

3. COLLECTION, GENERIC, I/O, THREADS

학습 목표

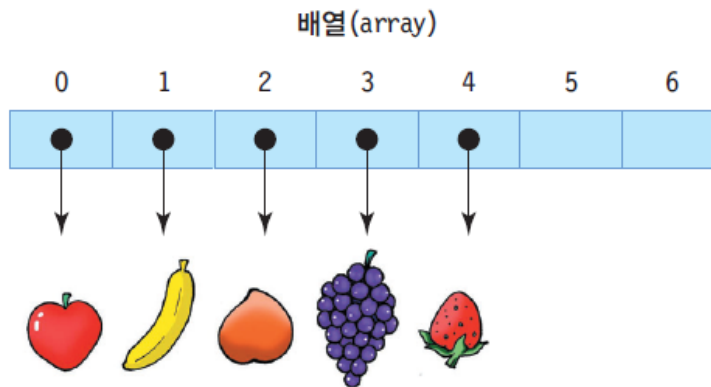
1. 컬렉션과 제네릭 개념
2. Vector<E> 활용
3. ArrayList<E> 활용
4. HashMap<K,V> 활용
5. Iterator<E> 활용
6. 사용자 제네릭 클래스 만들기

컬렉션(collection)의 개념

4

□ 컬렉션

- 요소(element)라고 불리는 가변 개수의 객체들의 저장소
 - 객체들의 컨테이너라고도 불림
 - 요소의 개수에 따라 크기 자동 조절
 - 요소의 삽입, 삭제에 따른 요소의 위치 자동 이동
- 고정 크기의 배열을 다루는 어려움 해소
- 다양한 객체들의 삽입, 삭제, 검색 등의 관리 용이



- 고정 크기 이상의 객체를 관리할 수 없다.
- 배열의 중간에 객체가 삭제되면 응용프로그램에서 자리를 옮겨야 한다.

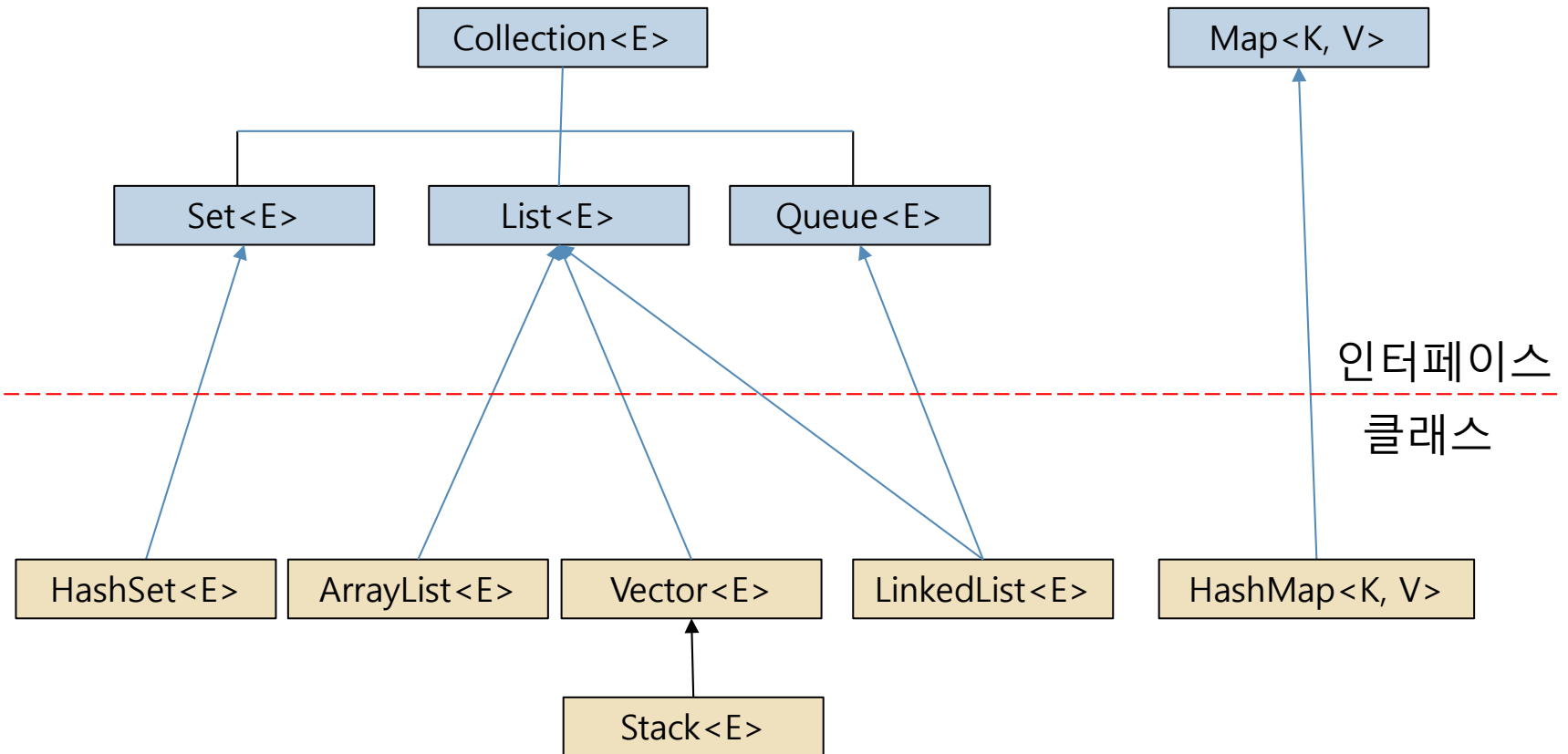
컬렉션(collection)



- 가변 크기로서 객체의 개수를 염려할 필요 없다.
- 컬렉션 내의 한 객체가 삭제되면 컬렉션이 자동으로 자리를 옮겨준다.

컬렉션 자바 인터페이스와 클래스

5



컬렉션의 특징

6

1. 컬렉션은 제네릭(generics) 기법으로 구현

▣ 제네릭

- 특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화시키는 기법
- 클래스나 인터페이스 이름에 <E>, <K>, <V> 등 타입매개변수 포함

▣ 제네릭 컬렉션 사례 : 벡터 Vector<E>

- <E>에서 E에 구체적인 타입을 주어 구체적인 타입만 다루는 벡터로 활용
- 정수만 다루는 컬렉션 벡터 Vector<Integer>
- 문자열만 다루는 컬렉션 벡터 Vector<String>

2. 컬렉션의 요소는 객체만 가능

- ▣ int, char, double 등의 기본 타입으로 구체화 불가
- ▣ 컬렉션 사례



```
Vector<int> v = new Vector<int>(); // 컴파일 오류. int는 사용 불가  
Vector<Integer> v = new Vector<Integer>(); // 정상 코드
```

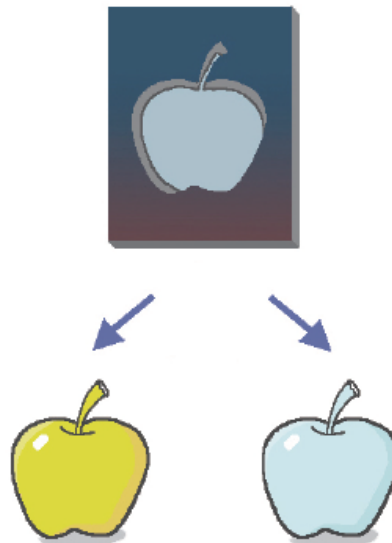
제네릭은 형판과 같은 개념

7

□ 제네릭

- ▣ 클래스나 메소드를 형판에서 찍어내듯이 생산할 수 있도록 일반화된 형판을 만드는 기법

금을 넣으면 금 사과,
은을 넣으면 은 사과가
만들어져요.



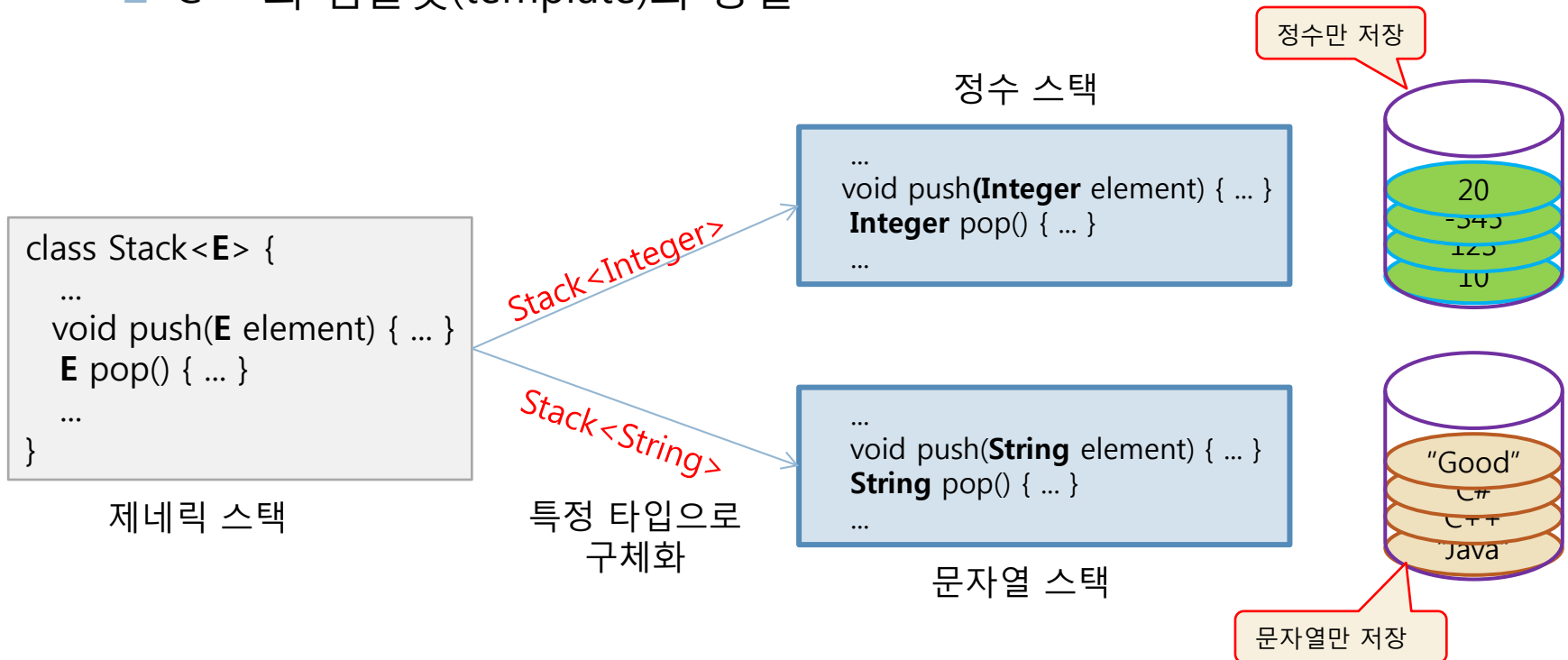
제네릭은 찍어내듯이
코드를 생산할 수 있는
형판입니다. 클래스나
메소드 모두 제네릭으로
만들어 찍어낼 수 있어요.



제네릭의 기본 개념

8

- 제네릭
 - ▣ JDK 1.5부터 도입(2004년 기점)
 - ▣ 모든 종류의 데이터 타입을 다룰 수 있도록 일반화된 타입 매개 변수로 클래스(인터페이스)나 메소드를 작성하는 기법
 - ▣ C++의 템플릿(template)과 동일



제네릭 Stack<E> 클래스

9

The screenshot shows the Java Platform SE 8 API documentation for the `Stack<E>` class. The browser address bar shows `http://docs.oracle.com/javase/8/docs/api/`. The page title is "Stack (Java Platform SE 8)". The navigation bar includes "OVERVIEW", "PACKAGE", "CLASS" (highlighted), "USE", "TREE", "DEPRECATED", "INDEX", and "HELP". The left sidebar shows a tree of classes and packages, with "SSLSession" selected. The main content area displays the class hierarchy for `Stack<E>`, showing it extends `Vector<E>`. The "All Implemented Interfaces:" section lists `Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, and `RandomAccess`. The class signature is `public class Stack<E>`.

compact1, compact2, compact3
java.util

Class Stack<E>

java.lang.Object
 java.util.AbstractCollection<E>
 java.util.AbstractList<E>
 java.util.Vector<E>
 java.util.Stack<E>

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

```
public class Stack<E>  
    extends Vector<E>
```


Vector<E>

10

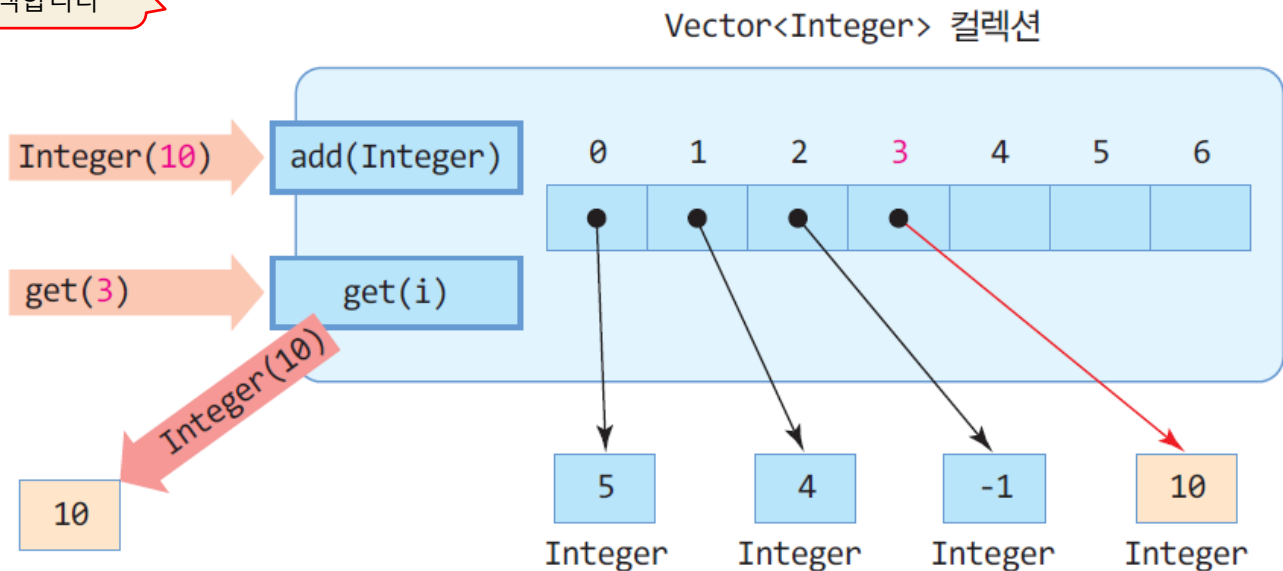
- 벡터 Vector<E>의 특성
 - ▣ <E>에 사용할 요소의 특정 타입으로 구체화
 - ▣ 배열을 가변 크기로 다룰 수 있게 하는 컨테이너
 - 배열의 길이 제한 극복
 - 요소의 개수가 넘치면 자동으로 길이 조절
 - ▣ 요소 객체들을 삽입, 삭제, 검색하는 컨테이너
 - 삽입, 삭제에 따라 자동으로 요소의 위치 조정
 - ▣ Vector에 삽입 가능한 것
 - 객체, null
 - 기본 타입의 값은 Wrapper 객체로 만들어 저장
 - ▣ Vector에 객체 삽입
 - 벡터의 맨 뒤, 중간에 객체 삽입 가능
 - ▣ Vector에서 객체 삭제
 - 임의의 위치에 있는 객체 삭제 가능

Vector<Integer> 벡터 컬렉션 내부

11

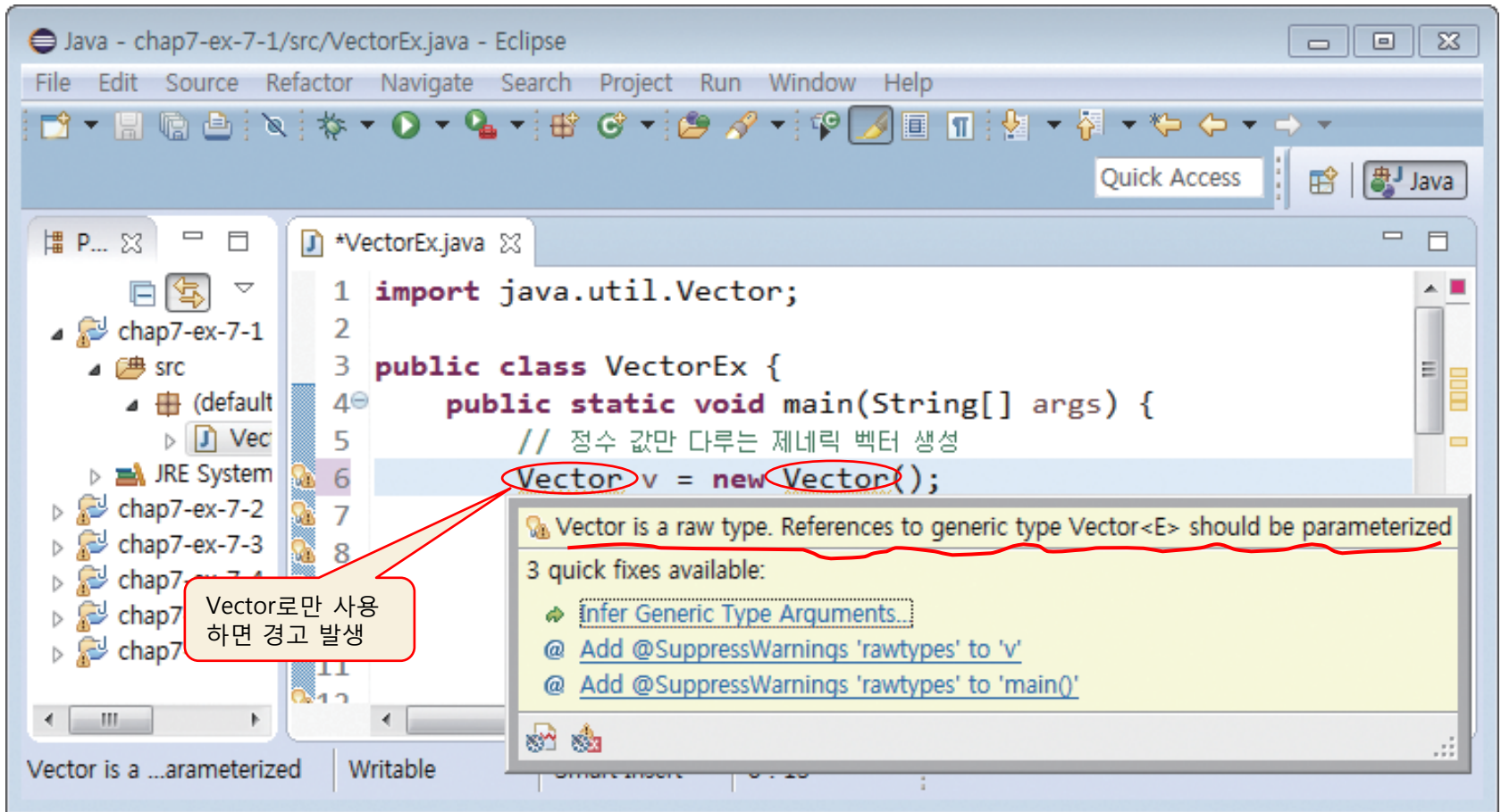
```
Vector<Integer> v = new Vector<Integer>();
```

add()를 이용하여 요소를 삽입하고
get()을 이용하여 요소를 검색합니다



타입 매개 변수 사용하지 않는 경우 경고 발생

12



Vector<Integer>나 Vector<String> 등 타입 매개 변수를 사용하여야 함

Vector<E> 클래스의 주요 메소드

13

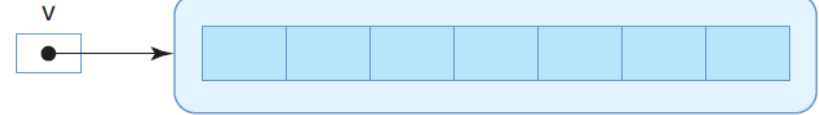
메소드	설명
<code>boolean add(E element)</code>	벡터의 맨 뒤에 element 추가
<code>void add(int index, E element)</code>	인덱스 index에 element를 삽입
<code>int capacity()</code>	벡터의 현재 용량 리턴
<code>boolean addAll(Collection<? extends E> c)</code>	컬렉션 c의 모든 요소를 벡터의 맨 뒤에 추가
<code>void clear()</code>	벡터의 모든 요소 삭제
<code>boolean contains(Object o)</code>	벡터가 지정된 객체 o를 포함하고 있으면 true 리턴
<code>E elementAt(int index)</code>	인덱스 index의 요소 리턴
<code>E get(int index)</code>	인덱스 index의 요소 리턴
<code>int indexOf(Object o)</code>	o와 같은 첫 번째 요소의 인덱스 리턴. 없으면 -1 리턴
<code>boolean isEmpty()</code>	벡터가 비어 있으면 true 리턴
<code>E remove(int index)</code>	인덱스 index의 요소 삭제
<code>boolean remove(Object o)</code>	객체 o와 같은 첫 번째 요소를 벡터에서 삭제
<code>void removeAllElements()</code>	벡터의 모든 요소를 삭제하고 크기를 0으로 만듦
<code>int size()</code>	벡터가 포함하는 요소의 개수 리턴
<code>Object[] toArray()</code>	벡터의 모든 요소를 포함하는 배열 리턴

Vector<Integer> 컬렉션 활용 사례

용량 7인 벡터

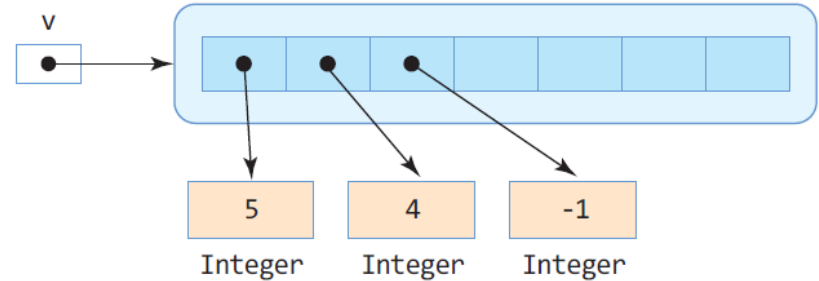
벡터 생성

```
Vector<Integer> v = new Vector<Integer>(7);
```



요소 삽입

```
v.add(5);  
v.add(new Integer(4));  
v.add(-1);
```



요소 개수 n
벡터의 용량 c

```
int n = v.size(); // n은 3  
int c = v.capacity(); // c는 7
```

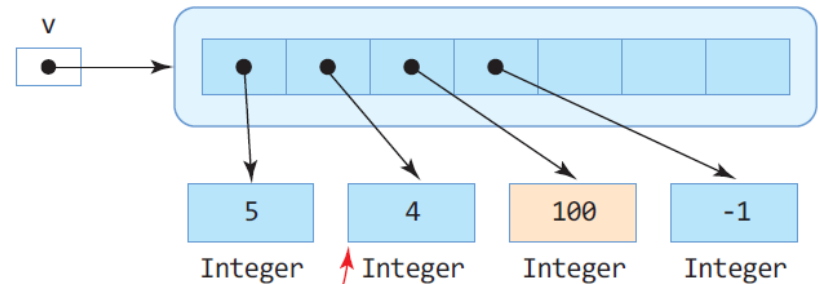
n = 3
c = 7

요소 중간 삽입

```
v.add(2, 100);
```

오류

```
v.add(5, 100);  
// v.size()보다 큰 곳에 삽입 불가능, 오류
```



요소 얻어내기

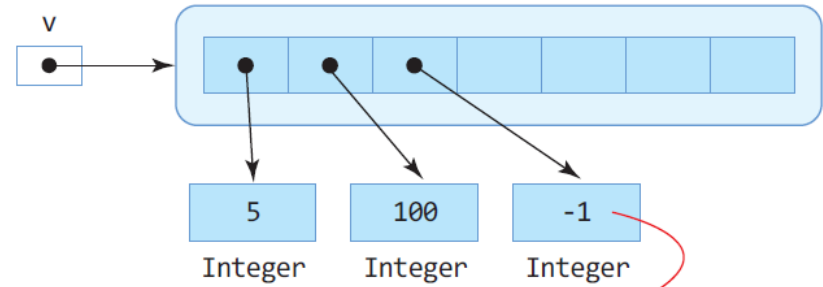
```
Integer obj = v.get(1);  
int i = obj.intValue();
```

obj
i = 4

Vector<Integer> 컬렉션 활용 사례(계속)

요소 삭제 `v.remove(1);`

오류 `v.remove(4);`
// 인덱스 4에 요소 객체가 없으므로 오류



마지막 요소 `int last = v.lastElement();`

`last = -1`

모든 요소 삭제 `v.removeAllElements();`



컬렉션과 자동 박싱/언박싱

16

□ JDK 1.5 이전

- 기본 타입 데이터를 Wrapper 객체로 만들어 삽입

```
Vector<Integer> v = new Vector<Integer>();  
v.add(new Integer(4));
```

- 컬렉션으로부터 요소를 얻어올 때, Wrapper 클래스로 캐스팅 필요


```
Integer n = (Integer)v.get(0);  
int k = n.intValue(); // k = 4
```

□ JDK 1.5부터

- 자동 박싱/언박싱이 작동하여 기본 타입 값 삽입 가능

```
Vector<Integer> v = new Vector<Integer> ();  
v.add(4); // 4 → new Integer(4)로 자동 박싱  
int k = v.get(0); // Integer 타입이 int 타입으로 자동 언박싱, k = 4
```

- 그러나, 타입 매개 변수를 기본 타입으로 구체화할 수는 없음

 `Vector<int> v = new Vector<int> (); // 컴파일 오류`

예제 7-1 : 정수만 다루는 Vector<Integer> 컬렉션 활용

17

정수만 다루는 Vector<Integer> 제네릭 벡터를 생성하고 활용하는 사례를 보인다.
다음 코드에 대한 결과는 무엇인가?

```
import java.util.Vector;

public class VectorEx {
    public static void main(String[] args) {
        // 정수 값만 다루는 제네릭 벡터 생성
        Vector<Integer> v = new Vector<Integer>();
        v.add(5); // 5 삽입
        v.add(4); // 4 삽입
        v.add(-1); // -1 삽입

        // 벡터 중간에 삽입하기
        v.add(2, 100); // 4와 -1 사이에 정수 100 삽입
        System.out.println("벡터 내의 요소 객체 수 : " + v.size());
        System.out.println("벡터의 현재 용량 : " + v.capacity());

        // 모든 요소 정수 출력하기
        for(int i=0; i<v.size(); i++) {
            int n = v.get(i); // 벡터의 i 번째 정수
            System.out.println(n);
        }
    }
}
```

```
// 벡터 속의 모든 정수 더하기
int sum = 0;
for(int i=0; i<v.size(); i++) {
    int n = v.elementAt(i); // 벡터의 i 번째 정수
    sum += n;
}
System.out.println("벡터에 있는 정수 합 : " + sum);
}
```

벡터 내의 요소 객체 수 : 4
벡터의 현재 용량 : 10
5
4
100
-1
벡터에 있는 정수 합 : 108

예제 7-2 : Point 클래스의 객체들만 저장하는 벡터 만들기

18

점 (x, y)를 표현하는 Point 클래스의 객체만 다루는 벡터의 활용을 보여라.

```
import java.util.Vector;
```

```
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 PointVectorEx {  
    public static void main(String[] args) {  
        Vector<Point> v = new Vector<Point>();  
  
        // 3 개의 Point 객체 삽입  
        v.add(new Point(2, 3));  
        v.add(new Point(-5, 20));  
        v.add(new Point(30, -8));  
  
        v.remove(1); // 인덱스 1의 Point(-5, 20) 객체 삭제  
  
        // 벡터에 있는 Point 객체 모두 검색하여 출력  
        for(int i=0; i<v.size(); i++) {  
            Point p = v.get(i); // 벡터의 i 번째 Point 객체 얻어내기  
            System.out.println(p); // p.toString()을 이용하여 객체 p 출력  
        }  
    }  
}
```

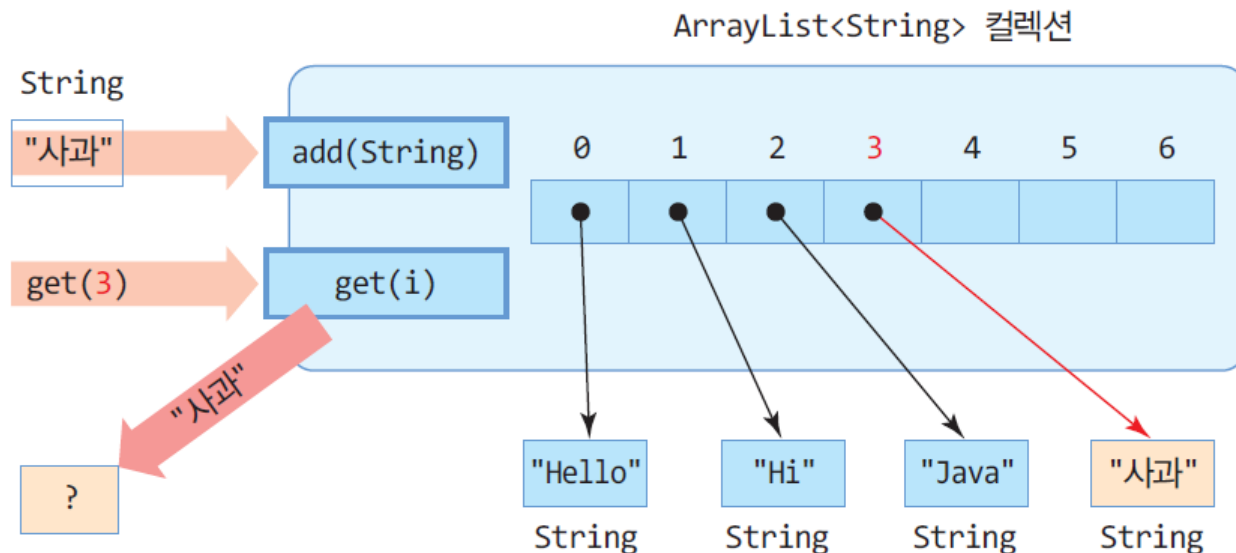
(2,3)
(30,-8)

ArrayList<E>

19

- ▣ 가변 크기 배열을 구현한 클래스
 - <E>에 요소로 사용할 특정 타입으로 구체화
- ▣ 벡터와 거의 동일
 - 요소 삽입, 삭제, 검색 등 벡터 기능과 거의 동일
 - 벡터와 달리 스레드 동기화 기능 없음
 - 다수 스레드가 동시에 *ArrayList*에 접근할 때 동기화되지 않음. 개발자가 스레드 동기화 코드 작성

```
ArrayList<String> = new ArrayList<String>();
```



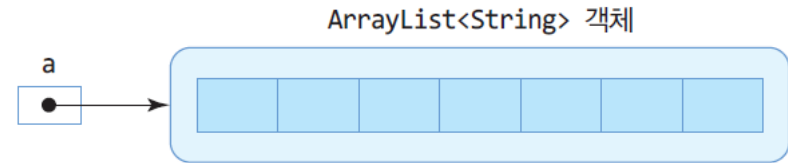
ArrayList<E> 클래스의 주요 메소드

20

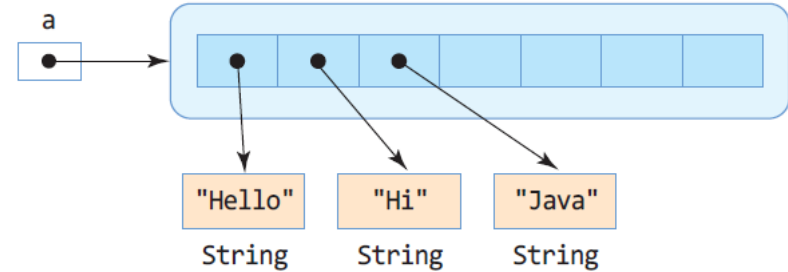
메소드	설명
<code>boolean add(E element)</code>	ArrayList의 맨 뒤에 element 추가
<code>void add(int index, E element)</code>	인덱스 index에 지정된 element 삽입
<code>boolean addAll(Collection<? extends E> c)</code>	컬렉션 c의 모든 요소를 ArrayList의 맨 뒤에 추가
<code>void clear()</code>	ArrayList의 모든 요소 삭제
<code>boolean contains(Object o)</code>	ArrayList가 지정된 객체를 포함하고 있으면 true 리턴
<code>E elementAt(int index)</code>	index 인덱스의 요소 리턴
<code>E get(int index)</code>	index 인덱스의 요소 리턴
<code>int indexOf(Object o)</code>	o와 같은 첫 번째 요소의 인덱스 리턴. 없으면 -1 리턴
<code>boolean isEmpty()</code>	ArrayList가 비어 있으면 true 리턴
<code>E remove(int index)</code>	index 인덱스의 요소 삭제
<code>boolean remove(Object o)</code>	o와 같은 첫 번째 요소를 ArrayList에서 삭제
<code>int size()</code>	ArrayList가 포함하는 요소의 개수 리턴
<code>Object[] toArray()</code>	ArrayList의 모든 요소를 포함하는 배열 리턴

ArrayList<String> 컬렉션 활용 사례

ArrayList 생성 `ArrayList<String> a = new ArrayList<String>(7);`



요소 삽입
`a.add("Hello");`
`a.add("Hi");`
`a.add("Java");`

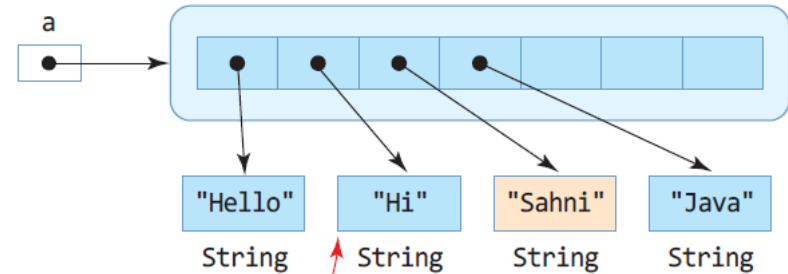


요소 개수 n `int n = a.size();` // n은 3
벡터의 용량 c `int c = a.capacity();` // capacity() 메소드 없음

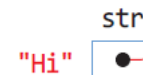
n = 3

요소 중간 삽입 `a.add(2, "Sahni");`

오류 `a.add(5, "Sahni");`
// a.size()보다 큰 위치에 삽입 불가능, 오류




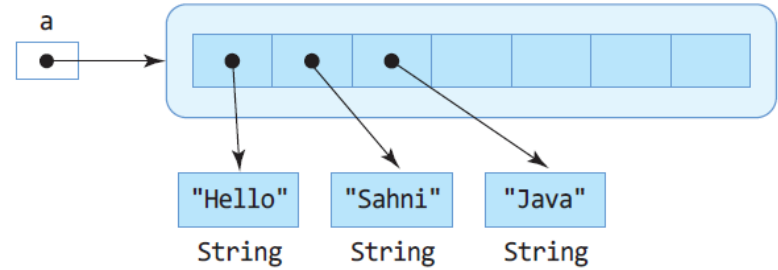
요소 알아내기 `String str = a.get(1);`



ArrayList<String> 컬렉션 활용 사례(계속)

요소 삭제 `a.remove(1);`

 `a.remove(4);` // 오류



모든 요소 삭제 `a.clear();`



예제 7-3 : 문자열만 다루는 ArrayList<String> 활용

23

이름을 4개 입력받아 ArrayList에 저장하고, ArrayList에 저장된 이름을 모두 출력한 후, 제일 긴 이름을 출력하라.

```
import java.util.*;

public class ArrayListEx {
    public static void main(String[] args) {
        // 문자열만 삽입가능한 ArrayList 컬렉션 생성
        ArrayList<String> a = new ArrayList<String>();

        // 키보드로부터 4개의 이름 입력받아 ArrayList에 삽입
        Scanner scanner = new Scanner(System.in);
        for(int i=0; i<4; i++) {
            System.out.print("이름을 입력하세요>>");
            String s = scanner.next(); // 키보드로부터 이름 입력
            a.add(s); // ArrayList 컬렉션에 삽입
        }

        // ArrayList에 들어 있는 모든 이름 출력
        for(int i=0; i<a.size(); i++) {
            // ArrayList의 i 번째 문자열 얻어오기
            String name = a.get(i);
            System.out.print(name + " ");
        }
    }
}
```

```
// 가장 긴 이름 출력
int longestIndex = 0;
for(int i=1; i<a.size(); i++) {
    if(a.get(longestIndex).length() < a.get(i).length())
        longestIndex = i;
}
System.out.println("\n가장 긴 이름은 : " +
    a.get(longestIndex));
}
```

```
이름을 입력하세요>>Mike
이름을 입력하세요>>Jane
이름을 입력하세요>>Ashley
이름을 입력하세요>>Helen
Mike Jane Ashley Helen
가장 긴 이름은 : Ashley
```

컬렉션의 순차 검색을 위한 Iterator

24

□ Iterator<E> 인터페이스

- 리스트 구조의 컬렉션에서 요소의 순차 검색을 위한 인터페이스
 - Vector<E>, ArrayList<E>, LinkedList<E>가 상속받는 인터페이스

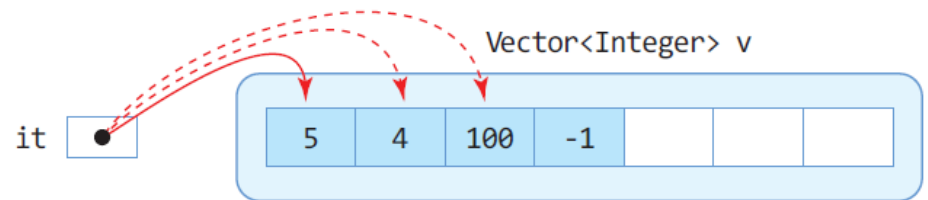
□ Iterator 객체 얻어내기

- 컬렉션의 iterator() 메소드 호출
 - 해당 컬렉션을 순차 검색할 수 있는 Iterator 객체 리턴
- 컬렉션 검색 코드

```
Vector<Integer> v = new Vector<Integer>();  
Iterator<Integer> it = v.iterator();
```

```
while(it.hasNext()) { // 모든 요소 방문  
    int n = it.next(); // 다음 요소 리턴  
    ...  
}
```

메소드	설명
boolean hasNext()	다음 반복에서 사용될 요소가 있으면 true 리턴
E next()	다음 요소 리턴
void remove()	마지막으로 리턴된 요소 제거



Vector<Integer> 객체와 Iterator 객체의 관계

예제 7-4 : Iterator<Integer>를 이용하여 정수 벡터 검색

25

예제 7-1의 코드 중에서 벡터 검색 부분을 Iterator<Integer>를 이용하여 수정하라.

```
import java.util.*;

public class IteratorEx {
    public static void main(String[] args) {
        // 정수 값만 다루는 제네릭 벡터 생성
        Vector<Integer> v = new Vector<Integer>();
        v.add(5); // 5 삽입
        v.add(4); // 4 삽입
        v.add(-1); // -1 삽입
        v.add(2, 100); // 4와 -1 사이에 정수 100 삽입

        // Iterator를 이용한 모든 정수 출력하기
        Iterator<Integer> it = v.iterator(); // Iterator 객체 얻기
        while(it.hasNext()) {
            int n = it.next();
            System.out.println(n);
        }
    }
}
```

```
// Iterator를 이용하여 모든 정수 더하기
int sum = 0;
it = v.iterator(); // Iterator 객체 얻기
while(it.hasNext()) {
    int n = it.next();
    sum += n;
}
System.out.println("벡터에 있는 정수 합 : " + sum);
}
```

```
5
4
100
-1
벡터에 있는 정수 합 : 108
```


HashMap<K,V>

26

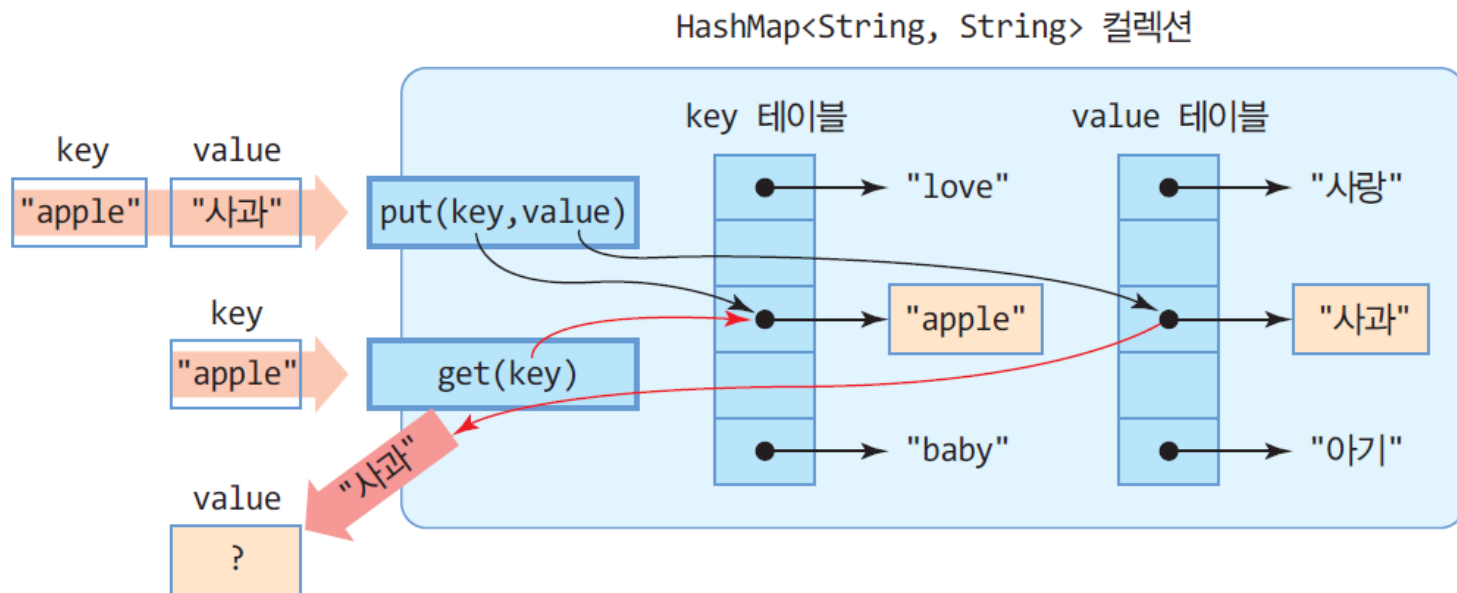
- ▣ 키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
 - K : 키로 사용할 요소의 타입
 - V : 값으로 사용할 요소의 타입
 - 키와 값이 한 쌍으로 삽입
 - '값'을 검색하기 위해서는 반드시 '키' 이용
- ▣ 삽입 및 검색이 빠른 특징
 - 요소 삽입 : put() 메소드
 - 요소 검색 : get() 메소드
- ▣ 예) HashMap<String, String> 생성, 요소 삽입, 요소 검색

```
HashMap<String, String> h = new HashMap<String, String>(); // 해시맵 객체 생성  
h.put("apple", "사과"); // "apple" 키와 "사과" 값의 쌍을 해시맵에 삽입  
String kor = h.get("apple"); // "apple" 키로 값 검색. kor는 "사과"
```

HashMap<String, String>의 내부 구성

27

```
HashMap<String, String> map = new HashMap<String, String>();
```



HashMap<K,V>의 주요 메소드

28

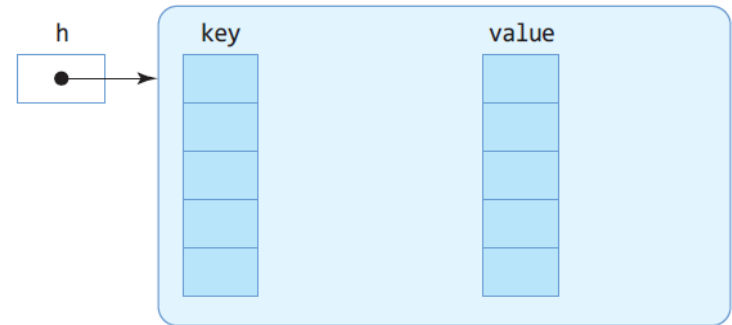
메소드	설명
<code>void clear()</code>	HashMap의 모든 요소 삭제
<code>boolean containsKey(Object key)</code>	지정된 키(key)를 포함하고 있으면 true 리턴
<code>boolean containsValue(Object value)</code>	하나 이상의 키를 지정된 값(value)에 매핑시킬 수 있으면 true 리턴
<code>V get(Object key)</code>	지정된 키(key)에 매핑되는 값 리턴. 키에 매핑되는 어떤 값도 없으면 null 리턴
<code>boolean isEmpty()</code>	HashMap이 비어 있으면 true 리턴
<code>Set<K> keySet()</code>	HashMap에 있는 모든 키를 담은 Set<K> 컬렉션 리턴
<code>V put(K key, V value)</code>	key와 value를 매핑하여 HashMap에 저장
<code>V remove(Object key)</code>	지정된 키(key)와 이에 매핑된 값을 HashMap에서 삭제
<code>int size()</code>	HashMap에 포함된 요소의 개수 리턴

HashMap<String, String> 컬렉션 활용 사례

해시맵 생성

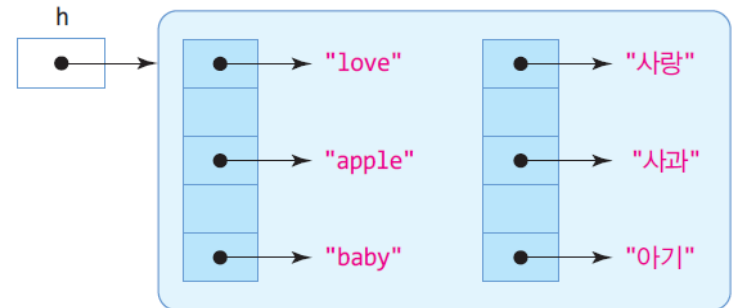
```
HashMap<String, String> h =  
new HashMap<String, String>();
```

HashMap<String, String> 객체



(키, 값) 삽입

```
h.put("baby", "아기");  
h.put("love", "사랑");  
h.put("apple", "사과");
```



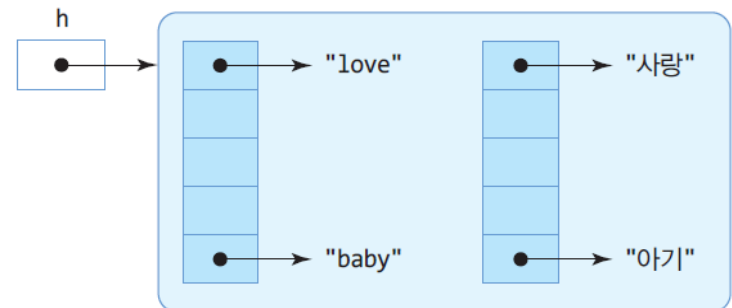
키로 값 읽기

```
String kor = h.get("love");
```

kor = "사랑"

키로 요소 삭제

```
h.remove("apple");
```



요소 개수

```
int n = h.size();
```

n = 2

예제 7-5 : HashMap<String, String>로 (영어, 한글) 단어 쌍을 저장하고 검색하기

30

영어 단어와 한글 단어의 쌍을 HashMap에 저장하고, 영어 단어로 한글 단어를 검색하는 프로그램을 작성하라.

```
import java.util.*;
public class HashMapDicEx {
    public static void main(String[] args) {
        // 영어 단어와 한글 단어의 쌍을 저장하는 HashMap 컬렉션 생성
        HashMap<String, String> dic = new HashMap<String, String>();

        // 3 개의 (key, value) 쌍을 dic에 저장
        dic.put("baby", "아기"); // "baby"는 key, "아기"은 value
        dic.put("love", "사랑");
        dic.put("apple", "사과");

        // dic 해시맵에 들어 있는 모든 (key, value) 쌍 출력
        Set<String> keys = dic.keySet(); // 모든 키를 Set 컬렉션에 받아옴
        Iterator<String> it = keys.iterator(); // Set에 접근하는 Iterator 리턴
        while(it.hasNext()) {
            String key = it.next(); // 키
            String value = dic.get(key); // 값
            System.out.print("(" + key + "," + value + ");");
        }
        System.out.println();
    }
}
```

```
// 영어 단어를 입력받고 한글 단어 검색
Scanner scanner = new Scanner(System.in);
for(int i=0; i<3; i++) {
    System.out.print("찾고 싶은 단어는?");
    String eng = scanner.next();
    // 해시맵에서 '키' eng의 '값' kor 검색
    String kor = dic.get(eng);
    if(kor == null)
        System.out.println(eng +
            "는 없는 단어 입니다.");
    else
        System.out.println(kor);
}
}
```

```
(love,사랑)(apple,사과)(baby,아기)
찾고 싶은 단어는?apple
사과
찾고 싶은 단어는?babo
babo는 없는 단어 입니다.
찾고 싶은 단어는?love
사랑
```

제네릭 만들기

31

□ 제네릭 클래스 작성

▣ 클래스 이름 옆에 일반화된 타입 매개 변수 추가

```
public class MyClass<T> {
```

val의 타입은 T

```
    T val;
```

```
    void set(T a) {
```

```
        val = a;
```

T 타입의 값 a를 val에 지정

```
    }
```

```
    T get() {
```

```
        return val;
```

T 타입의 값 val 리턴

```
    }
```

```
}
```

제네릭 클래스 MyClass 선언, 타입 매개 변수 T

▣ 제네릭 객체 생성 및 활용

■ 제네릭 타입에 구체적인 타입을 지정하여 객체를 생성하는 것을 **구체화**라고 함

```
MyClass<String> s = new MyClass<String>(); // T를 String으로 구체화
```

```
s.set("hello");
```

```
System.out.println(s.get()); // "hello" 출력
```

```
MyClass<Integer> n = new MyClass<Integer>(); // T를 Integer로 구체화
```

```
n.set(5);
```

```
System.out.println(n.get()); // 숫자 5 출력
```

예제 7-6 : 제네릭 스택 만들기

32

스택을 제네릭 클래스로 작성하고, String과 Integer형 스택을 사용하는 예를 보여라.

```
class GStack<T> {
    int tos;
    Object [] stck;
    public GStack() {
        tos = 0;
        stck = new Object [10];
    }
    public void push(T item) {
        if(tos == 10)
            return;
        stck[tos] = item;
        tos++;
    }
    public T pop() {
        if(tos == 0)
            return null;
        tos--;
        return (T)stck[tos];
    }
}
```

```
public class MyStack {
    public static void main(String[] args) {
        GStack<String> stringStack = new GStack<String>();
        stringStack.push("seoul");
        stringStack.push("busan");
        stringStack.push("LA");

        for(int n=0; n<3; n++)
            System.out.println(stringStack.pop());

        GStack<Integer> intStack = new GStack<Integer>();
        intStack.push(1);
        intStack.push(3);
        intStack.push(5);

        for(int n=0; n<3; n++)
            System.out.println(intStack.pop());
    }
}
```

LA
busan
seoul
5
3
1

학습 목표

1. 자바의 입출력 스트림에 대한 이해
2. 텍스트 파일 입출력
3. 바이너리 파일 입출력
4. File 클래스로 파일 속성 알아내기
5. 파일 복사 응용 사례

자바의 입출력 스트림

34

□ 자바의 입출력 스트림

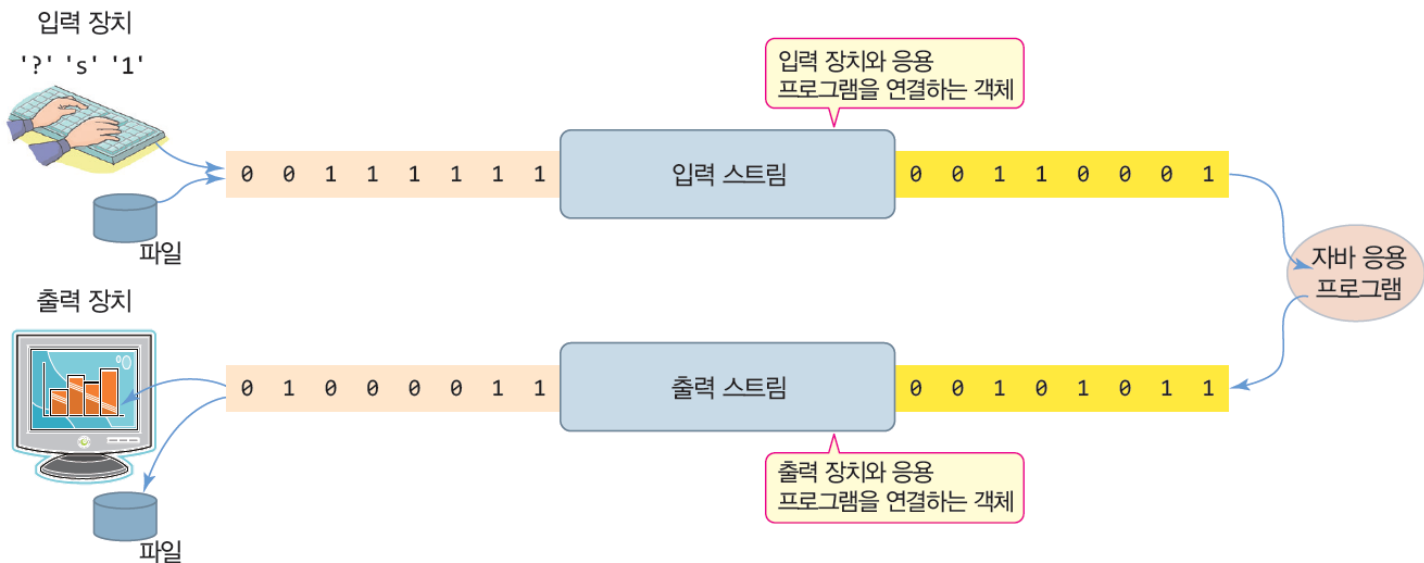
▣ 입출력 장치와 자바 응용 프로그램 연결

- 입력 스트림 : 입력 장치로부터 자바 프로그램으로 데이터를 전달하는 객체
- 출력 스트림 : 자바 프로그램에서 출력 장치로 데이터를 보내는 객체

▣ 특징

- 입출력 스트림 기본 단위 : 바이트
- 단방향 스트림, 선입선출 구조

자바 프로그램 개발자는 직접 입력 장치에서 읽지 않고 입력 스트림을 통해 읽으며, 스크린 등 출력 장치에 직접 출력하지 않고 출력 스트림에 출력하면 된다.



자바의 입출력 스트림 종류

35

▣ 문자 스트림

- 문자만 입출력하는 스트림
- 문자가 아닌 바이너리 데이터는 스트림에서 처리하지 못함
- 문자가 아닌 데이터를 문자 스트림으로 출력하면 깨진 기호가 출력
- 바이너리 파일을 문자 스트림으로 읽으면 읽을 수 없는 바이트가 생겨서 오류 발생

예) 텍스트 파일을 읽는 입력 스트림

▣ 바이트 스트림

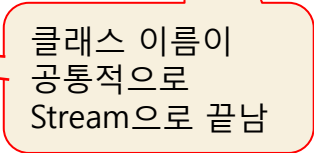
- 입출력 데이터를 단순 바이트의 흐름으로 처리
- 문자 데이터든 바이너리 데이터든 상관없이 처리 가능

예) 바이너리 파일을 읽는 입력 스트림

문자 스트림과 바이트 스트림의 흐름 비교

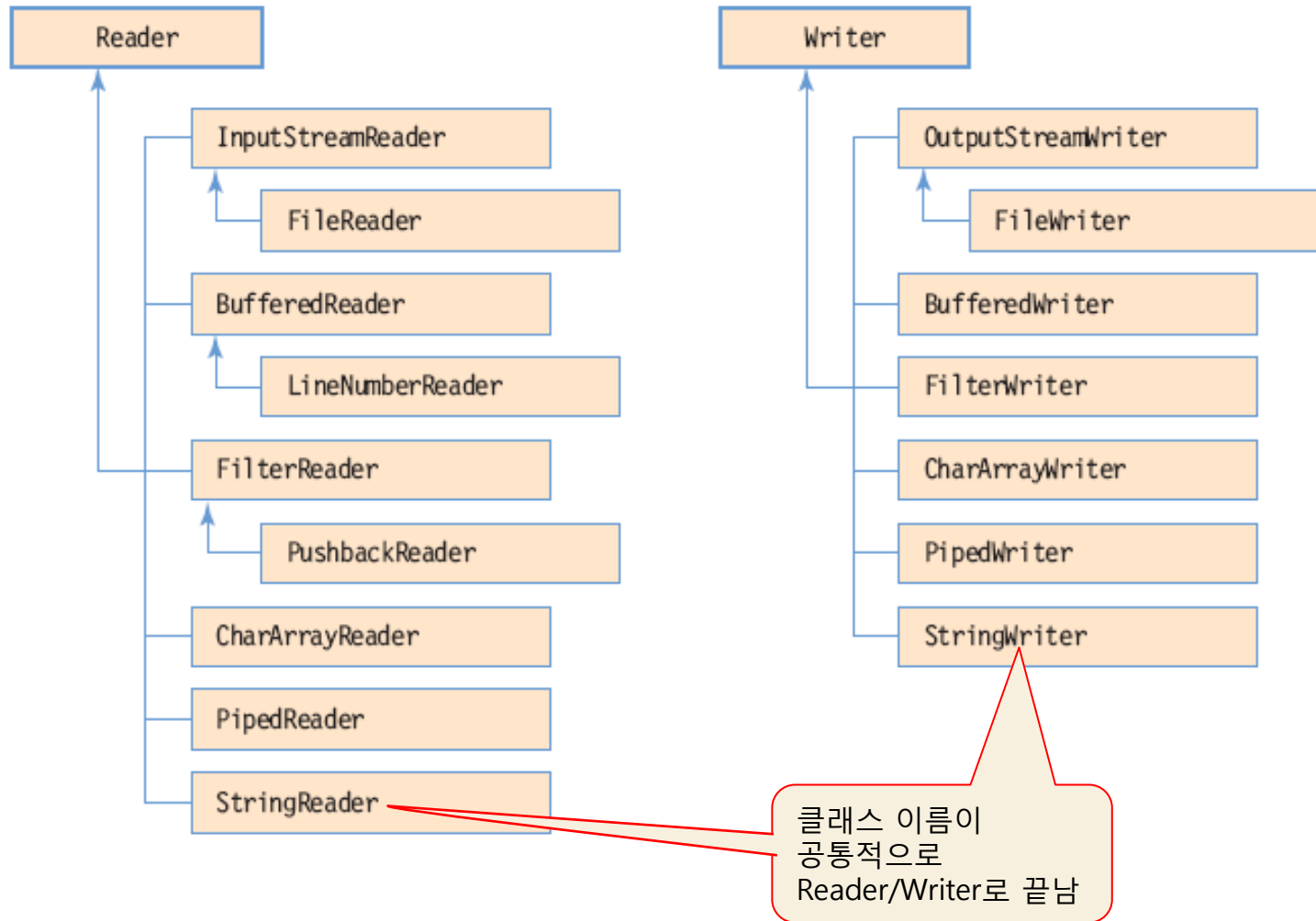


37



JDK의 문자 스트림 클래스 계층 구조

38



스트림 연결

39

- 여러 개의 스트림을 연결하여 사용할 수 있음
 - 예) 키보드에서 문자를 입력받기 위해 System.in과 InputStreamReader를 연결한 코드

```
InputStreamReader rd = new InputStreamReader(System.in);
```

```
while(true) {  
    int c = rd.read(); // 입력 스트림으로부터 키 입력. c는 입력된 키 문자 값  
    if(c == -1) // 입력 스트림의 끝을 만나는 경우  
        break; // 입력 종료  
}
```



문자 스트림으로 텍스트 파일 읽기

40

- 텍스트 파일을 읽기 위해 문자 스트림 `FileReader` 클래스 이용

1. 파일 입력 스트림 생성(파일 열기)

- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileReader fin = new FileReader("c:\\test.txt");
```

2. 파일 읽기

- `read()`로 문자 하나 씩 파일에서 읽음

```
int c;  
while((c = fin.read()) != -1) { // 문자를 c에 읽음. 파일 끝까지 반복  
    System.out.print((char)c); // 문자 c 화면에 출력  
}
```

3. 스트림 닫기

- 스트림이 더 이상 필요 없으면 닫아야 함. 닫힌 스트림에서는 읽을 수 없음
- `close()`로 스트림 닫기

```
fin.close();
```

파일 입출력과 예외 처리

41

□ 파일 입출력 동안 예외 발생 가능

□ 스트림 생성 동안 : **FileNotFoundException** 발생 가능

- 파일의 경로명이 틀리거나, 디스크의 고장 등으로 파일을 열 수 없음

```
FileReader fin = new FileReader("c:\wwtest.txt"); // FileNotFoundException 발생가능
```

□ 파일 읽기, 쓰기, 닫기를 하는 동안 : **IOException** 발생 가능

- 디스크 오동작, 파일이 중간에 깨진 경우, 디스크 공간이 모자라서 파일 입출력 불가

```
int c = fin.read(); // IOException 발생 가능
```

□ try-catch 블록 반드시 필요

□ 자바 컴파일러의 강제 사항

생략 가능. FileNotFoundException은 IOException을 상속받기 때문에 아래의 catch 블록 하나만 있으면 됨

```
try {
    FileReader fin = new FileReader("c:\wwtest.txt");
    ..
    int c = fin.read();
    ...
    fin.close();
} catch(FileNotFoundException e) {
    System.out.println("파일을 열 수 없음");
} catch(IOException e) {
    System.out.println("입출력 오류");
}
```


FileReader의 생성자와 주요 메소드

42

생성자	설명
<code>FileReader(File file)</code>	file에 지정된 파일로부터 읽는 <code>FileReader</code> 생성
<code>FileReader(String name)</code>	name 이름의 파일로부터 읽는 <code>FileReader</code> 생성

메소드	설명
<code>int read()</code>	한 개의 문자를 읽어 정수형으로 리턴
<code>int read(char[] cbuf)</code>	최대 <code>cbuf</code> 배열의 크기만큼 문자들을 읽어 <code>cbuf</code> 배열에 저장. 만일 읽는 도중 EOF를 만나면 실제 읽은 문자 개수 리턴
<code>int read(char[] cbuf, int off, int len)</code>	최대 <code>len</code> 크기만큼 읽어 <code>cbuf</code> 배열의 <code>off</code> 부터 저장. 읽는 도중 EOF를 만나면 실제 읽은 문자 개수 리턴
<code>String getEncoding()</code>	스트림이 사용하는 문자 집합의 이름 리턴
<code>void close()</code>	입력 스트림을 닫고 관련된 시스템 자원 해제

예제 13-1 : FileReader로 텍스트 파일 읽기

43

FileReader를 이용하여 c:\windows\system.ini 파일을 읽어 화면에 출력하는 프로그램을 작성하라. system.ini는 텍스트 파일이다..

```
import java.io.*;

public class FileReaderEx {
    public static void main(String[] args) {
        FileReader in = null;
        try {
            in = new FileReader("c:\\windows\\system.ini");
            int c;
            while ((c = in.read()) != -1) { // 한 문자씩 파일 끝까지 읽는다.
                System.out.print((char)c);
            }
            in.close();
        }
        catch (IOException e) {
            System.out.println("입출력 오류");
        }
    }
}
```

파일 끝을 만나면 -1 리턴

```
; for 16-bit app support
[386Enh]
woafont=dosapp.fon
EGA80WOA.FON=EGA80WOA.FON
EGA40WOA.FON=EGA40WOA.FON
CGA80WOA.FON=CGA80WOA.FON
CGA40WOA.FON=CGA40WOA.FON
```

```
[drivers]
wave=mmdrv.dll
timer=timer.driv
```

```
[mci]
```

문자 스트림으로 텍스트 파일 쓰기

44

- 텍스트 파일에 쓰기 위해 문자 스트림 `FileWriter` 클래스 이용

1. 파일 출력 스트림 생성(파일 열기)

- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileWriter fout = new FileWriter("c:\\test.txt");
```

2. 파일 쓰기

- `write()`로 문자 하나 씩 파일에 기록

```
fout.write('A'); // 문자 'A'를 파일에 기록
```

- 블록 단위로 쓰기 가능

```
char [] buf = new char [1024];  
fout.write(buf, 0, buf.length); // buf[0]부터 버퍼 크기만큼 쓰기
```

3. 스트림 닫기

- `close()`로 스트림 닫기

```
fout.close(); // 스트림 닫기. 더 이상 스트림에 기록할 수 없다.
```

FileWriter의 생성자와 주요 메소드

45

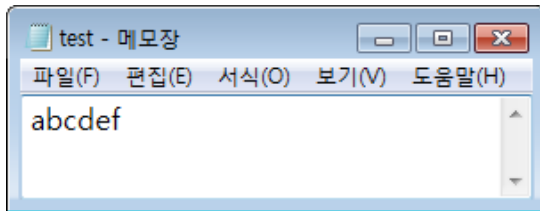
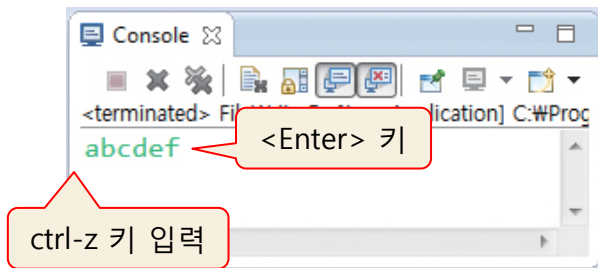
생성자	설명
<code>FileWriter(File file)</code>	file에 데이터를 저장할 <code>FileWriter</code> 생성
<code>FileWriter(String name)</code>	name 파일에 데이터를 저장할 <code>FileWriter</code> 생성
<code>FileWriter(File file, boolean append)</code>	<code>FileWriter</code> 를 생성하며, <code>append</code> 가 <code>true</code> 이면 파일의 마지막부터 데이터 저장
<code>FileWriter(String name, boolean append)</code>	<code>FileWriter</code> 를 생성하며, <code>append</code> 가 <code>true</code> 이면 파일의 마지막부터 데이터 저장

메소드	설명
<code>void write(int c)</code>	c를 char로 변환하여 한 개의 문자 출력
<code>void write(String str, int off, int len)</code>	인덱스 off부터 len개의 문자를 str 문자열에서 출력
<code>void write(char[] cbuf, int off, int len)</code>	인덱스 off부터 len개의 문자를 배열 cbuf에서 출력
<code>void flush()</code>	스트림에 남아 있는 데이터 모두 출력
<code>String getEncoding()</code>	스트림이 사용하는 문자 집합의 이름 리턴
<code>void close()</code>	출력 스트림을 닫고 관련된 시스템 자원 해제

예제 13-2 : FileWriter를 이용하여 텍스트 파일 쓰기

46

사용자로부터 입력받은 텍스트를 c:\tmp\test.txt 파일에 저장하는 프로그램을 작성하라. 사용자는 키 입력 후 라인 첫 위치에 ctrl-z 키(EOF)를 입력하라.



실행 결과 test.txt 파일 생성

```
import java.io.*;

public class FileWriterEx {
    public static void main(String[] args) {
        InputStreamReader in = new InputStreamReader(System.in);

        FileWriter fout = null;
        int c;
        try {
            fout = new FileWriter("c:\\tmp\\test.txt");
            while ((c = in.read()) != -1) {
                fout.write(c); // 키보드로부터 받은 문자를 파일에 저장
            }
            in.close();
            fout.close();
        }
        catch (IOException e) {
            System.out.println("입출력 오류");
        }
    }
}
```

바이트 스트림으로 바이너리 파일 쓰기

47

□ 바이너리 값을 파일에 저장하기

- ▣ 프로그램 내의 변수, 배열, 버퍼에 든 바이너리 값을 파일에 그대로 기록
 - FileOutputStream 클래스 이용

1. 파일 출력 스트림 생성(파일 열기)

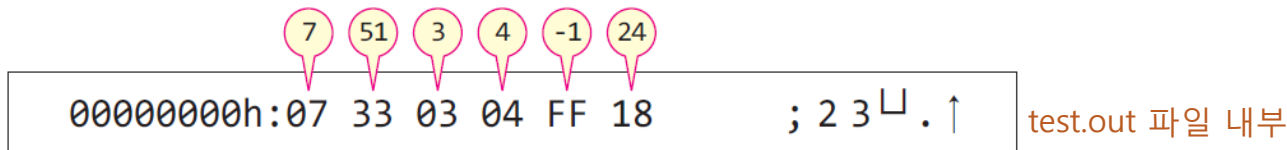
- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileOutputStream fout = new FileOutputStream("c:\\test.out");
```

2. 파일 쓰기

- write()로 문자 하나 씩 파일에 기록

```
byte b[] = {7,51,3,4,-1,24};  
for(int i=0; i<b.length; i++) fout.write(b[i]); // 배열 b를 바이너리 그대로 기록
```



3. 스트림 닫기

- close()로 스트림 닫기

FileOutputStream의 생성자와 주요 메소드

48

생성자	설명
<code>FileOutputStream(File file)</code>	<code>file</code> 이 지정하는 파일에 출력하는 <code>FileOutputStream</code> 생성
<code>FileOutputStream(String name)</code>	<code>name</code> 이 지정하는 파일에 출력하는 <code>FileOutputStream</code> 생성
<code>FileOutputStream</code> <code>(File file, boolean append)</code>	<code>FileOutputStream</code> 을 생성하며 <code>append</code> 가 <code>true</code> 이면 <code>file</code> 이 지정하는 파일의 마지막부터 데이터 저장
<code>FileOutputStream</code> <code>(String name, boolean append)</code>	<code>FileOutputStream</code> 을 생성하며 <code>append</code> 가 <code>true</code> 이면 <code>name</code> 이름의 파일의 마지막부터 데이터 저장

메소드	설명
<code>void write(int b)</code>	<code>int</code> 형으로 넘겨진 한 바이트를 출력 스트림으로 출력
<code>void write(byte[] b)</code>	배열 <code>b</code> 의 바이트를 모두 출력 스트림으로 출력
<code>void write(byte[] b, int off, int len)</code>	<code>len</code> 크기만큼 <code>off</code> 부터 배열 <code>b</code> 를 출력 스트림으로 출력
<code>void flush()</code>	출력 스트림에서 남아 있는 데이터 모두 출력
<code>void close()</code>	출력 스트림을 닫고 관련된 시스템 자원 해제

예제 13-3 : FileOutputStream으로 바이너리 파일 쓰기

49

FileOutputStream을 이용하여 byte [] 배열 속에 들어 있는 바이너리 값을 c:\test.out 파일에 저장하라. 이 파일은 바이너리 파일이 된다. 이 파일은 예제 13-4에서 읽어 출력할 것이다.

```
import java.io.*;

public class FileOutputStreamEx {
    public static void main(String[] args) {
        byte b[] = {7,51,3,4,-1,24};

        try {
            FileOutputStream fout = new FileOutputStream("c:\test.out");
            for(int i=0; i<b.length; i++)
                fout.write(b[i]); // 배열 b의 바이너리를 그대로 기록

            fout.close();
        } catch(IOException e) { }
        System.out.println("c:\test.out을 저장하였습니다.");
    }
}
```

fout.write(b); 한 줄로 코딩 가능

c:\test.out을 저장하였습니다.

00000000h: 07 33 03 04 FF 18 ; 2 3 4 . ↑

test.out 파일 내부

바이트 스트림으로 바이너리 파일 읽기

50

- 바이너리 파일에서 읽기 위해 FileInputStream 클래스 이용

1. 파일 입력 스트림 생성(파일 열기)

- 스트림을 생성하고 파일을 열어 스트림과 연결

```
FileInputStream fin = new FileInputStream("c:\\test.out");
```

2. 파일 읽기

- read()로 문자 하나 씩 파일에서 읽기

```
int n=0, c;  
while((c = fin.read()) != -1) {  
    b[n] = (byte)c; // 읽은 바이트를 배열에 저장  
    n++;  
}
```

- 블록 단위로 읽기 가능

```
fin.read(b); // 배열 b의 바이트 크기만큼 바이너리 그대로 읽기
```

3. 스트림 닫기

- close()로 스트림 닫기

FileInputStream의 생성자와 주요 메소드

51

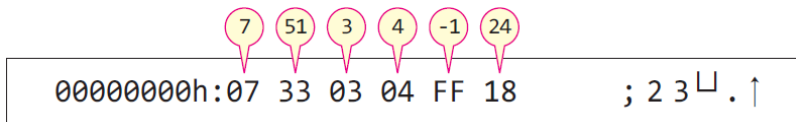
생성자	설명
<code>FileInputStream(File file)</code>	<code>file</code> 이 지정하는 파일로부터 읽는 <code>FileInputStream</code> 생성
<code>FileInputStream(String name)</code>	<code>name</code> 이 지정하는 파일로부터 읽는 <code>FileInputStream</code> 생성

메소드	설명
<code>int read()</code>	입력 스트림에서 한 바이트를 읽어 <code>int</code> 형으로 리턴
<code>int read(byte[] b)</code>	최대 배열 <code>b</code> 의 크기만큼 바이트를 읽음. 읽는 도중 EOF를 만나면 실제 읽은 바이트 수 리턴
<code>int read(byte[] b, int off, int len)</code>	최대 <code>len</code> 개의 바이트를 읽어 <code>b</code> 배열의 <code>off</code> 위치에 저장. 읽는 도중 EOF를 만나면 실제 읽은 바이트 수 리턴
<code>int available()</code>	입력 스트림에서 현재 읽을 수 있는 바이트 수 리턴
<code>void close()</code>	입력 스트림을 닫고 관련된 시스템 자원 해제

예제 13-4 : FileInputStream으로 바이너리 파일 읽기

52

FileInputStream을 이용하여 c:\test.out 파일(예제 13-3에서 저장한 파일)을 읽어 바이너리 값들을 byte [] 배열 속에 저장하고 화면에 출력하라.



test.out 파일 내부

```
import java.io.*;
public class FileInputStreamEx {
    public static void main(String[] args) {
        byte b[] = new byte [6]; // 비어 있는 byte 배열
        try {
            FileInputStream fin = new FileInputStream("c:\\test.out");
            int n=0, c;
            while((c = fin.read())!= -1) {
                b[n] = (byte)c; // 읽은 바이트를 배열에 저장
                n++;
            }

            System.out.println("c:\\test.out에서 읽은 배열을 출력합니다.");
            for(int i=0; i<b.length; i++)
                System.out.print(b[i]+" ");
            System.out.println();

            fin.close();
        } catch(IOException e) { }
    }
}
```

c:\\test.out에서 읽은 배열을 출력합니다.
7 51 3 4 -1 24

File 클래스

53

□ File 클래스

- ▣ 파일의 경로명 및 속성을 다루는 클래스
 - java.io.File
 - 파일과 디렉터리 경로명의 추상적 표현
- ▣ 파일 이름 변경, 삭제, 디렉터리 생성, 크기 등 파일 관리
- ▣ File 객체에는 파일 읽기/쓰기 기능 없음
 - 파일 입출력은 파일 입출력 스트림 이용

□ File 객체 생성

- ▣ 생성자에 파일 경로명을 주어 File 객체 생성

```
File f = new File("c:\\tmp\\test.txt");
```

- ▣ 디렉터리와 파일명을 나누어 생성자 호출

```
File f = new File("c:\\tmp", "test.txt");
```

File 클래스 생성자와 주요 메소드

메소드	설명
<code>File(File parent, String child)</code>	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
<code>File(String pathname)</code>	pathname의 완전 경로명이 나타내는 File 객체 생성
<code>File(String parent, String child)</code>	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
<code>File(URI uri)</code>	file:URI를 추상 경로명으로 변환하여 File 객체 생성

메소드	설명
<code>boolean mkdir()</code>	새로운 디렉터리 생성
<code>String[] list()</code>	디렉터리 내의 파일과 서브 디렉터리 리스트를 문자열 배열로 리턴
<code>File [] listFiles()</code>	디렉터리 내의 파일과 서브 디렉터리 리스트를 File [] 배열로 리턴
<code>boolean renameTo(File dest)</code>	dest가 지정하는 경로명으로 파일 이름 변경
<code>boolean delete()</code>	파일 또는 디렉터리 삭제
<code>long length()</code>	파일의 크기 리턴
<code>String getPath()</code>	경로명 전체를 문자열로 변환하여 리턴
<code>String getParent()</code>	파일이나 디렉터리의 부모 디렉터리 이름 리턴
<code>String getName()</code>	파일 또는 디렉터리 이름을 문자열로 리턴
<code>boolean isFile()</code>	일반 파일이면 true 리턴
<code>boolean isDirectory()</code>	디렉터리이면 true 리턴
<code>long lastModified()</code>	파일이 마지막으로 변경된 시간 리턴
<code>boolean exists()</code>	파일 또는 디렉터리가 존재하면 true 리턴

File 클래스 활용

55

- 파일 크기

```
long size = f.length();
```

- 파일 경로명

```
File f = new File("c:\\windows\\system.ini");  
String filename = f.getName();    // "system.ini"  
String path = f.getPath();        // "c:\\windows\\system.ini"  
String parent = f.getParent();    // "c:\\windows"
```

- 파일 타입

```
if(f.isFile())  
    System.out.println(f.getPath() + "는 파일입니다."); // 파일  
else if(f.isDirectory())  
    System.out.println(f.getPath() + "는 디렉터리입니다."); // 디렉터리
```

c:\\windows\\system.ini은 파일입니다.

- 디렉터리 파일
리스트 얻기

```
File f = new File("c:\\tmp");  
File[] subfiles = f.listFiles(); // c:\\tmp의 파일 및 서브 디렉터리 리스트 얻기  
  
for(int i=0; i<subfiles.length; i++) {  
    System.out.print(subfiles[i].getName()); // 서브 파일명 출력  
    System.out.println("파일 크기: " + subfiles[i].length()); //서브파일크기출력  
}
```

예제 13-5 : File 클래스를 활용한 파일 관리

56

File 클래스를 이용하여, 파일 타입 및 경로명 알아내기, 디렉터리 생성, 파일 이름 변경, 디렉터리의 파일 리스트 출력 등 다양한 파일 관리 사례를 보여준다..

```
import java.io.File;

public class FileClassExample {
    public static void listDirectory(File dir) {
        System.out.println("-----" + dir.getPath() + "의 서브 리스트 입니다.-----");
        File[] subFiles = dir.listFiles();

        for(int i=0; i<subFiles.length; i++) {
            File f = subFiles[i];
            long t = f.lastModified(); // 마지막으로 수정된 시간
            System.out.print(f.getName());
            System.out.print("\t파일 크기: " + f.length()); // 파일 크기
            System.out.printf("\t수정된 시간: %tb %td %ta %tT\n",t, t, t, t);
        }
    }

    public static void main(String[] args) {
        File f1 = new File("c:\\windows\\system.ini");
        System.out.println(f1.getPath() + ", " + f1.getParent() + ", " +
            f1.getName());

        String res="";
        if(f1.isFile()) res = "파일";
        else if(f1.isDirectory()) res = "디렉토리";
        System.out.println(f1.getPath() + "은 " + res + "입니다.");
    }
}
```

```
File f2 = new File("c:\\tmp\\java_sample");
if(!f2.exists()) {
    f2.mkdir();
}
listDirectory(new File("c:\\tmp"));
f2.renameTo(new
    File("c:\\tmp\\javasample"));
listDirectory(new File("c:\\tmp"));
}
```

```
c:\\windows\\system.ini, c:\\windows, system.ini
c:\\windows\\system.ini은 파일입니다.
-----c:\\tmp의 서브 리스트 입니다.-----
java_sample    파일 크기: 0      수정된 시간: 4월 28 월 18:31:02
song.txt       파일 크기: 20     수정된 시간: 5월 30 수 15:09:33
student.txt    파일 크기: 244    수정된 시간: 7월 08 월 21:46:47
telephone.txt  파일 크기: 14     수정된 시간: 5월 13 월 12:38:11
wtc2-02f.mid   파일 크기: 4100   수정된 시간: 4월 01 금 05:18:54
-----c:\\tmp의 서브 리스트 입니다.-----
javasample     파일 크기: 0      수정된 시간: 4월 28 월 18:31:02
song.txt       파일 크기: 20     수정된 시간: 5월 30 수 15:09:33
student.txt    파일 크기: 244    수정된 시간: 7월 08 월 21:46:47
telephone.txt  파일 크기: 14     수정된 시간: 5월 13 월 12:38:11
wtc2-02f.mid   파일 크기: 4100   수정된 시간: 4월 01 금 05:18:54
```

예제 13-6 : 텍스트 파일 복사

57

문자 스트림 `FileReader`와 `FileWriter`를 이용하여 `c:\windows\system.ini`를 `c:\tmp\system.txt` 파일로 복사하는 프로그램을 작성하라.

```
import java.io.*;

public class TextCopy {
    public static void main(String[] args){
        File src = new File("c:\\windows\\system.ini"); // 원본 파일 경로명
        File dest = new File("c:\\tmp\\system.txt"); // 복사 파일 경로명
        int c;
        try {
            FileReader fr = new FileReader(src); // 파일 입력 문자 스트림 생성
            FileWriter fw = new FileWriter(dest); // 파일 출력 문자 스트림 생성
            while((c = fr.read()) != -1) { // 문자 하나 읽고
                fw.write((char)c); // 문자 하나 쓰고
            }
            fr.close();
            fw.close();
            System.out.println( src.getPath()+ "를 " + dest.getPath()+ "로 복사하였습니다.");
        } catch (IOException e) {
            System.out.println("파일 복사 오류");
        }
    }
}
```

`c:\\windows\\system.ini`를 `c:\\tmp\\system.txt`로 복사하였습니다.

예제 13-7 : 바이너리 파일 복사

58

바이트 스트림 `FileInputStream`과 `FileOutputStream`을 이용하여 이미지 파일을 복사하라.

```
import java.io.*;

public class BinaryCopy {
    public static void main(String[] args) {
        File src = new File( "c:\\Users\\Public\\Pictures\\Sample Pictures\\desert.jpg");
        File dest = new File("c:\\tmp\\desert.jpg");
        int c;
        try {
            FileInputStream fi = new FileInputStream(src);
            FileOutputStream fo = new FileOutputStream(dest);
            while((c = fi.read()) != -1) {
                fo.write((byte)c);
            }
            fi.close();
            fo.close();
            System.out.println( src.getPath()+ "를 " + dest.getPath()+
                "로 복사하였습니다.");
        } catch (IOException e) {
            System.out.println("파일 복사 오류");
        }
    }
}
```

c:\\Users\\Public\\Pictures\\Sample Pictures\\desert.jpg를 c:\\tmp\\desert.jpg로 복사하였습니다.



예제 13-8 : 고속 복사를 위한 블록 단위 바이너리 파일 복사

59

예제 13-7을 10KB씩 읽고 쓰도록 수정하여 **고속으로** 파일을 복사하라.

```
import java.io.*;

public class BlockBinaryCopy {
    public static void main(String[] args) {
        File src = new File("c:\\Users\\Public\\Pictures\\Sample Pictures\\desert.jpg"); // 원본 파일
        File dest = new File("c:\\tmp\\desert.jpg"); // 복사 파일
        try {
            FileInputStream fi = new FileInputStream(src); // 파일 입력 바이트 스트림 생성
            FileOutputStream fo = new FileOutputStream(dest); // 파일 출력 바이트 스트림 생성

            byte [] buf = new byte [1024*10]; // 10KB 버퍼
            while(true) {
                int n = fi.read(buf); // 버퍼 크기만큼 읽기. n은 실제 읽은 바이트
                fo.write(buf, 0, n); // buf[0]부터 n 바이트 쓰기
                if(n < buf.length)
                    break; // 버퍼 크기보다 작게 읽었기 때문에 파일 끝에 도달. 복사 종료
            }
            fi.close();
            fo.close();
            System.out.println(src.getPath() + "를 " + dest.getPath() + "로 복사하였습니다.");
        } catch (IOException e) { System.out.println("파일 복사 오류"); }
    }
}
```

c:\\Users\\Public\\Pictures\\Sample Pictures\\desert.jpg를 c:\\tmp\\desert.jpg로 복사하였습니다.

학습 목표

1. 멀티태스킹과 스레드의 개념 이해
2. Thread 클래스를 상속받아 자바 스레드 만들기
3. Runnable 인터페이스를 구현하여 자바 스레드 만들기
4. 스레드 종료 시키기
5. 스레드의 동기화 개념과 필요성 이해
6. synchronized로 간단한 스레드 동기화
7. wait()-notify()로 간단한 스레드 동기화

멀티태스킹(multi-tasking) 개념

61

□ 멀티태스킹

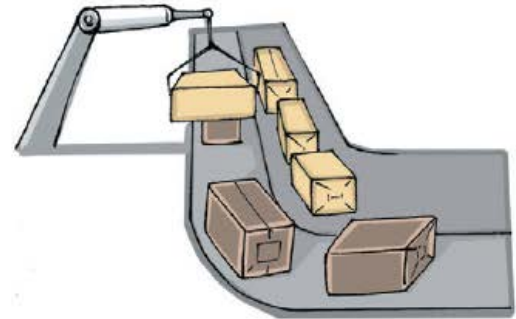
- ▣ 여러 개의 작업(태스크)이 동시에 처리되는 것



다림질하면서
전화 받는 주부



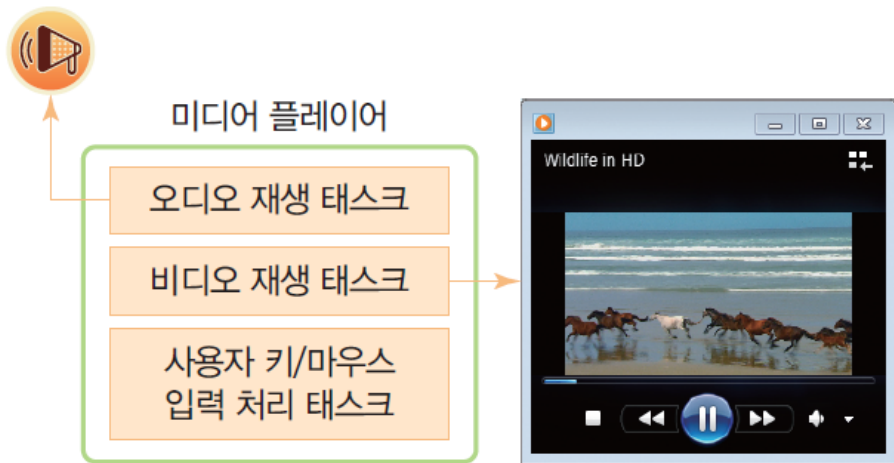
운전하면서 음악을
듣는 연수 양



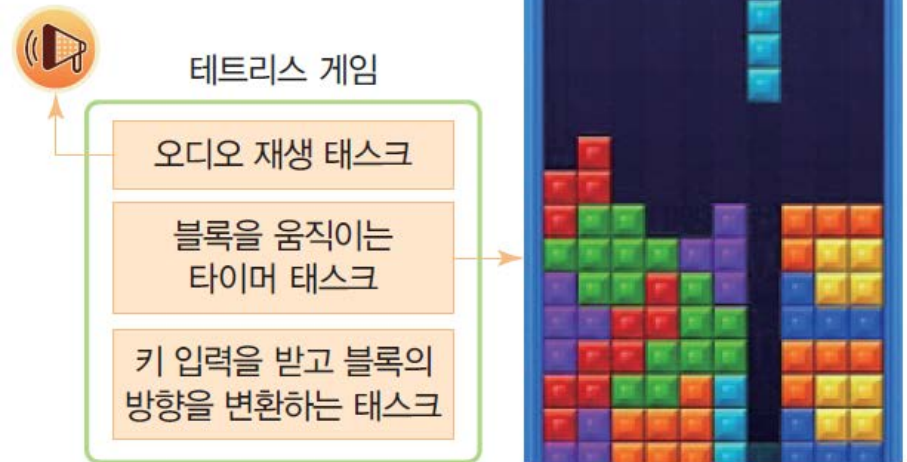
판독과 포장을
동시에 하는 기계

멀티태스킹 프로그램 사례

62



(a) 미디어 플레이어의 멀티태스킹



(b) 테트리스 게임의 멀티태스킹

스레드와 운영체제

63

- 스레드(thread)
 - ▣ 운영체제에 의해 관리되는 하나의 작업 혹은 태스크
 - ▣ 스레드와 태스크(혹은 작업)은 바꾸어 사용해도 무관

- 멀티스레딩(multi-threading)
 - ▣ 여러 스레드를 동시에 실행시키는 응용프로그램을 작성하는 기법

- 스레드 구성
 - ▣ 스레드 코드
 - 작업을 실행하기 위해 작성한 프로그램 코드
 - 개발자가 작성
 - ▣ 스레드 정보
 - 스레드 명, 스레드 ID, 스레드의 실행 소요 시간, 스레드의 우선 순위 등
 - 운영체제가 스레드에 대해 관리하는 정보

멀티태스킹과 멀티스레딩

64

□ 멀티태스킹 구현 기술

▣ 멀티프로세싱(multi-processing)

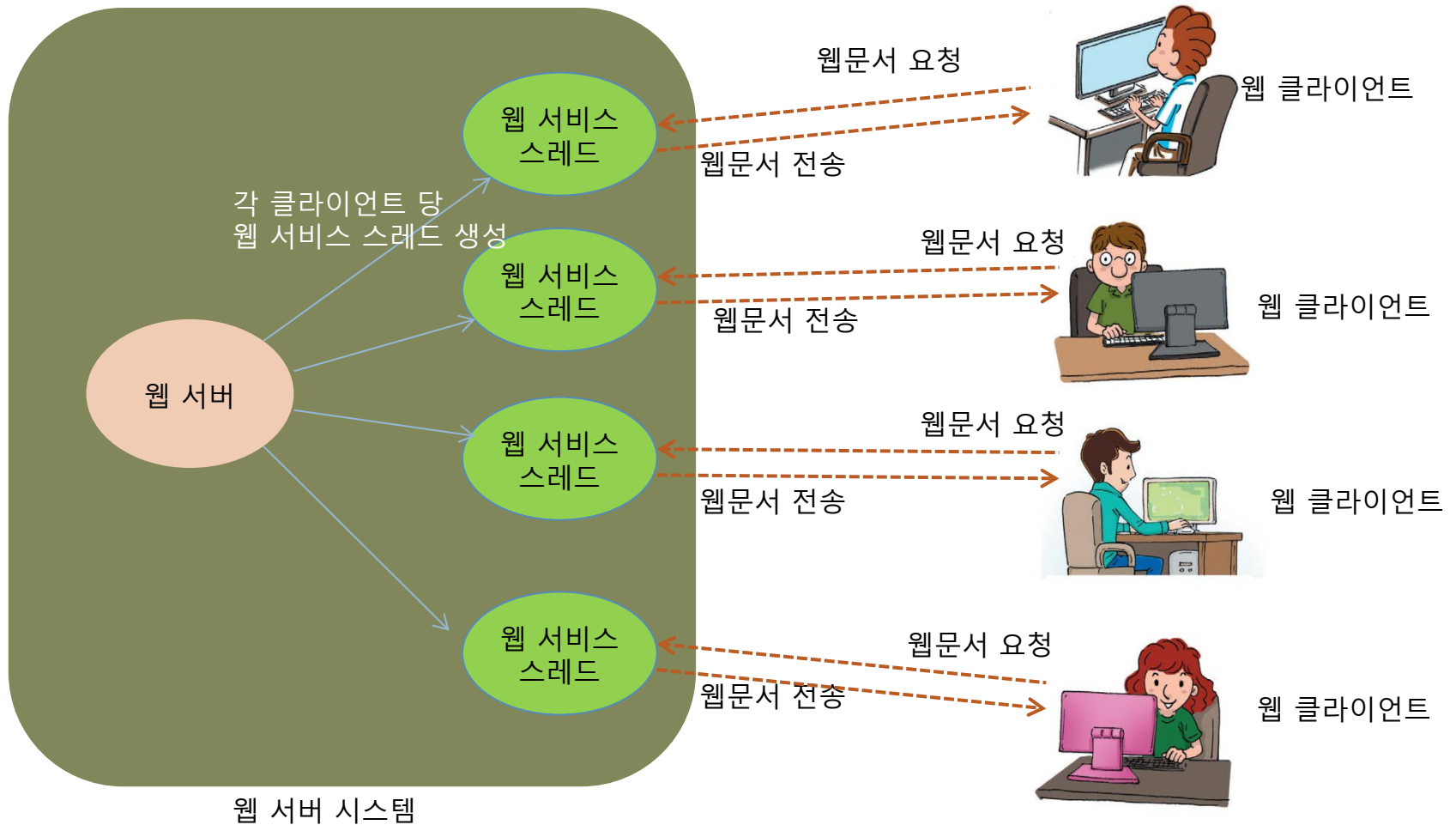
- 하나의 응용프로그램이 여러 개의 프로세스를 생성하고, 각 프로세스가 하나의 작업을 처리하는 기법
- 각 프로세스 독립된 메모리 영역을 보유하고 실행
- 프로세스 사이의 문맥 교환에 따른 과도한 오버헤드와 시간 소모의 문제점

▣ 멀티스레딩(multi-threading)

- 하나의 응용프로그램이 여러 개의 스레드를 생성하고, 각 스레드가 하나의 작업을 처리하는 기법
- 하나의 응용프로그램에 속한 스레드는 변수 메모리, 파일 오픈 테이블 등 자원으로 공유하므로, 문맥 교환에 따른 오버헤드가 매우 작음
- 현재 대부분의 운영체제가 멀티스레딩을 기본으로 하고 있음

웹 서버의 멀티스레딩 사례

65



자바 스레드(Thread)와 JVM

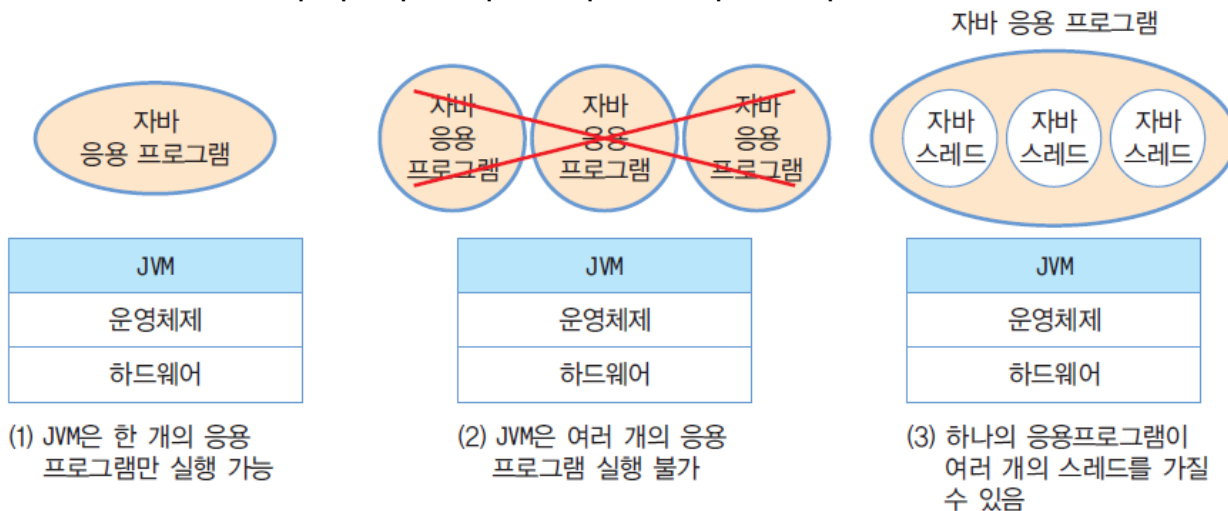
66

□ 자바 스레드

- 자바 가상 기계(JVM)에 의해 스케줄되는 실행 단위의 코드 블록
- 스레드의 생명 주기는 JVM에 의해 관리됨 : JVM은 스레드 단위로 스케줄링

□ JVM과 자바의 멀티스레딩

- 하나의 JVM은 하나의 자바 응용프로그램만 실행
 - 자바 응용프로그램이 시작될 때 JVM이 함께 실행됨
 - 자바 응용프로그램이 종료하면 JVM도 함께 종료함
- 응용프로그램은 하나 이상의 스레드로 구성 가능



자바 스레드 만들기

67

- 스레드 만드는 2 가지 방법
 - ▣ `java.lang.Thread` 클래스를 상속받아 스레드 작성
 - ▣ `java.lang.Runnable` 인터페이스를 구현하여 스레드 작성

Thread 클래스를 상속받아 스레드 만들기(1)

68

Thread 클래스의 주요 메소드

Thread의 메소드	내용
Thread() Thread(Runnable target) Thread(String name) Thread(Runnable target, String name)	스레드 객체 생성 Runnable 객체인 target을 이용하여 스레드 객체 생성 이름이 name인 스레드 객체 생성 Runnable 객체를 이용하여, 이름이 name인 스레드 객체 생성
void run()	스레드 코드로서 JVM에 의해 호출된다. 개발자는 반드시 오버라이딩하여 스레드 코드를 작성하여야 한다. 이 메소드가 종료하면 스레드도 종료
void start()	JVM에게 스레드 실행을 시작하도록 요청
void interrupt()	스레드 강제 종료
static void yield()	다른 스레드에게 실행을 양보한다. 이때 JVM은 스레드 스케줄링을 시행하며 다른 스레드를 선택하여 실행시킨다.
void join()	스레드가 종료할 때까지 기다린다.
long getId()	스레드의 ID 값 리턴
String getName()	스레드의 이름 리턴
int getPriority()	스레드의 우선순위 값 리턴. 1에서 10 사이 값
void setPriority(int n)	스레드의 우선순위 값을 n으로 변경
Thread.State getState()	스레드의 상태 값 리턴
static void sleep(long mills)	스레드는 mills 시간 동안 잔다. mills의 단위는 밀리초
static Thread currentThread()	현재 실행 중인 스레드 객체의 레퍼런스 리턴

Thread 클래스를 상속받아 스레드 만들기(2)

69

- Thread를 상속받아 run() 오버라이딩
 - ▣ Thread 클래스 상속. 새 클래스 작성
 - ▣ run() 메소드 작성
 - run() 메소드를 스레드 코드라고 부름
 - run() 메소드에서 스레드 실행 시작
- 스레드 객체 생성
 - ▣ 생성된 객체는 필드와 메소드를 가진 객체일 뿐 스레드로 작동하지 않음
- 스레드 시작
 - ▣ start() 메소드 호출
 - 스레드로 작동 시작
 - 스레드 객체의 run()이 비로소 실행
 - JVM에 의해 스케줄되기 시작함

```
class TimerThread extends Thread {  
    .....  
  
    public void run() { // run() 오버라이딩  
        .....  
    }  
}
```

```
TimerThread th = new TimerThread();
```

```
th.start();
```

Thread를 상속받아 1초 단위로 초 시간을 출력하는 TimerThread 스레드 작성 사례

70

스레드 클래스 선언

```
class TimerThread extends Thread {  
    int n = 0;
```

스레드 코드 작성

```
    public void run() {  
        while(true) { // 무한루프를 실행한다.  
            System.out.println(n);  
            n++;  
            try {  
                sleep(1000); //1초 동안 잠을 잔 후 깨어난다.  
            }  
            catch(InterruptedException e){return;}  
        }  
    }  
}
```

1초에 한 번씩
n을 증가시켜 콘솔에
출력한다.

스레드 객체 생성

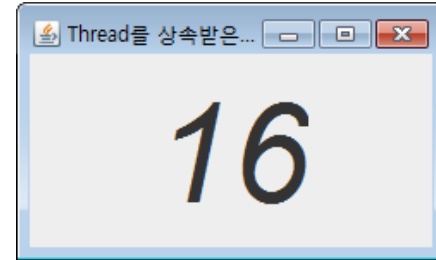
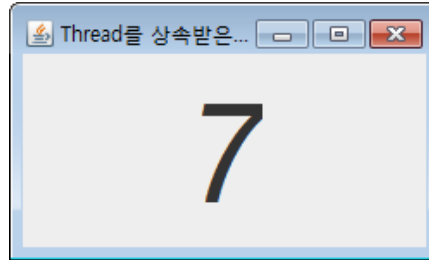
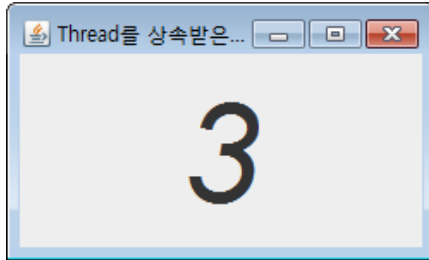
```
public class TestThread {  
    public static void main(String [] args) {  
        TimerThread th = new TimerThread();  
        th.start();  
    }  
}
```

스레드 시작

0
1
2
3
4
.
.
.

예제 12-1 : Thread를 상속받아 1초 단위 타이머 스레드 만들기

71



```
import java.awt.*;
import javax.swing.*;

class TimerThread extends Thread {
    JLabel timerLabel; // 타이머 값이 출력되는 레이블
    public TimerThread(JLabel timerLabel) {
        this.timerLabel = timerLabel;
    }

    // 스레드 코드. run()이 종료하면 스레드 종료
    public void run() {
        int n=0; // 타이머 카운트 값
        while(true) { // 무한 루프
            timerLabel.setText(Integer.toString(n));
            n++; // 카운트 증가
            try {
                Thread.sleep(1000); // 1초동안 잠을 잔다.
            }
            catch(InterruptedException e) { return;}
        }
    }
}
```

```
public class ThreadTimerEx extends JFrame {
    public ThreadTimerEx() {
        setTitle("Thread를 상속받은 타이머 스레드 예제");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());

        // 타이머 값을 출력할 레이블 생성
        JLabel timerLabel = new JLabel();
        timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));
        c.add(timerLabel);

        TimerThread th = new TimerThread(timerLabel);
        setSize(250,150);
        setVisible(true);
        th.start(); // 타이머 스레드의 실행을 시작하게 한다.
    }

    public static void main(String[] args) {
        new ThreadTimerEx();
    }
}
```

Runnable 인터페이스로 스레드 만들기

72

□ Runnable 인터페이스 구현하는 새 클래스 작성

▣ run() 메소드 구현

- run() 메소드를 스레드 코드라고 부름
- run() 메소드에서 스레드 실행 시작

```
class TimerRunnable implements Runnable {  
    .....  
  
    public void run() { // run() 메소드 구현  
        .....  
    }  
}
```

□ 스레드 객체 생성

```
Thread th = new Thread(new TimerRunnable());
```

□ 스레드 시작

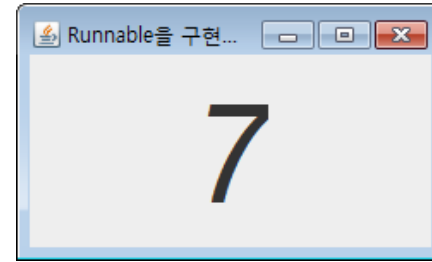
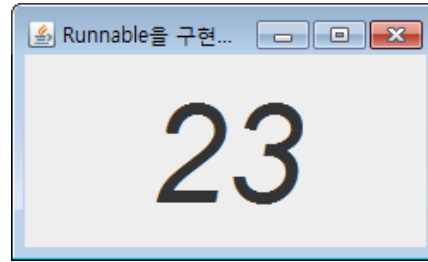
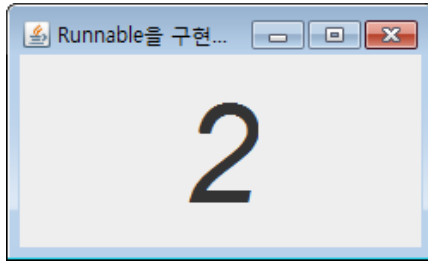
▣ start() 메소드 호출

- 스레드로 작동 시작
- 스레드 객체의 run()이 비로소 실행
- JVM에 의해 스케줄되기 시작함

```
th.start();
```

예제 12-2 : Runnable 인터페이스를 이용하여 1초 단위로 출력하는 타이머 스레드 만들기

73



```
import java.awt.*;
import javax.swing.*;

class TimerRunnable implements Runnable {
    JLabel timerLabel;
    public TimerRunnable(JLabel timerLabel) {
        this.timerLabel = timerLabel;
    }

    // 스레드 코드. run()이 종료하면 스레드 종료
    public void run() {
        int n=0; // 타이머 카운트 값
        while(true) { // 무한 루프
            timerLabel.setText(Integer.toString(n));
            n++;
            try {
                Thread.sleep(1000); // 1초동안 잠을 잔다.
            }
            catch(InterruptedException e) { return; }
        }
    }
}
```

```
public class RunnableTimerEx extends JFrame {
    public RunnableTimerEx() {
        setTitle("Runnable을 구현한 타이머 스레드 예제");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());

        // 타이머 값을 출력할 레이블 생성
        JLabel timerLabel = new JLabel();
        timerLabel.setFont(new Font("Gothic", Font.ITALIC, 80));
        c.add(timerLabel); // 레이블을 콘텐츠팬에 부착

        TimerRunnable runnable = new TimerRunnable(timerLabel);
        Thread th = new Thread(runnable); // 스레드 객체 생성
        setSize(250,150);
        setVisible(true);
        th.start(); // 타이머 스레드가 실행을 시작하게 한다.
    }

    public static void main(String[] args) {
        new RunnableTimerEx();
    }
}
```


main 스레드

74

- main 스레드
 - ▣ JVM이 응용프로그램을 실행할 때 디폴트로 생성되는 스레드
 - main() 메소드 실행 시작
 - main() 메소드가 종료하면 main 스레드 종료

예제 12-3 : main 스레드 확인과 스레드 정보를 알아내는 코드

75

main() 메소드 내에서 현재 스레드 정보를 가진 Thread 객체를 알아내어 현재 실행 중인 스레드에 관한 다양한 정보를 출력한다.

```
public class ThreadMainEx {  
    public static void main(String [] args) {  
        long id = Thread.currentThread().getId();  
        String name = Thread.currentThread().getName();  
        int priority = Thread.currentThread().getPriority();  
        Thread.State s = Thread.currentThread().getState();  
  
        System.out.println("현재 스레드 이름 = " + name);  
        System.out.println("현재 스레드 ID = " + id);  
        System.out.println("현재 스레드 우선순위 값 = " + priority);  
        System.out.println("현재 스레드 상태 = " + s);  
    }  
}
```

```
현재 스레드 이름 = main  
현재 스레드 ID = 1  
현재 스레드 우선순위 값 = 5  
현재 스레드 상태 = RUNNABLE
```

스레드 종료와 타 스레드 강제 종료

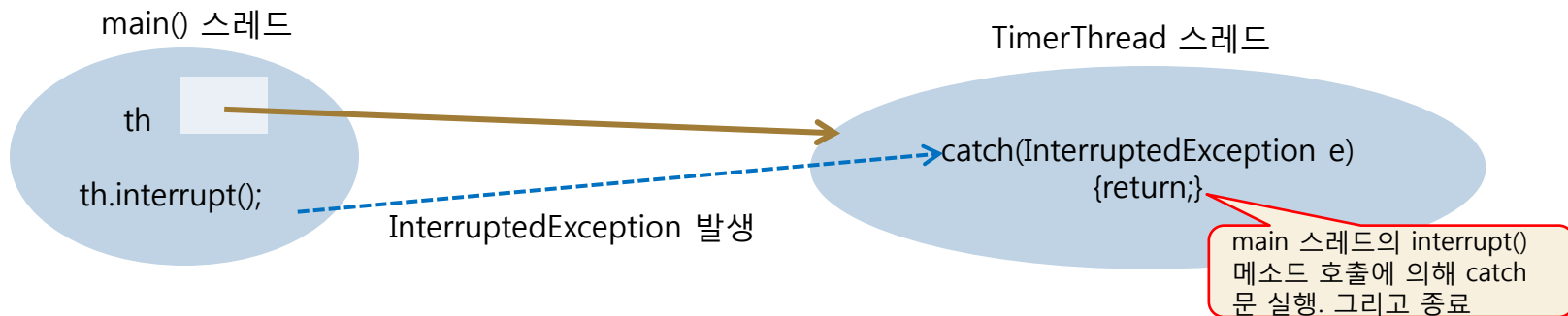
76

- 스스로 종료
 - ▣ run() 메소드 리턴
- 타 스레드에서 강제 종료
 - ▣ interrupt() 메소드 사용

```
public static void main(String [] args) {  
    TimerThread th = new TimerThread();  
    th.start();  
  
    th.interrupt(); // TimerThread 강제 종료  
}
```

```
class TimerThread extends Thread {  
    int n = 0;  
    public void run() {  
        while(true) {  
            System.out.println(n); // 화면에 카운트 값 출력  
            n++;  
            try {  
                sleep(1000);  
            }  
            catch(InterruptedException e){  
                return; // 예외를 받고 스스로 리턴하여 종료  
            }  
        }  
    }  
}
```

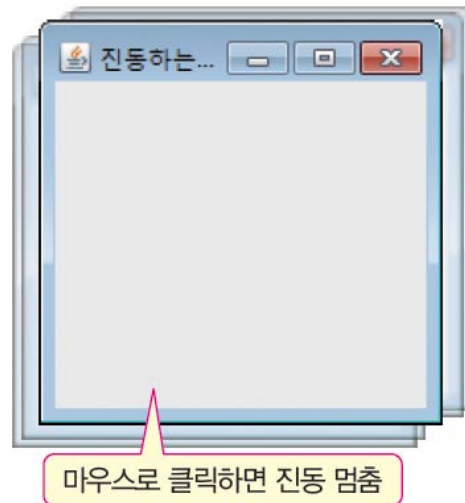
return 하지 않으면
스레드는 종료하지 않음



예제 12-4 : 진동하는 스레드와 스레드의 강제 종료

77

Runnable을 받은 스레드를 작성하여 프레임이 심하게 진동하도록 프로그램을 작성하라. 그리고 콘텐츠팬에 마우스를 클릭하면 진동 스레드를 종료시켜 진동이 멈추도록 하라



예제 12-4 정답

78

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;
```

```
public class VibratingFrame extends JFrame implements Runnable
```

```
{
    Thread th; // 진동하는 스레드
    public VibratingFrame() {
        setTitle("진동하는 프레임 만들기");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200,200);
        setLocation(300,300);
        setVisible(true);
    }
}
```

```
    getContentPane().addMouseListener(new MouseAdapter() {
        public void mousePressed(MouseEvent e) {
            if(!th.isAlive()) return;
            th.interrupt();
        }
    });
```

```
    th = new Thread(this); // 진동하는 스레드 객체 생성
    th.start(); // 진동 시작
}
```

Runnable 인터페이스 구현. 프레임에
run() 메소드 반드시 작성 필요

프레임 객체가 Runnable 인터페이스를
구현한 객체이므로 this 가능

```
public void run() { // 프레임의 진동을 일으키기 위해
    // 20ms마다 프레임의 위치를 랜덤하게 이동
    Random r = new Random();
    while(true) {
        try {
            Thread.sleep(20); // 20ms 잠자기
        }
        catch (InterruptedException e){
            return; // 리턴하면 스레드 종료
        }
        int x = getX() + r.nextInt()%5; // 새 위치 x
        int y = getY() + r.nextInt()%5; // 새 위치 y
        setLocation(x, y); // 프레임의 위치 이동
    }
}

public static void main(String [] args) {
    new VibratingFrame();
}
```

스레드 동기화(Thread Synchronization)

79

- 멀티스레드 프로그램 작성시 주의점
 - ▣ 다수의 스레드가 공유 데이터에 동시에 접근하는 경우
 - 공유 데이터의 값에 예상치 못한 결과 발생 가능
- 스레드 동기화
 - ▣ 동기화란?
 - 스레드 사이의 실행순서 제어, 공유데이터에 대한 접근을 원활하게 하는 기법
 - ▣ 멀티스레드의 공유 데이터의 동시 접근 문제 해결
 - 방법1) 공유 데이터를 접근하는 모든 스레드의 한 줄 세우기
 - 방법2) 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 대기 하도록 함
- 자바의 스레드 동기화 방법 - 2가지
 - ▣ synchronized 키워드로 동기화 블록 지정
 - ▣ wait()-notify() 메소드로 스레드의 실행 순서 제어

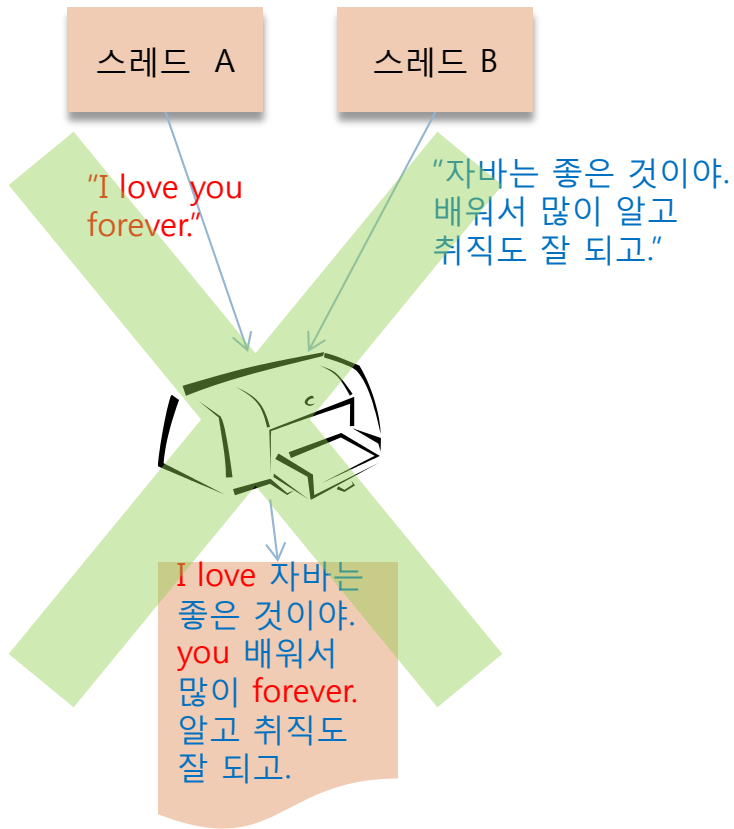
동기화의 필요성 - 두 스레드가 프린터에 동시 쓰기로 충돌하는 경우

80

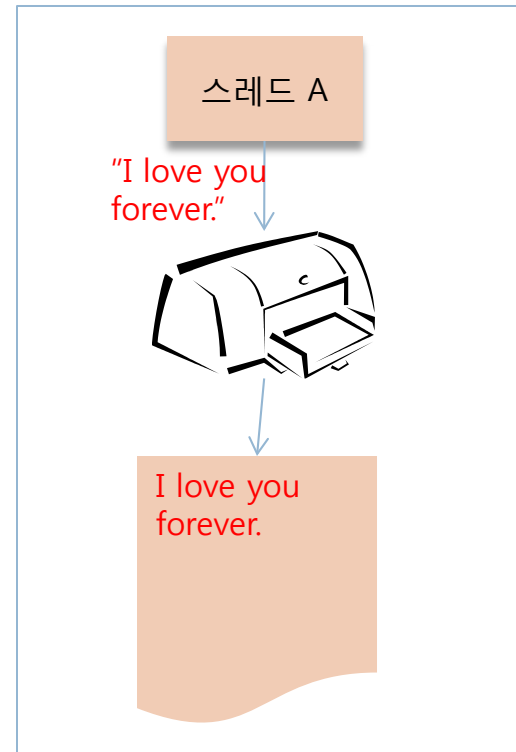
스레드 B

"자바는 좋은 것이야.
배워서 많이 알고
취직도 잘 되고."

스레드 A가 프린터 사용을 끝낼때까지 기다린다.



두 스레드가 동시에 프린터에 쓰는 경우
문제 발생



한 스레드의 출력이 끝날 때까지
다른 스레드 대기함으로써
정상 출력

synchronized 블록 지정

81

- synchronized 키워드
 - ▣ 스레드가 독점적으로 실행해야 하는 부분(동기화 코드)을 표시하는 키워드
 - 임계 영역(critical section) 표기 키워드
 - ▣ synchronized 블록 지정 방법
 - 메소드 전체 혹은 코드 블록
- synchronized 블록이 실행될 때,
 - ▣ 먼저 실행한 스레드가 모니터 소유
 - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
 - ▣ 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드 대기

```
synchronized void print(String text) { // 동기화 메소드
    ...
    for(int i=0; i<text.length(); i++) // text의 각 문자 출력
        System.out.print(text.charAt(i));
    ...
}
```

synchronized 메소드

```
void execute(String text) {
    ...
    synchronized(this) { // 동기화 코드 블록
        ...
        for(int i=0; i<text.length(); i++)
            System.out.print(text.charAt(i));
        ...
    }
}
```

synchronized 코드 블록

예제 12-5 : 두 스레드가 공유 프린터 객체를 통해 동시에 출력하는 경우 동기화 - synchronized 블록 지정

82

```
public class SynchronizedEx {
    public static void main(String[] args) {
        SharedPrinter p = new SharedPrinter(); // 공유 데이터 생성
        String [] engText = { "Wise men say, ",
                                "only fools rush in",
                                "But I can't help, ",
                                "falling in love with you",
                                "Shall I stay? ",
                                "Would it be a sin?",
                                "If I can't help, ",
                                "falling in love with you" };
        String [] korText = { "동해물과 백두산이 마르고 닳도록, ",
                                "하느님이 보우하사 우리 나라 만세",
                                "무궁화 삼천리 화려강산, ",
                                "대한 사람 대한으로 길이 보전하세",
                                "남산 위에 저 소나무, 철갑을 두른 듯",
                                "바람서리 불변함은 우리 기상일세.",
                                "무궁화 삼천리 화려강산, ",
                                "대한 사람 대한으로 길이 보전하세" };

        Thread th1 = new WorkerThread(p, engText);//영문출력스레드
        Thread th2 = new WorkerThread(p, korText);//국문출력스레드

        // 두 스레드를 실행시킨다.
        th1.start();
        th2.start();
    }
}
```

```
// 두 WorkerThread 스레드에 의해 동시 접근되는 공유 프린터
class SharedPrinter {
    // synchronized를 생략하면
    // 한글과 영어가 한 줄에 섞여 출력되는 경우가 발생한다.
    synchronized void print(String text) {
        // Thread.yield();
        for(int i=0; i<text.length(); i++)
            System.out.print(text.charAt(i));
        System.out.println();
    }
}

// 스레드 클래스
class WorkerThread extends Thread {
    SharedPrinter p; // 공유 프린터 주소
    String [] text;
    WorkerThread(SharedPrinter p, String[] text) {
        this.p = p; this.text = text;
    }

    // 스레드는 반복적으로 공유 프린터에 10번 접근 text[] 출력
    public void run() {
        for (int i=0; i<text.length; i++) // 한 줄씩 출력
            p.print(text[i]); // 공유 프린터에 출력
    }
}
```

예제 12-5 실행 결과

83

한글이 한 줄 출력되든지 영문이 한 줄 출력되는 것이 정상이지만 synchronized가 생략된 print() 메소드가 두 스레드에 의해 동시 호출되면 두 스레드의 동기화가 이루어지지 않아서 한글과 영문이 섞여 출력된다.

Wise men say,
only fools rush in
But I can't help,
falling in love with you
Shall I stay?
Would it be a sin?
If I can't help,
falling in love with you
동해물과 백두산이 마르고 닳도록,
하느님이 보우하사 우리 나라 만세
무궁화 삼천리 화려강산,
대한 사람 대한으로 길이 보전하세
남산 위에 저 소나무, 철갑을 두른 듯
바람서리 불변함은 우리 기상일세.
무궁화 삼천리 화려강산,
대한 사람 대한으로 길이 보전하세

라인 31에 synchronized로 선언한 경우

Wise 동해물과 백두산이 마르고 닳도록, men say,
only fools rush in
But I can't help,
하느님이 보우하사 우리 나라 만세
falling in love with you
무궁화 삼천리 Shall I stay?
화려강산,
Would it be a sin?
대한 사람 대한으로 길이 보전하세
If I can't help,
남산 위에 저 소나무, 철갑을 두른 듯
falling in love바람서리 불변함은 우리 기상일세.
with you
무궁화 삼천리 화려강산,
대한 사람 대한으로
길이 보전하세

print() 메소드 충돌

print() 메소드 충돌

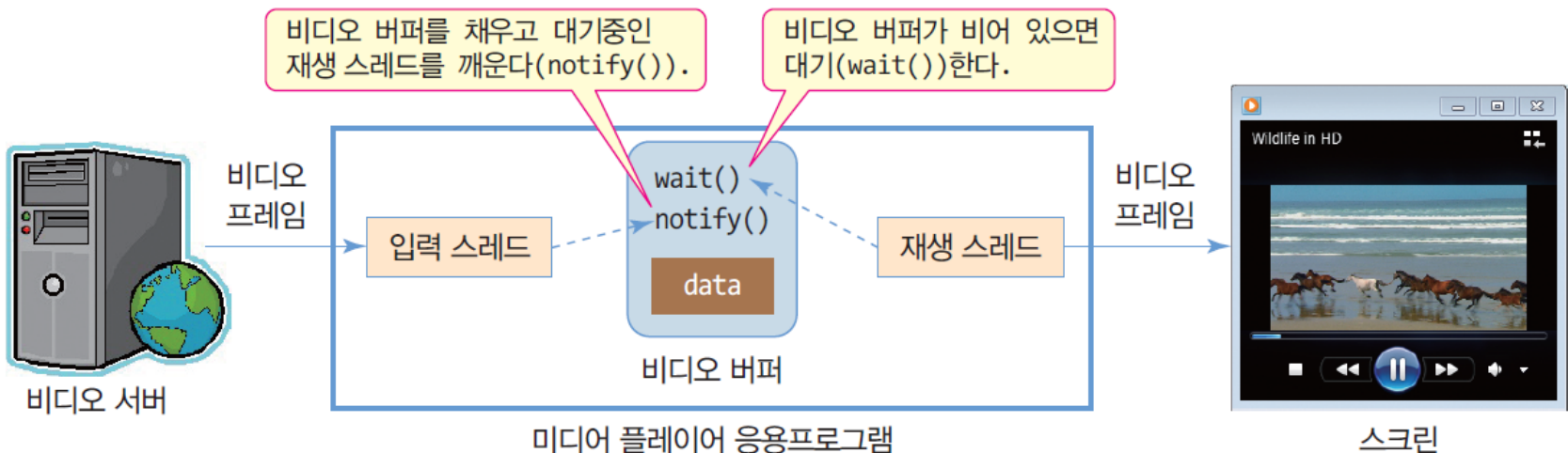
print() 메소드 충돌

라인 31에 synchronized를 생략한 경우

wait()-notify()를 이용한 스레드 동기화

84

- wait()-notify()가 필요한 경우
 - ▣ 공유 데이터로 두 개 이상의 스레드가 데이터를 주고 받을 때
 - producer-consumer문제
- 동기화 메소드
 - ▣ wait() : 다른 스레드가 notify()를 불러줄 때까지 기다린다.
 - ▣ notify() : wait()를 호출하여 대기중인 스레드를 깨운다.
 - wait(), notify()는 Object의 메소드

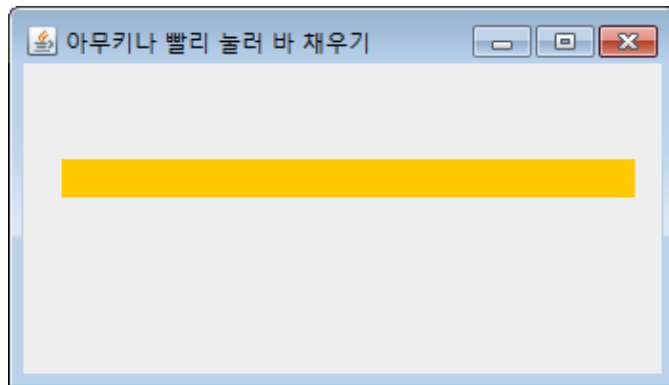


예제 12-6 : wait(), notify()를 이용한 바 채우기

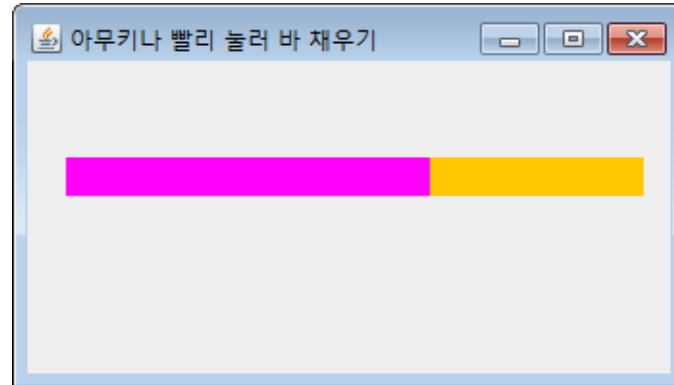
85

다음 설명과 같이 작동하는 스윙 프로그램을 작성하라.

아래 그림에는 스레드를 가진 bar가 있다. 아무 키나 누르면 bar에 마젠타색이 오른쪽으로 1/100씩 채워진다. 가만히 있으면 스레드에 의해 0.1초 간격으로 bar의 마젠타색을 1/100씩 감소시킨다. 키를 빨리 누르지 않으면 스레드의 감소 속도를 이기지 못한다. bar는 JLabel을 상속받은 MyLabel로 작성하고 MyLabel의 paintComponent() 메소드가 bar를 마젠타색으로 채우도록 하라.



초기 화면



키를 반복하여 빨리 누른 화면

예제 12-6 정답

86

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class MyLabel extends JLabel {
    int barSize = 0; // 바의 크기
    int maxBarSize;

    MyLabel(int maxBarSize) {
        this.maxBarSize = maxBarSize;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.MAGENTA);
        int width = (int)((double)(this.getWidth())
            /maxBarSize*barSize);
        if(width==0) return;
        g.fillRect(0, 0, width, this.getHeight());
    }

    synchronized void fill() {
        if(barSize == maxBarSize) {
            try {
                wait();
            } catch (InterruptedException e) { return; }
        }
        barSize++;
        repaint(); // 바 다시 그리기
        notify();
    }
}
```

```
synchronized void consume() {
    if(barSize == 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            return;
        }
        barSize--;
        repaint(); // 바 다시 그리기
        notify();
    }
}

class ConsumerThread extends Thread {
    MyLabel bar;

    ConsumerThread(MyLabel bar) {
        this.bar = bar;
    }

    public void run() {
        while(true) {
            try {
                sleep(200);
                bar.consume();
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}
```

```
public class TabAndThreadEx extends
JFrame {
    MyLabel bar = new MyLabel(100);
    TabAndThreadEx(String title) {
        super(title);
        this.setDefaultCloseOperation
            (JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(null);
        bar.setBackground(Color.ORANGE);
        bar.setOpaque(true);
        bar.setLocation(20, 50);
        bar.setSize(300, 20);
        c.add(bar);

        c.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e)
            {
                bar.fill();
            }
        });
        setSize(350,200);
        setVisible(true);

        c.requestFocus();
        ConsumerThread th = new
            ConsumerThread(bar);
        th.start(); // 스레드 시작
    }

    public static void main(String[] args) {
        new TabAndThreadEx(
            "아무키나 빨리 눌러 바 채우기");
    }
}
```