Chapter 13 제네릭 프로그래밍

13.1 제네릭 프로그래밍

- generic programming: 클래스를 정의할 때 클래스 안에서 사용되는 자료형(타입)을 구체적으로 명시하지 않고 T와 같이 기호로 적어 자료형을 클래스의 매개변수로 만든 것
- 다양한 종류의 데이터를 처리할 수 있는 클래스와 메소드를 작성하는 기법
- ArrayList: 어떤 종류의 객체도 저장할 수 있는 배열

이전의 방법

- Object type: Object type의 변수는 어떤 객체도 참조할 수 있다.
- 내부에 Object type의 변수를 선언하고 이 변수로 데이터를 가리킨다.
- 모든 객체는 Object의 자손이다.
- 데이터를 꺼낼 때마다 형 변환을 해야 한다.(P)

제네릭을 이용한 방법

- generic class: type을 변수로 표시한다.
- type parameter: 객체 생성 시에 프로그래머에 의하여 결정된다.
- 타입 변수는 대문자로 표기한다.
- 타입 매개 변수의 값은 객체를 생성할 때 구체적으로 결정된다.

제네릭 메소드

예제 13-1

```
// 일반 클래스의 메소드에서도 타입 매개 변수를 사용하여서 제네릭 메소드를 정의할 수 있다.
public class GenericMethodTest {
    public static <T> void printArray(T[] array) {
        for (T element: array) {
            System.out.printf("%s", element);
        }
        System.out.println();
    }

    public static void main(String [] args) {
        Integer [] iArray = {10, 20, 30, 40, 50};
        Double [] dArray = {1.1, 1.2, 1.3, 1.4, 1.5};
        Character [] cArray = {'k', 'o', 'r', 'e', 'a'};

        printArray(iArray);
        printArray(dArray);
        printArray(cArray);
    }
}
```

13.2 컬렉션이란?

- Collection: 데이터를 저장하는 자료구조
- Collection은 제네릭 기법으로 구현되어 있어 어떤 타입의 데이터도 저장할 수 있다.
- 많이 사용되는 자료구조: 리스트(list), 스택(stack), 큐(queue), 집합(set), 해쉬 테이블(hash table)
- 배열은 크기가 고정되어 있기 때문에 데이터가 수시로 삽입되고 삭제되는 환경에서 사용하기 불편하다.

컬렉션의 종류

Interface	Description
Collection	모든 자료구조의 부모 Interface로서 객체의 모임을 나타낸다
Set	집합(중복된 원소를 가지지 않는)을 나타내는 자료구조
List	순서가 있으며 중복된 원소를 가질 수 있는 자료구조
Map	사전과 같이 키와 값들이 연관되어 있는 자료구조
Queue	극장에서의 대기줄과 같이 들어온 순서대로 나가는 자료구조

컬렉션의 특징

- 컬렉션은 제네릭을 사용한다.
- 컬렉션에서 기초 자료형을 저장할 수 없으며 클래스만 저장할 수 있다.
- auto boxing: 기본 자료형을 저장하면 자동으로 랩퍼 클래스의 객체로 변환하는 것

컬렉션 인터페이스의 주요 메소드

- Collection Interface는 모든 자료구조의 부모 인터페이스로서 Collection이 제공하는 Method는 모든 자료구조에서 사용할 수 있다.

Method	Description
boolean isEmpty() boolean contains(Object obj) boolean containsAll(Collection<?> c)	공백 상태이면 true를 반환 obj를 포함하고 있으면 true를 반환
boolean add(E element) boolean addAll(Collection<? extends E> from)	원소를 추가
boolean remove(Object obj) boolean removeAll(Collection c) retainAll(Collection c) clear()	원소를 삭제
Iterator iterator() Stream stream() Stream parallelStream()	원소를 방문
int size()	원소의 개수 반환

Method	Description
Object toArray() T[] toArray(T[] a)	컬렉션을 배열로 변환

컬렉션의 모든 요소 방문하기

```
String a[] = new String [] {"A", "B", "C", "D", "E"};
List<String> list = Arrays.asList(a);

// 1. for statement
for (int i = 0; i < list.size(); i++ )
    System.out.println(list.get(i));

// 2. for-each statement
for (string s: list)
    System.out.println(s);

// 3. Iterator
String s;
Iterator e = list.iterator();
while(e.hasNext())
{
    s = (String)e.next();
    System.out.println(s);
}

// 4. Stream 라이브러리를 이용하는 방법
list.forEach((n) -> System.out.println(n));
```

- Iterator의 목적은 컬렉션의 원소들을 접근하는 것이다.
- Iterator는 모든 Collection에 적용될 수 있다.
- Iterator는 java.util package에 정의되어 있는 Iterator 객체를 구현하는 객체다.
- Iterator는 3개의 정의된 메소드를 이용하여 Collection의 원소들을 하나씩 처리한다.

Method	Description
hasNext()	아직 방문하지 않은 원소가 있으면 true를 반환한다.
next()	다음 원소를 반환한다.
remove()	최근에 반환된 원소를 삭제한다.

13.3 벡터

- 벡터 클래스는 가변 크기의 배열을 구현한다.

Method of Vector	Description

Method of Vector	Description
add()	벡터에 요소 추가
add(index, object)	정해진 위치에 요소 삽입
get()	값을 추출
size()	현재 벡터 안에 있는 요소들의 개수를 반환

제네릭 기능을 사용하는 벡터

예제 13-2

```
import java.util.*;

class Monster {
    String name;
    double hp;
    public Monster(String name, double hp) {
        this.name;
        this.hp;
    }
    public String toString() { return "{" + name + ", " + hp +"}"};
}

public class VectorExample2 {
    public static void main(String [] args) {
        vector<Monster>list = new Vector<>() // Vector Class도 Generic을 지원하기 때문에 Vector 객체를 생성할 때 new Vector <Monster>이라고 하면 Monster 객체만을 저장하는 Vector를 생성할 수 있다.

        list.add(new Mondser("Mon1", 100));
        list.add(new Monster("Mod2", 200));
        list.add(new Monster("Mod3", 300));

        Systewm.out.println("벡터의 크기: " + )
    }
}
```

13.4 ArrayList

- ArrayList: 가변 크기의 배열을 구현하는 클래스(예: Vector)
- Vector는 스레드 간의 동기화를 지원하는데 반해 ArrayList는 동기화를 지원하지 않는다.

ArrayList의 기본 연산

- ArrayList는 타입 매개 변수를 가지고 제네릭 클래스를 제공하므로 ArrayList를 생성하려면 타입 매개 변수를 지정해야 한다.

ArrayList Method	Description

ArrayList Method	Description
add()	객체에 데이터를 저장한다.
add(index, obj)	기존의 데이터가 들어 있는 위치를 지정하여 객체에 데이터를 저장한다.
set()	특정한 위치에 있는 원소를 바꾼다.
remove(index)	index에 해당하는 데이터를 삭제한다.
get(index)	index에 해당하는 데이터를 반환한다.
size()	현재 저장된 원소의 개수를 알 수 있다.
contains()	어떤 값이 리스트에 포함되어 있는지 검사한다.

배열을 리스트로 변경하기

예제 13-3

```
import java.util.*;

class Point {
    private int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {return "(" + x + ", " + y + ")";}
}

public class ArrayListTest {
    public static void main(String [] args) {
        ArrayList<Point> list = new ArrayList<Point>();

        list.add(new Point(1, 1));
        list.add(new Point(1, 2));
        list.add(new Point(1, 3));
        list.add(new Point(1, 4));
        list.add(new Point(1, 5));

        System.out.println(list);
    }
}
// [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)]
```

예제 13-4

```
import java.util.ArrayList;

public class ArrayListTest2 {
```

```
    public static void main(String [] args) {
        ArrayList<String> list = new ArrayList<String>();

        list.add("Apple");
        list.add("Banana");
        list.add("Mango");
        list.add("Pear");
        list.add("Graph");

        int index = list.indexOf("Mango"); // indexOf() : ArrayList 안에 저장
된 데이터를 찾아서 인덱스를 반환하는 메소드

        System.out.println("Mango의 index: " + index);
    }
}
```

13.5 LinkedList

- 삽입이나 삭제는 뒤에 있는 원소들을 이동시켜야 하기 때문에 ArrayList의 중간에서 데이터를 빈번하게 삽이하거나 삭제하는 것은 문제가 된다.
- linked list는 각 원소를 링크로 연결한다. 각 원소들은 다음 원소를 가리키는 링크를 가지고 있으며 삽입이나 삭제가 발생하면 그 위치의 바로 앞에 있는 원소의 링크값만을 변경한다.

```
import java.util.LinkedList;

public class LinkedListTest {
    public static void main(String [] args) {
        LinkedList<String> list = new LinkedList<String>();

        list.add("Milk");
        list.add("Bread");
        list.add("Butter");
        list.add(1, "Apple");
        list.set(2, "Graph");
        list.remove(3);

        for (int i = 0; i < list.size(); i++)
            System.out.print(list.get(i) + " ");
    }
}

// Milk Apple Graph
```

Array vs LinkedList

13.6 Set

- Set Interface는 Collection Interface에 정의된 Method를 제공하며 데이터의 중복을 막도록 설계되었다.

Set Interface	Description
HashSet	해쉬 테이블에 원소를 저장하여 성능이 가능우수하지만 원소들의 순서가 일정하지 않다.
TreeSet	red-black tree에 원소를 저장하여 값에 따라 순서가 결정되지만 HashTree보다 느리다.
LinkedHashSet	해쉬 테이블과 연결 리스트를 결합한 것으로 원소들의 순서는 삽입되었던 순서와 같다.

예제 13-5

```
import java.util.Set;
import java.util.HashSet;
import java.util.TreeSet;
import java.util.LinkedHashSet;

public class SetTest {
    public static void main(String [] args) {
        TreeSet<String> set = new TreeSet<String>();

        set.add("Milk");
        set.add("Bread");
        set.add("Butter");
        set.add("Cheese");
        set.add("Ham");
        set.add("Ham");

        System.out.println(set);

        if (set.contains("Ham"))
            System.out.println("Ham 있음");
    }

    // Set<Integer> s1 = new HashSet<Integer>
(Arrays.asList(1,2,3,4,5,6,7));
    // Set<Integer> s2 = new HashSet<Integer>(Arrays.asList(2, 4, 6, 8));

    // s1.retainAll(S2);
    // System.out.println(s2);
}

/*
HashSet인 경우
[Ham, Butter, Cheese, Milk, Bread]
Ham 있음

TreeSet인 경우 : 알파벳 순서대로 정렬됨
[Bread, Butter, Cheese, Ham, Milk]
Ham 있음

LinkedHashSet인 경우 : 입력한 순서대로 출력됨
[Milk, Bread, Butter, Cheese, Ham]
Ham 있음
 */
```

합집합과 교집합

```
import java.util.Set;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.TreeSet;
import java.util.Arrays;

public class retainAllTest {
    public static void main(String [] rags) {
        Set<Integer> set1 = new HashSet<Integer>(Arrays.asList(1, 2, 3, 4,
5));
        Set<Integer> set2 = new TreeSet<>(Arrays.asList(3, 4, 5, 6, 7));
        Set<Integer> set3 = new LinkedHashSet<>(Arrays.asList(6, 7, 8, 9,
10));

        set1.retainAll(set2);
        set2.addAll(set3);

        System.out.println(set1); // [3, 4, 5]
        System.out.println(set2); // [3, 4, 5, 6, 7, 8, 9, 10]
    }
}
```

예제 13-6

```
import java.util.*;

public class FindDupplication {
    public static void main(String [] args) {
        Set<String> s = new HashSet<String>();

        String [] sample = { "사과", "사과", "바나나", "토마토" };

        for (String a : sample)
            if (!s.add(a))
                System.out.println("중복된 단어: " + a);

        System.out.println(s.size() + " 중복되지 않은 단어: " + s);
    }
}
```

13.7 Map

- key들은 중복되지 않아야 한다.
- put(): 데이터 저장, get(): 값 추출

예제 13-7

```
import java.util.Map;
import java.util.HashMap;

public class MapTest {
    public static void main(String [] args) {
        Map<String, String> map = new HashMap<String, String>();

        map.put("kim", "1234");
        map.put("park", "pass");
        map.put("lee", "word");

        System.out.println(map.get("lee"));

        for (String key: map.keySet()) {
            String value = map.get(key);
            System.out.println("key=" + key + ", value=" + value);
        }

        map.remove(3);
        map.put("chot", "password");
        System.out.println(map);
    }
}

/*
word
key=lee, value=word
key=kim, value=1234
key=park, value=pass
{chot=password, lee=word, kim=1234, park=pass}
*/
```

Map의 모든 요소 방문하기

- Map은 Collection을 구현하지 않는다.

```
// 1. for-each statement and keySet()
for (String key: map.keySet())
    System.out.println("key=" + key +", value=" + value);

// 2. 변수 추론 타입
for (var key: map.keySet()) {
    System.out.println("key=" + key + ", value=" + value);
}

// 3. 반복자
Iterator<String> it = map.keySet().iterator();
while(it.hasNext()) {
```

```
    String key = it.next();
    System.out.println("key=" + key + ", value=" + map.get(key));
}

// 4. 스트림 라이브러리
map.forEach((key, value) -> {
    System.out.println("key=" + key + ", value=" + value);
})
```

13.8 Queue

- 큐 : 데이터를 처리하기 전에 잠기 저장하고 있는 자료구조
- 후단(tail)에 원소가 추가하고 전단(head)에서 원소를 삭제한다.
- FIFO(first-in-first-out)
- 자바에서 큐sms Queue interface로 정의되며 이 Queue 인터페이스를 구현한 3개의 클래스가 주어진다. ArrauDeque, LinkedList, PriorityQueue

Queue 메소드

Queue Method	Description
add()	새로운 원소의 추가가 큐의 용량을 넘어서지 않으면 원소를 추가한다. 만약 용량을 넘어가면 IllegalStateException이 발생한다.
offer()	원소 추가에 실패하면 false가 반환되는 것만 add()와 다르다.
remove()	큐의 처음에 있는 원소를 제거하고 가져온다. 만약 원소가 없으면 NoSuchElementException이 발생한다.
poll()	큐의 처음에 있는 원소를 제거하고 가져온다. 만약 원소가 없으면 null을 반환한다.
element()	큐의 처음에 있는 원소를 반환한다. 큐가 비어있을 경우 NoSuchElementException이 발생한다.
peek()	큐의 처음에 있는 원소를 반환한다. 큐가 비어있을 경우 null을 반환한다.

예제

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueTest {
    public static void main(String [] args) {
        Queue<Integer> q = new LinkedList<Integer>();

        for (int i = 0; i < 5; i++) {
            q.add(i);
            System.out.println("add 이후 " + q);
        }

        for (int i = 0; i < 10; i++) {
```

```
            q.poll();
            System.out.println(q);
        }

        for (int i = 5; i < 10; i++) {
            q.offer(i);
            System.out.println("offer 이후 : " + q);
        }

        System.out.println("peak : " + q.peek());

        for (int i = 0; i < 5; i++) {
            q.remove();
            System.out.println("remove 이후 : " + q);
        }

        System.out.println("element : " + q.element());
    }
}
```

우선순위 큐

- PriorityQueue는 원소들이 무작위로 삽입되더라도 정렬된 상태로 원소들을 추출한다.
- 하지만 항상 정렬된 상태로 원소들을 저장하고 있는 것은 아니다.

```
import java.util.PriorityQueue;

public class PriorityQue {
    public static void main(String [] args) {
        PriorityQueue<Integer> q = new PriorityQueue<Integer>();

        q.add(3);
        q.offer(2);
        q.add(1);
        q.remove();

        System.out.println(q); // [2, 3]
    }
}
```

13.9 Collections 클래스

정렬

- 정렬 : 데이터를 어떤 기준에 의하여 순서대로 나열하는 것
- 정렬 알고리즘은 퀵 정렬, 합병 정렬, 히프 정렬 중 합병 정렬을 이용한다.
- Collection 클래스의 sort() 메소드는 List Interface를 구현하는 Collection에 대해 정렬을 수행한다.
- list에 들어 있는 원소가 String 타입이라면 알파벳 순서대로 정렬되며 Date 원소들이라면 시간적인 순서로 정렬된다.

예제 13-8

```
import java.util.Collections;
import java.util.List;
import java.util.Arrays;

public class Sort {
    public static void main(String [] args) {
        String[] s = {"i", "am", "iron", "man"};

        System.out.println(s);

        List<String> list = Arrays.asList(s);

        System.out.println(list);

        Collections.sort(list);

        System.out.println(list);
    }
}
/*
[Ljava.lang.String;@1dbd16a6
[i, am, iron, man]
[am, i, iron, man]
 */
```

예제 13-9

```
import java.util.Collections;
import java.util.List;
import java.util.Arrays;
import java.lang.Comparable;

class Student implements Comparable<Student> {
    private int number;
    private String name;

    public Student(int number, String name) {
        this.number = number;
        this.name = name;
    }

    public String toString() { return name; }

    public int compareTo(Student s) {
        return s.number - number;
    }
}
```

```
public class SortTest {
    public static void main(String [] args) {
        Student array [] = {
            new Student(10, "김철수"),
            new Student(20, "이철수"),
            new Student(30, "박철수")
        };

        System.out.println(array);

        List<Student> list = Arrays.asList(array);

        System.out.println(list);

        Collections.sort(list);

        System.out.println(list);

        Collections.sort(list, Collections.reverseOrder());

        System.out.println(list);
    }
}
```

섞기

- Shuffling(섞기) : 리스트에 존재하는 정렬을 파괴하여서 원소들의 순서를 랜덤하게 만든다.

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class Shuffle {
    public static void main(String [] args) {
        List<Integer> list = new ArrayList<Integer>();

        for (int i = 0; i < 10; i++) {
            list.add(i);
        }

        Collections.shuffle(list);

        System.out.println(list);

        Collections.sort(list);

        System.out.println(list);
    }
}
/*
[6, 8, 9, 3, 1, 4, 2, 7, 5, 0]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 */
```

탐색

- 탐색 : 리스트 안에서 원하는 원소를 찾는 것
- 리스트가 정렬되어 있지 않다면 처음부터 모든 원소를 방문한다.(선형 탐색)
- 리스트가 정렬되어 있다면 중간에 있는 원소와 먼저 비교한다.(이진 탐색)
- binarySearch() : 리스트가 정렬되어 있다고 가정하고 정렬된 리스트에서 지정된 원소를 이진 탐색한다. 리스트와 탐색할 원소를 받는다.
- binarySearch()의 반환값이 양수면 탐색이 성공한 객체의 위치이며 음수이면 탐색이 실패한 것이다.

예제 13-10 숫자들의 리스트 탐색하기

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class Search {
    public static void main(String [] args) {
        List<Integer> list = new ArrayList<Integer>();

        for (int i = 0; i <= 100; i++) {
            list.add(i);
        }

        System.out.println("탐색의 반환값: " + Collections.binarySearch(list,
50)); // 50

    }
}
```