Collection<E> → Set<E> interface I

Collection<E> → Set<E> 인터페이스를 구현하는 컬랙션즈 클래스

Set<E> Inteface

- 인스턴스를 저장하고 **집합을 표현**하는 컨테이너
- **인스턴스의 중복저장을 허용하지 않는다.**
- 인스턴스 저장순서의 개념이 없다.

- Set<E>의 대표적인 메소드

add(E e)	인자로 전달된 인스턴스 데이터 저장
remove(int idx)	인자로 전달된 인덱스값에 위치한 인스턴스 삭제
int size()	List<E>에 저장된 인스턴스 개수 반환
boolean ContainsAll(E e)	부분집합
boolean addAll(E e)	합집합
boolean retainAll(E e)	교집합
boolean removeAll(E e)	차집합

i. HashSet<E>

- 집합의 특성
- 해시 자료구조를 기반으로 데이터 관리
- 전달되는 인스턴스 내의 hashCode(), equals() Method를 이용한 빠른 검색속도를 자랑한다.
- 인스턴스 저장순서의 개념이 없다.

```
import java.util.HashSet;
import java.util.Iterator;

public class HashSetDemo {
     public static void main(String[] args) {
          HashSet<String> hSet = new HashSet<String>();
          hSet.add("first");
          hSet.add("second");
          hSet.add("third");
          hSet.add("first"); // 중복저장을 허용하지 않는다!

          System.out.println("저장된 데이터의 수 : "+hSet.size());

          Iterator<String> iterator = hSet.iterator();
          while(iterator.hasNext())
               System.out.println(iterator.next());
     }
}
```

```
Command Prompt

저장된 데이터의 수 : 3
third
first
second
-----------------------------------------------------------------------------------------
```

- **HashSet<E>의 동일 인스턴스 판단기준 관찰**

```
public class SimpleNumber {
     int num;

     public SimpleNumber(int num){ this.num = num; }

     @Override
     public String toString(){ return String.valueOf(num); }
}

public class HashSetDemo {
     public static void main(String[] args) {
          HashSet<SimpleNumber> hSet = new HashSet<SimpleNumber>();
          hSet.add(new SimpleNumber(10));
          hSet.add(new SimpleNumber(20));
          hSet.add(new SimpleNumber(20));

          System.out.println("저장된 데이터의 수 : "+hSet.size());

          Iterator<SimpleNumber> iterator = hSet.iterator();
          while(iterator.hasNext())
               System.out.println(iterator.next());
     }
}
-----------------------------------------------------------------------------------------
Command Prompt

저장된 데이터의 수 : 3
20
10
20
-----------------------------------------------------------------------------------------
```

  - **10,20,20 순으로 저장했으나. Set은 순서개념이 없다.**
  - **인스턴스 멤버(num)에 저장된 값이 같다고 해서 중복저장으로 보지 않는다.**

HashSet<E>의 인스턴스 중복저장을 막기 위한 동등비교 기준 정의하기

Object Class에는 equals(), hashCode() Method가 있는데,
인스턴스의 동등비교는 Set에 전달되는 인스턴스 내의 equals(), hashCode() Method로
동등비교 진행하고 중복저장을막기 때문에 Overriding 해서 직접 정의해주어야 한다.

- **equals() Method**

- 두 인스턴스가 같은 주소값을 참조하면 true, 주소값이 서로 다르면 false 반환
- 동등에 대한 기준결정 역할

- **hashCode() Method**
  - 해시 알고리즘이 적용된 메소드
  - hashCode() Method는 hash 알고리즘을 반환하는데, hash 알고리즘을 이용해 HashSet<E>에 전달된 인스턴스가 어느 집합에 속하는지 판별 후, 집합 내의 인스턴스들과 equals() Method로 비교를 하게된다.

해시 알고리즘의 소개 (데이터의 구분)

Data	3, 5, 7, 12, 25, 31

해시 알고리즘	num % 3	해시 알고리즘은 직접 정의 할 수 있다.

해시 알고리즘을 적용한 데이터집합

- 해시 알고리즘의 경우 두 데이터가 같으면 같은 집합에 속하게 된다.
- **집합으로 분류했기 때문에, 특정 데이터를 해시 알고리즘 연산을 하고 결과에 맞는 집합에 찾아가서 특정 데이터가 있는지 찾기만 하면 된다. (검색 시간의 단축)**

Gruop A	나머지 0	3, 12
Gruop B	나머지 1	25, 7, 31
Gruop C	나머지 2	5

HashSet<E> 컬렉션즈 클래스의 활용 예

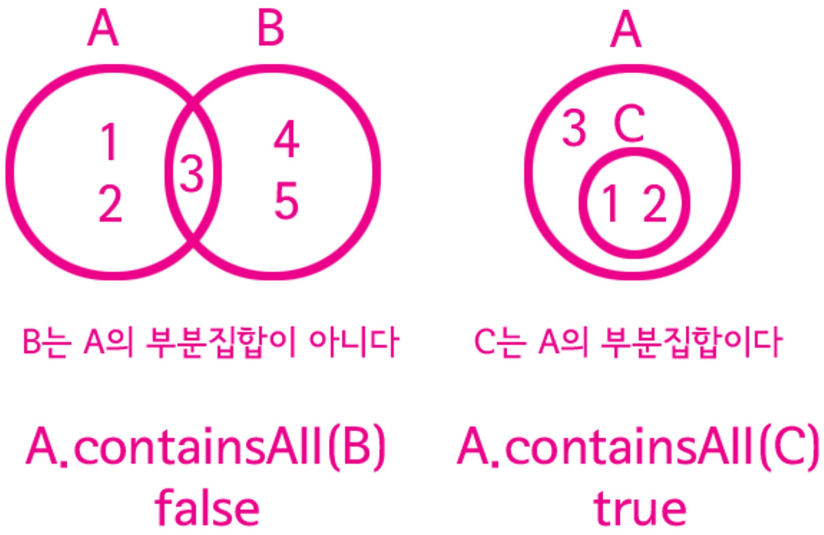
```
public class SimpleNumber {
	int num;

	public SimpleNumber(int num){ this.num = num; }

	@Override
	public String toString(){ return String.valueOf(num); }

	@Override
	public int hashCode() {
		return num % 3; // 해시 알고리즘 반환, 해시집합은 세 부류로 나뉜다.
	}

	// 인스턴스의 멤버 num이 같을 때, 동일 인스턴스로 간주하겠다.
	@Override
	public boolean equals(Object obj) {
		SimpleNumber comp = (SimpleNumber)obj;

		if(comp.num == num){
			return true;
		} else
			return false;
	}
}
```

```
public class HashSetDemo {
     public static void main(String[] args) {
          HashSet<SimpleNumber> hSet = new HashSet<SimpleNumber>();
          hSet.add(new SimpleNumber(10));
          hSet.add(new SimpleNumber(20));
          hSet.add(new SimpleNumber(20));

          System.out.println("저장된 데이터의 수 : "+hSet.size());

          Iterator<SimpleNumber> iterator = hSet.iterator();
          while(iterator.hasNext())
               System.out.println(iterator.next());
     }
}
```

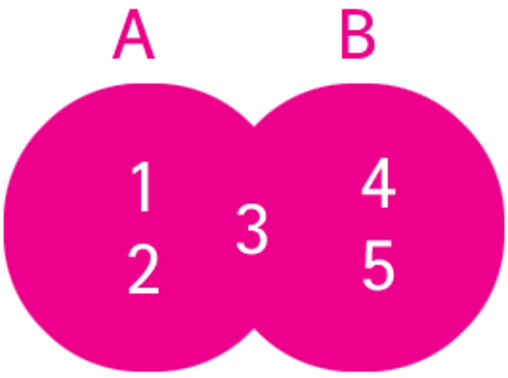
```
Command Prompt

저장된 데이터의 수 : 2
20
10
```

Set<E> interface의 집합연산 메소드

- **부분집합 (containsAll)**



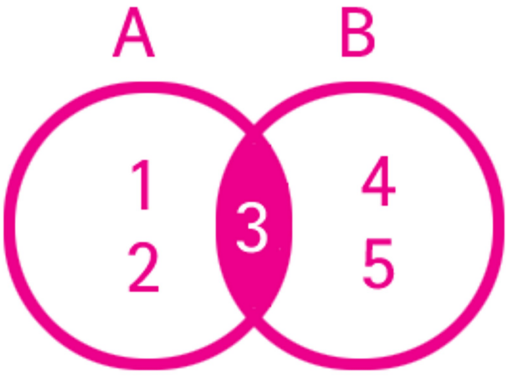
- **합집합 (addAll)**



A와 B의 합집합

A.addAll(B)

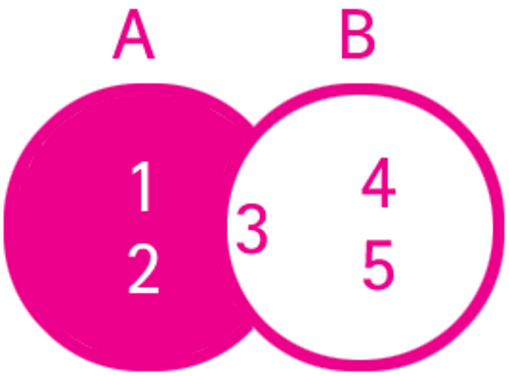
- **교집합 (retainAll)**



A와 B의 교집합

A.retainAll(B)

- **차집합 (removeAll)**



A에서 B를 뺀 차집합

A.removeAll(B)