

10강

# JAVA\_PROGRAMMING



# Collection Framework

## ❖ Collection

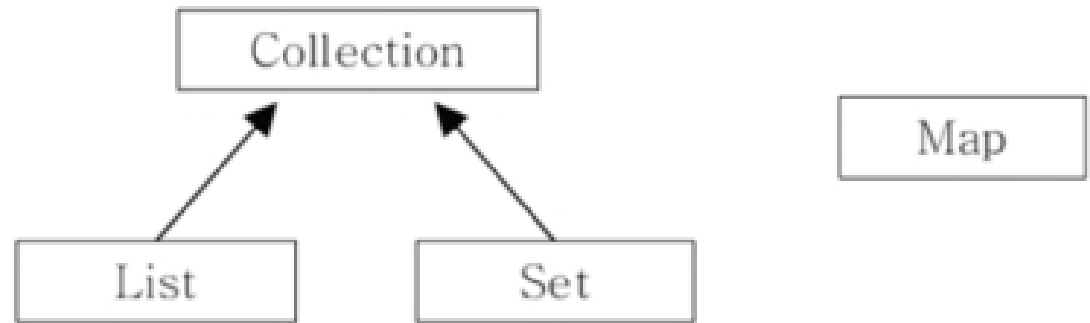
- 무한한 데이터의 집합

## ❖ Framework

- 정형화된 방식

## ❖ Collection Framework

- **무한한 데이터를 저장**하기 위해 사용하는 정형화된 형식 클래스



Set	비순차적 저장 , 중복 데이터 불허
Map	비순차적 저장, 중복 데이터 허용
List	순차적 저장, 중복 데이터 허용

# Set

- ❖ Set인터페이스를 구현한 컬렉션 클래스
- ❖ 순서가 없고 중복된 데이터를 허용하지 않는 방식
  - HashSet
  - TreeSet
  - LinkedHashSet(순서 유지 기능 존재)

## 예제(HashSet)

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

public class TestSet {
    public static void main(String[] args) {
        HashSet hs = new HashSet();
        hs.add(new Integer(10));
        hs.add(new Integer(30));
        hs.add(new String("Hello"));
        hs.add(new Integer(10));

        Iterator it = hs.iterator();

        while(it.hasNext())
            System.out.println(it.next());
    }
}
```

<실행결과>

Hello

10

30

# Enumeration, Iterator, ListIterator

## ❖ Enumeration

- 컬렉션 객체에 저장된 데이터에 접근하는 방법을 제공하는 클래스

메서드	설 명
<code>boolean hasMoreElements()</code>	읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
<code>Object nextElement()</code>	다음 요소를 읽어 온다. <code>nextElement()</code> 를 호출하기 전에 <code>hasMoreElements()</code> 를 호출해서 읽어올 요소가 남아있는지 확인하는 것이 안전하다.

## ❖ Enumeration → Iterator → ListIterator(양방향 순회 추가) 순으로 발전

# Iterator

## ❖ 반복자(Iterator)

- 컬렉션 내부의 요소에 접근하여 순환할 수 있는 기능을 제공
- 컬렉션을 구현한 클래스 객체로부터 iterator()메서드로 얻어옴

메서드	설 명
boolean hasNext()	읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next()	다음 요소를 읽어 온다. next()를 호출하기 전에 hasNext()를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
void remove()	next()로 읽어 온 요소를 삭제한다. next()를 호출한 다음에 remove()를 호출해야한다.(선택적 기능)

```
Iterator it = hs.iterator();
```

```
while(it.hasNext())  
    System.out.println(it.next());
```

## 예제(Generic)

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

public class TestSet {
    public static void main(String[] args) {
        HashSet<Integer> hs = new HashSet<Integer>();
        hs.add(new Integer(10));
        hs.add(new Integer(30));
        //hs.add(new String("Hello")); //저장 불가
        hs.add(new Integer(10));

        Iterator<Integer> it = hs.iterator();

        while(it.hasNext())
            System.out.println(it.next());
    }
}
```

<실행결과>

10

30

# Generic

## ❖ Generic

- jdk1.5 버전부터 추가된 기능
- 컬렉션에 저장할 데이터 타입을 지정하는 기능
- 서로 다른 자료들이 하나의 컬렉션에 저장되는 혼란을 방지

- 적용 전(Object 형태로 모든 객체 저장 가능)

```
HashSet hs = new HashSet();  
hs.add(new Integer(10));  
hs.add(new Integer(30));  
hs.add(new String("Hello"));
```

- 적용 후(Generic으로 지정된 Integer만 저장 가능)

```
HashSet<Integer> hs = new HashSet<Integer>();  
hs.add(new Integer(10));  
hs.add(new Integer(30));  
//hs.add(new String("Hello")); //저장 불가
```



# 실습

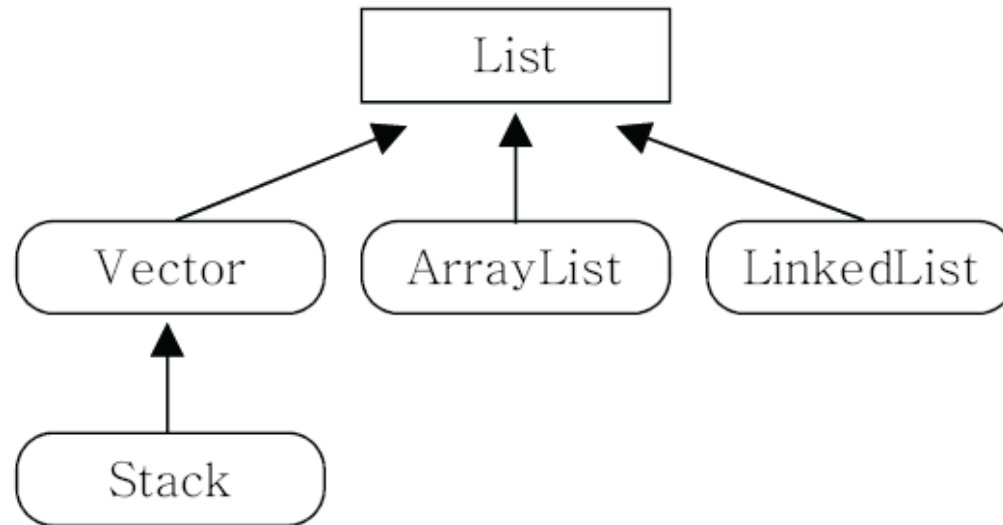
## ❖ Java API Document에서 Set 인터페이스를 찾아 Method 확인

- Set을 구현한 클래스를 이용하여 다음 클래스 테스트
- 임의의 데이터용 객체를 생성하고 아래 클래스로 컬렉션 객체 생성 후 관리 테스트
  - TreeSet
  - LinkedHashSet(순서 유지 기능 존재)

# List

❖ 순서가 있고 데이터의 중복을 허용하는 방식(순차 저장)

- ArrayList(동기화 적용 안 됨)
- Vector(자체 동기화 적용)
- LinkedList
- Stack



# 예제(ArrayList)

```
import java.util.ArrayList;

public class TestList{
    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        al.add(new Integer(10));
        al.add("hello");
        al.add(new Integer(20));
        al.add(new Integer(10));
        al.add(new Float(3.14));

        for(Object tmp : al){
            System.out.println("forEach:" + tmp);
        }
        for(int i = 0; i < al.size(); i++){
            System.out.println("for(index):" + al.get(i));
        }
    }
}
```

## <실행결과>

```
forEach:10
forEach:hello
forEach:20
forEach:10
forEach:3.14
for(index):10
for(index):hello
for(index):20
for(index):10
for(index):3.14
```

# 예제(Generic)

```
import java.util.ArrayList;

public class TestList{
    public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(new Integer(10));
        //al.add("hello");
        al.add(new Integer(20));
        al.add(new Integer(10));
        //al.add(new Float(3.14));

        for(Integer tmp : al){
            System.out.println("forEach:" + tmp);
        }
        for(int i = 0; i < al.size(); i++){
            System.out.println("for(index):" + al.get(i));
        }
    }
}
```

<실행결과>

```
forEach:10
forEach:20
forEach:10
for(index):10
for(index):20
for(index):10
```

# 실습

## ❖ Java API Document에서 List 인터페이스를 찾아 Method 확인

- List를 구현한 클래스를 이용하여 다음 클래스 테스트
- 임의의 데이터용 객체를 생성하고 아래 클래스로 컬렉션 객체 생성 후 관리 테스트
  - ArrayList
  - LinkedList
  - Vector
  - Stack

# Map

- ❖ 키(Key)와 값(Value)을 mapping 시켜 데이터를 저장하는 방식
- ❖ 순서가 없으며 키의 중복은 허용하지 않고 값의 중복은 허용
  - HashMap
  - TreeMap
  - Hashtable

```
HashMap map = new HashMap();  
map.put("castello", "1234");  
map.put("asdf", "1111");  
map.put("asdf", "1234");
```

키(key) - 컬렉션 내의 키(key) 중에서 유일해야 한다.  
값(value) - 키(key)와 달리 데이터의 중복을 허용한다.

키(key)	값(value)
castello	1234
asdf	1234

# 예제(Hashtable)

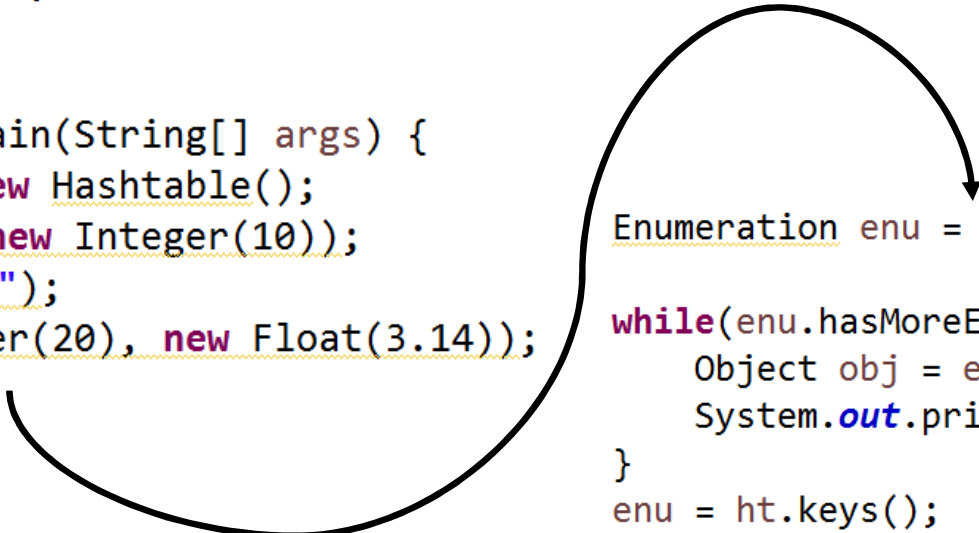
- ❖ Map을 구현한 컬렉션 클래스
- ❖ 자체적으로 동기화 처리가 되어 있다.

```
import java.util.Enumeration;
import java.util.Hashtable;

public class TestMap {
    public static void main(String[] args) {
        Hashtable ht = new Hashtable();
        ht.put("hello", new Integer(10));
        ht.put("바늘", "실");
        ht.put(new Integer(20), new Float(3.14));
    }
}
```

<실행결과>

```
obj = 3.14
obj = 10
obj = 실
3.14
10
실
```



```
Enumeration enu = ht.elements();

while(enu.hasMoreElements()) {
    Object obj = enu.nextElement();
    System.out.println("obj = " + obj);
}

enu = ht.keys();
while(enu.hasMoreElements()) {
    Object obj = ht.get(enu.nextElement());
    System.out.println(obj);
}
```

# 예제(HashMap)

- ❖ Map을 구현한 컬렉션 클래스

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class TestMap {
    public static void main(String[] args) {
        HashMap hm = new HashMap();
        hm.put("hello", new Integer(10));
        hm.put("바늘", "실");
        hm.put(new Integer(20), new Float(3.14));

        Set keys = hm.keySet();
        Iterator it = keys.iterator();
        while(it.hasNext()) {
            Object key = it.next();
            Object value = hm.get(key);
            System.out.println(key + " -> " + value);
        }
    }
}
```

<실행결과>

바늘 -> 실

20 -> 3.14

hello -> 10



## 예제(Generic)

❖ Map에 저장될 키와 값을 지정한다.

```
HashMap<String, Integer> hm = new HashMap<String, Integer>();  
hm.put("hello", new Integer(10));  
//hm.put("바늘", "실");  
//hm.put(new Integer(20), new Float(3.14));
```

# 실습

- ❖ Java API Document에서 Map 인터페이스를 찾아 Method 확인
  - Map을 구현한 클래스를 이용하여 다음 클래스 테스트
  - 임의의 데이터용 객체를 생성하고 아래 클래스로 컬렉션 객체 생성 후 관리 테스트
    - TreeMap

# 예제(Properties)

- ❖ 내부적으로 Hashtable을 사용(키와 값을 <String, String> 형식으로 저장)
- ❖ 어플리케이션의 속성을 지정하는 용도로 사용(파일을 사용하는 기능 제공)

```
import java.util.*;
import java.io.*;

class PropertiesEx3
{
    public static void main(String[] args)
    {
        Properties prop = new Properties();

        prop.setProperty("timeout", "30");
        prop.setProperty("language", "한글");
        prop.setProperty("size", "10");
        prop.setProperty("capacity", "10");

        try {
            prop.store(new FileOutputStream("output.txt"), "Properties Example");
            prop.storeToXML(new FileOutputStream("output.xml"), "Properties Example");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

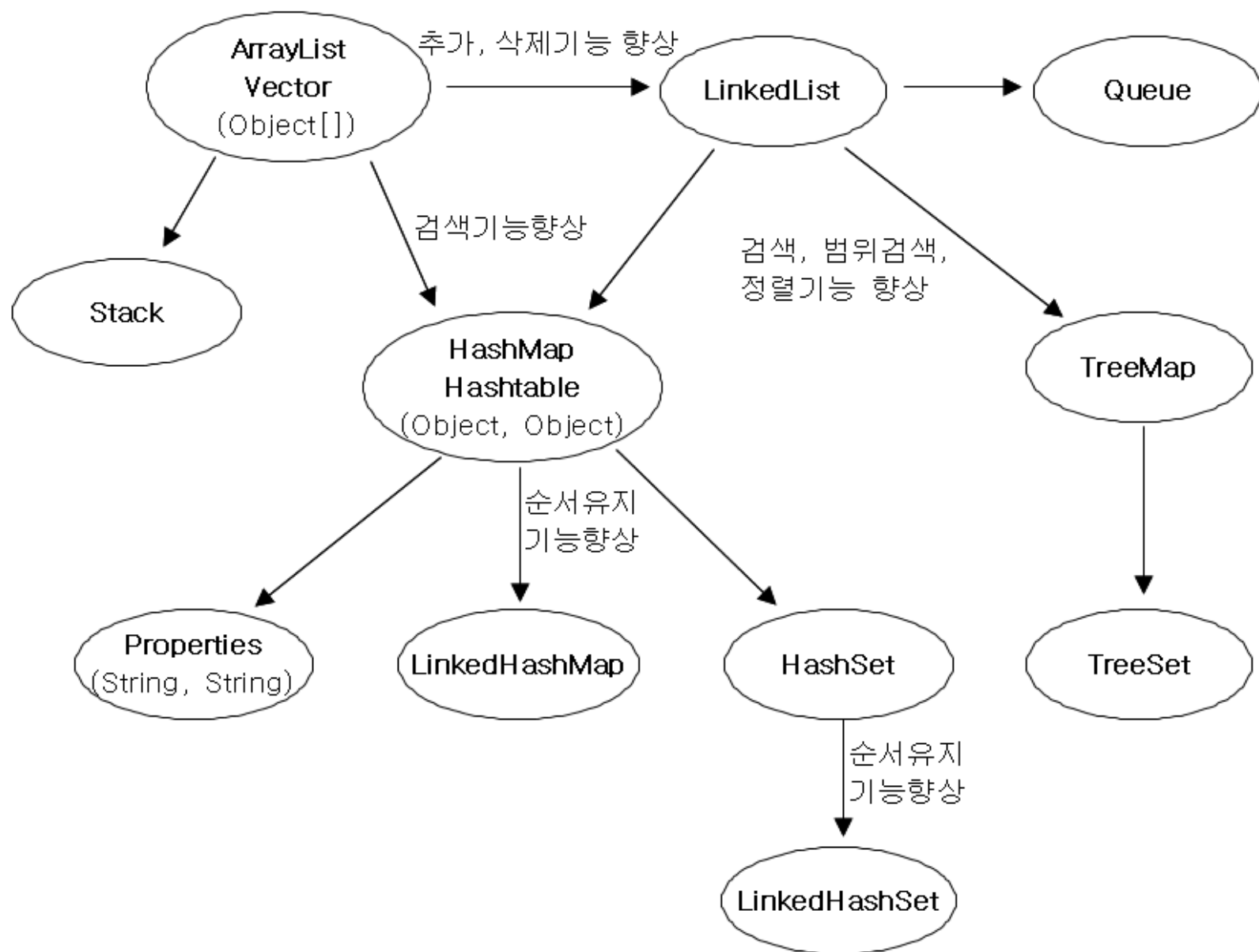
## [output.txt]

```
#Properties Example
#Sat Aug 29 10:58:41 KST 2009
capacity=10
size=10
timeout=30
language=\uD55C\uAE00
```

## [output.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>Properties Example</comment>
  <entry key="capacity">10</entry>
  <entry key="size">10</entry>
  <entry key="timeout">30</entry>
  <entry key="language">한글</entry>
</properties>
```

# 요약



# Quiz

- ❖ 클래스 파트에서 만들었던 관리프로그램을 컬렉션을 적용하여 Refactoring
- ❖ Refactoring?
  - 기존 동작 결과에 영향을 주지 않으며 소스코드를 더 효율적으로 수정하는 것
- ❖ 수정할 항목 예
  - List를 이용하여 전체 정보 저장할 컬렉션 클래스 선정 및 적용
  - Map을 이용하여 각 정보의 일련번호 추가(유일해야 되는 데이터에 대해서만)
  - Set을 이용하여 중복을 체크하는 기능 수정(중복처리가 필요한 경우)
- ❖ 위의 제시를 참조
- ❖ 배열 또는 직접 만든 리스트가 적용된 프로그램을 수정