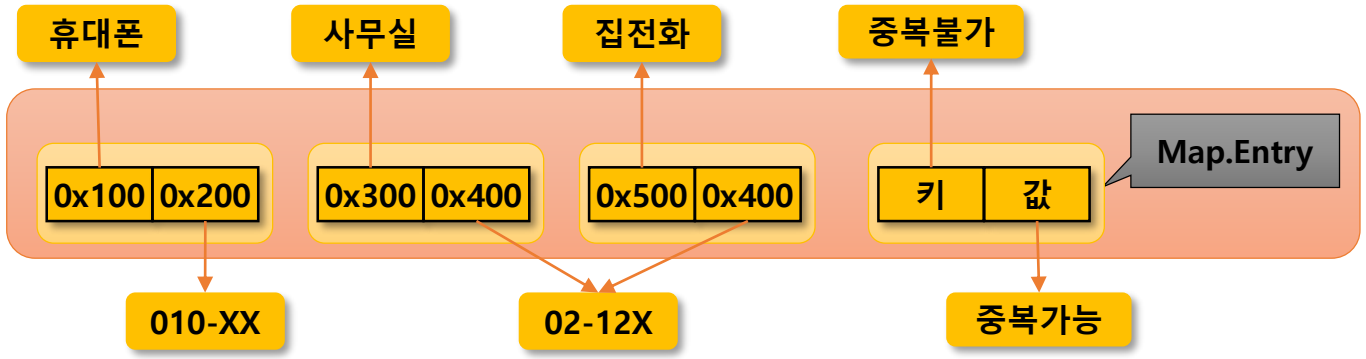


Collection Framework - Set

- add(E e) : 데이터 입력
- size() : 입력된 데이터의 크기 반환
- remove(Object o) : 특정한 데이터를 삭제
- clear() : 모든 데이터 삭제
- contains(Object o) : 특정 객체가 포함되어 있는지 체크
- isEmpty() : 비어있는지 체크 (true, false)
- iterator() : iterator 인터페이스 객체 반환
- toArray() : Set의 내용을 Object 형의 배열로 변환

Collection Framework - Map

- 특징 : Key(키)와 value(값)를 하나의 Entry로 묶어서 데이터 관리,
  순서는 없으며, 키에 대한 중복은 없음
- 장점 : 빠른 속도
- 구현 클래스
  - HashMap
  - TreeMap



Collection Framework - Map

- V put(K key, V value) : 데이터 입력
- V get(Object key) : 데이터 추출
- V remove(K key) : 키의 값을 지우고 반환 , 없다면 null을 반환
- boolean containsKey(Object key) : 특정한 key 포함 여부
- void putAll(Map<K key, V value> m) : 기존 콜렉션 데이터 추가
- Set<Map.Entry<K, V>> entrySet() :
  - (key 와 value) 쌍을 표현하는 Map.Entry 집합을 반환

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Queue & Stack

Collection Framework - Queue

- Queue는 인터페이스, 구현체는 LinkedList를 사용
- 큐 자료구조: FIFO,(first-in-first-out) 가장 먼저 들어온 값이 가장 먼저 빠져나감
- boolean offer(E e): 데이터를 추가
- E peek(): 가장 앞에 있는 데이터 조회
- E poll(): 가장 앞에 있는 데이터 빼내기
- boolean isEmpty() : 큐가 비어있는지 여부

Collection Framework - Stack

- Stack 클래스를 사용
- 스택 자료구조: LIFO,(last-in-first-out) 가장 나중에 들어온 값이 가장 먼저 빠져나감
- E push(E e): 데이터를 추가
- E peek(): 가장 위에 있는 데이터 조회
- E pop(): 가장 위에 있는 데이터 빼내기
- boolean isEmpty() : 스택이 비어있는지 여부

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

정렬

정렬

- 요소를 특정 기준에 대한 내림차순 또는 오름차순으로 배치 하는 것
- 순서를 가지는 Collection들만 정렬 가능
  - List 계열
  - Set에서는 SortedSet의 자식 객체
  - Map에서는 SortedMap의 자식 객체(Key 기준)

- Collections의 sort()를 이용한 정렬
  - sort(List<T> list)
  - 객체가 Comparable을 구현하고 있는 경우 내장 알고리즘을 통해 정렬

```
private List<String> names = Arrays.asList("Hi", "Java", "World", "Welcome");

public void basicSort() {
   Collections.sort(names);
   System.out.println(names); // [Hi, Java, Welcome, World]
}
```

Comparable interface

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

양수: 자리 바꿈
음수: 자리 유지
0: 동일 위치

Comparator의 활용

- 객체가 Comparable을 구현하고 있지 않거나 사용자 정의 알고리즘으로 정렬하려는 경우
  - String을 알파벳 순이 아니라 글자 수 별로 정렬을 하고 싶다.
- sort(List<T> list, Comparator<? Super T> c)

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

```
public class StringLengthComparator implements Comparator<String> {
  @Override
  public int compare(String o1, String o2) {
    int len1 = o1.length();
    int len2 = o2.length();
    return Integer.compare(len1, len2);
  }
}
```

```
public void stringLengthSort() {
    Collections.sort(names, new StringLengthComparator());
    System.out.println(names);  // [Hi, Java, World, Welcome]
}
```

Comparator의 활용

- 1회성 객체 사용 시 anonymous inner class 사용
- 클래스 정의, 객체 생성을 한번에 처리

```
// before
Collections.sort(names, new StringLengthComparator());

// after
Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(String o1, String o2) {
        return Integer.compare(o1.length(), o2.length());
    }
});
```

- 람다 표현식 사용

```
Collections.sort(names, (o1, o2)->{
    return Integer.compare(o1.length(), o2.length());
});
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미