

# 자바 클래스 개요

한국능력개발원

지도교수 : 박인상

QR 코드



# 자바의 특징

---

- 자바가 프로그래밍 언어임에도 단순한 언어의 차원을 넘어 자바 컴퓨팅(Java Computing)으로 까지 불려지고 있는 것은
  - 방대한 API(Application Programming Interface)와 라이브러리가 있기 때문이다.
-

# 자바의 특성

---

- 단순성
  - 유연성
  - 확장성
  - 안전성
  - 효율성
-

# 자바의 데이터 타입(data type)

---

## □ 기본 타입 : 8개

- 정수 타입 – byte(1), short(2), int(4), long(8)
- 실수 타입 – float(4), double(8)
- 문자 타입 – char(2byte, Unicode)
- 논리 타입 – boolean(1byte, true 또는 false)

\* 단, 기본타입으로는 자바에서 문자열을 표현할 수 없다  
String 클래스를 이용하여 문자열을 표시할 수 있다

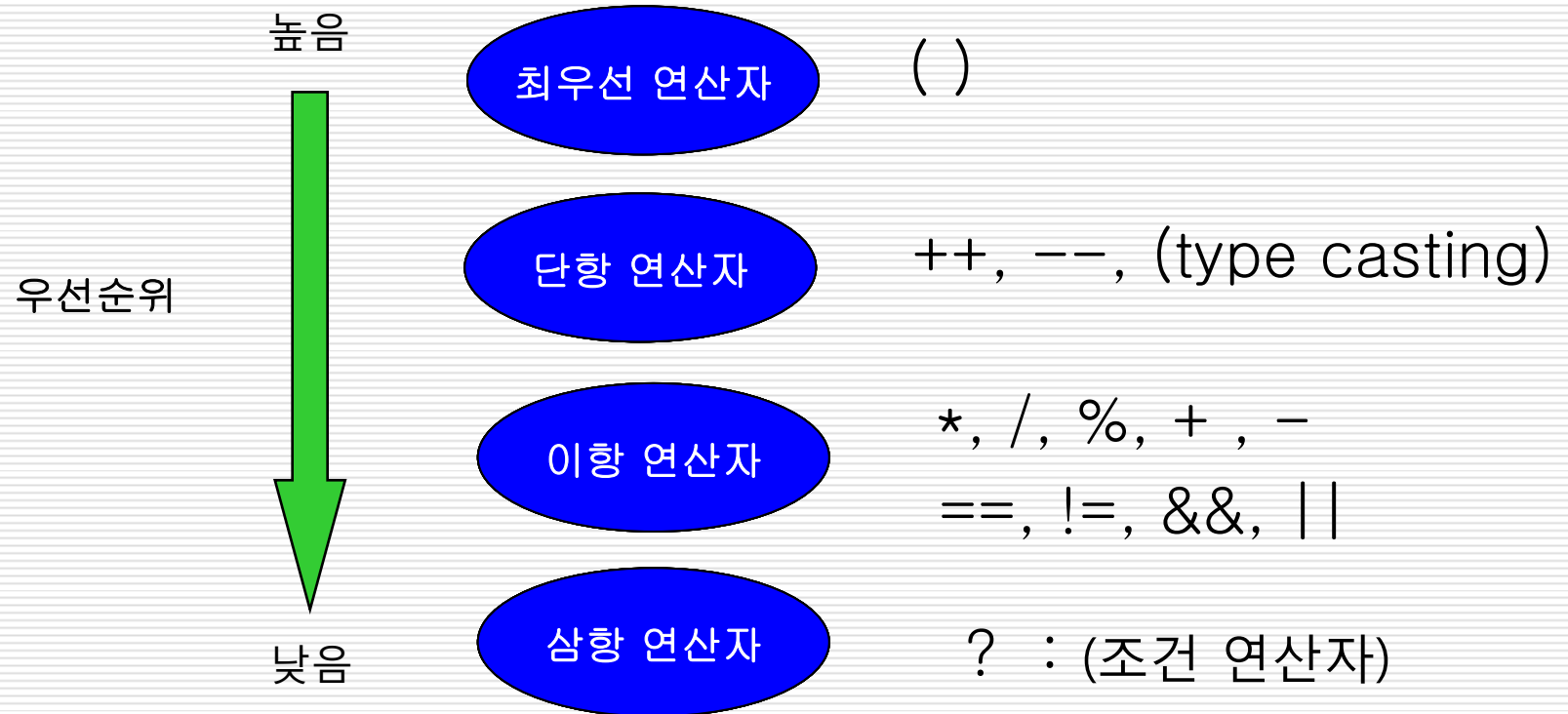
## □ 레퍼런스 타입 : 1개

레퍼런스 타입은 한 가지이지만 용도는 3가지이다

- 배열에 대한 레퍼런스
  - 클래스에 대한 레퍼런스
  - 인터페이스에 대한 레퍼런스
-

# 연산자의 종류와 우선순위

---



# 제어문

---

- (1) 조건문 – if문, if-else문, if-else if-else문, switch문
  - (2) 반복문 – for문, 확장for문, while문, do-while문
  - (3) 분기문 – continue문, break문, return문
-

# 다중 if문

---

<형식>

```
if(조건식 1) {  
    실행문장 1;  
}  
else if(조건식 2) {  
    실행문장 2;  
}  
else if(조건식 m) {  
    실행문장 m;  
}  
else {  
    실행문장 n;  
}
```

---

# switch문

---

<형식>

```
switch(식) {  
    case 값1:  
        실행문장 1;  
        break;  
    case 값2:  
        실행문장 2;  
        break;  
    case 값m:  
        실행문장 m;  
        break;  
    default:  
        실행문장 n;  
}
```

---



# continue문과 break문

---

## □ continue문

- 반복을 한 번 건너뛰기 위해 사용
- for문 안에서 continue문을 만나면 제어가 **증가**로 가고,  
while문 안에서 continue문을 만나면 제어가 **조건**으로 간다

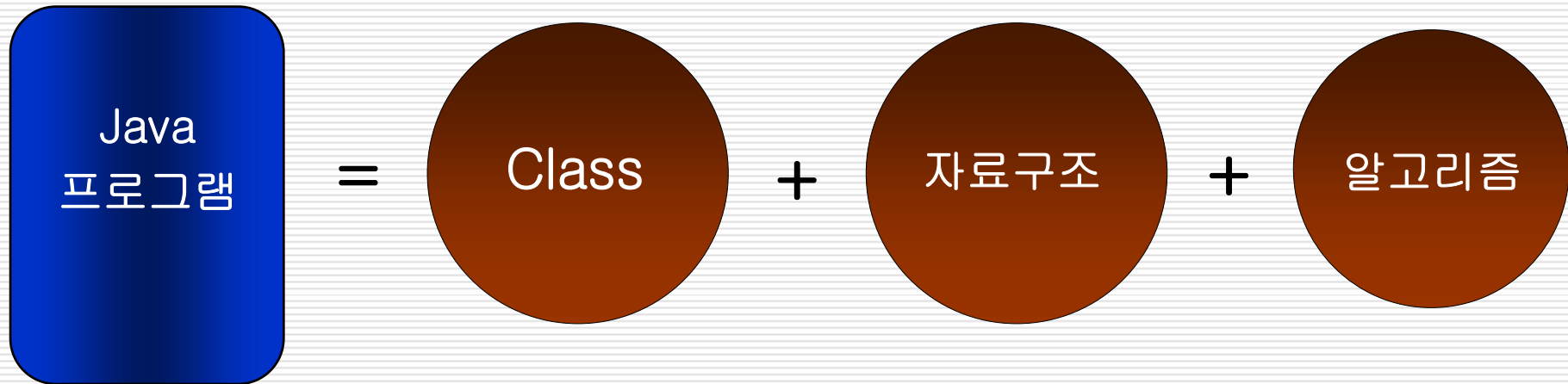
## □ break문

반복문을 벗어나기 위해 사용

---

# Java 프로그램

---



# 자료구조

---

- 기본 자료형을 근간으로 여러가지 구조를 만들 수 있는데 이러한 구조를 통칭하여 자료구조라한다
  - 자료의 접근과 처리가 용이하도록 잘 조직화된 자료의 집단
-

# 알고리즘

---

- 복잡한 문제를 쉽고 효율적으로 해결하기 위해 정의된 방법과 절차
  - 주어진 문제를 해결하기 위한 동작과 순서를 정의하는 것
  - 중요 알고리즘
    - 정렬 알고리즘
      - 선택(selection) 정렬 알고리즘
      - 버블(bubble) 정렬 알고리즘
    - 검색 알고리즘
      - 순차(sequential) 검색 알고리즘
      - 이분(binary) 검색 알고리즘
-

# 배열과 컬렉션의 개념 차이

---

## □ 배열(array)

- 고정 크기 이상의 객체를 관리할 수 없다
- 배열의 중간에 객체가 삭제되면 응용 프로그램에서 자리를 옮겨야 한다

## □ 컬렉션(collection)

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

# 컬렉션(collection)

---

- ❑ 고정 크기의 배열이 가지는 단점을 극복하고
  - ❑ 가변 개수의 객체들을 쉽게 삽입,삭제,검색을 할 수 있는 가변 크기의 컨테이너이다
  - ❑ 따라서 컬렉션은 요소(element)라고 불리는 가변 개수의 객체들의 모임이다
  - ❑ 컬렉션은 요소들을 관리하는 자료구조로서 요소의 추가,삭제,검색 등의 기능을 제공한다
-

# 컬렉션을 구현한 클래스들

---

- java.util 패키지는 컬렉션의 개념을 구현한 핵심적인 다양한 클래스를 제공한다
  - **Vector**
  - **ArrayList**
  - Hashtable
  - HashMap 등

[참고]

컬렉션을 구현한 모든 클래스들은 Object를 상속받는 객체들만 요소로 받아들인다. 즉 byte, char, short, int, long, float, double, boolean 등 8종류의 기본타입은 원칙적으로 사용할 수 없다

---

# Enumeration과 Iterator 인터페이스

---

- 어떤 객체들의 모임이 있을 이 객체들을 어떤 순서에 의해서 하나씩 접근하여 사용하기 위한 인터페이스



# Collections 클래스 활용

---

- 이 클래스는 컬렉션에 대해 연산을 수행하고 **결과로 컬렉션을 리턴하는** 유용한 유틸리티 메소드를 제공하며 모든 멤버 메소드는 static 메소드이다
  - 따라서 Collections 클래스를 사용하기 위해 객체를 생성할 필요는 없다
  - 주요 메소드  
shuffle()  
sort()
-

# 자바 클래스 라이브러리

---

- 패키지는 연관된 클래스와 인터페이스의 그룹
  - 자바 API는 그 자체가 패키지의 그룹으로 구성
-

# java.lang 패키지

---

- ❑ 자바 language 패키지는 자바 언어의 핵심을 구성한다
  - ❑ 일반적으로 자바 애플리케이션을 수행하기 위하여 기본적으로 필요한 클래스들이 java.lang 패키지에 선언되어 있다
  - ❑ java.lang 패키지는 import 하지 않아도 항상 자바 컴파일러에 의하여 바이트 코드가 만들어질 때 자동적으로 포함된다
-

# String 클래스

---

- C나 C++와는 다르게 자바의 String은 문자의 배열이 아니라 클래스이다
  - C나 C++에서의 스트링은 null로 끝나는 문자의 배열인데 반하여 자바의 스트링은 String 클래스의 객체이다
  - 모든 스트링은 `java.lang.String` 클래스로부터 상속받으며 이 클래스에 있는 메소드를 적절하게 사용할 수 있게 된다
-

# 스트링 객체 생성 방법 두 가지

---

(1) 스트링 리터널로 객체 생성

(예) `String str1 = "홍길동";`

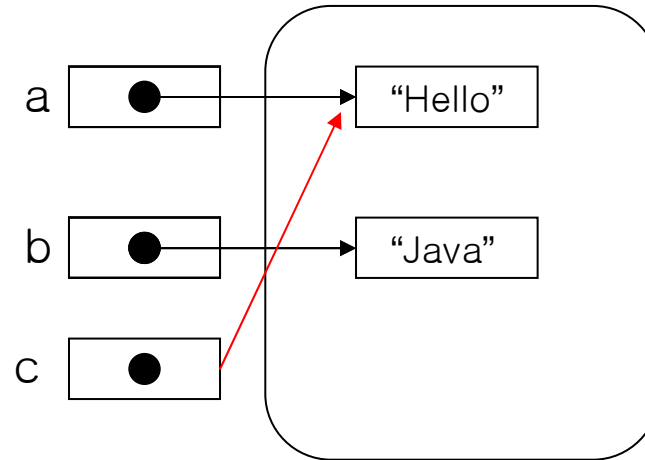
(2) String 클래스의 생성자를 이용하여 객체 생성

(예) `String str2 = new String("박지성");`

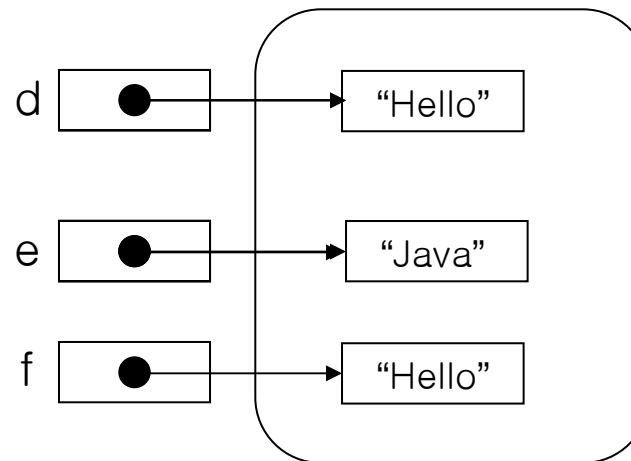
---

## <차이점>

```
String a = "Hello";  
String b = "Java";  
String c = "Hello";
```



```
String d = new String("Hello");  
String e = new String("Java");  
String f = new String("Hello");
```



힙(Heap) 메모리

# StringTokenizer 클래스

---

- java.util 패키지에 포함되어 있다
  - 문자열을 분리하기 위해 사용된다
  - 문자열을 사용할 때 사용되는 문자들을 **구분문자(delimiter)**라고 하고, 구분문자로 분리된 문자열을 **토큰(token)**이라고 한다
-

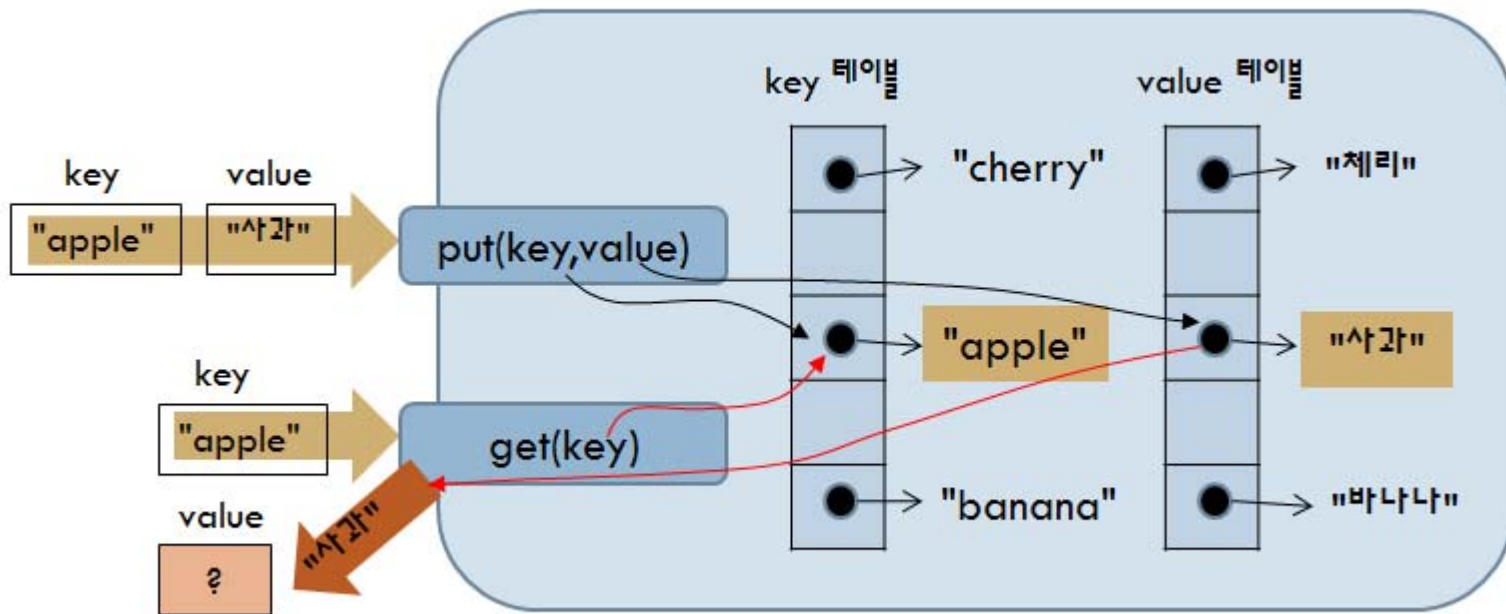
# Hashtable 클래스

---

- ❑ java.util 패키지에 포함되어 있다
  - ❑ Hashtable이 다루는 요소 객체는 항상 **키(key)와 값(value)**의 쌍으로 구성되며 키를 특정 값에 매핑시키는 해시 테이블 기능을 제공한다
  - ❑ 키와 값은 모두 객체만 사용 가능하며 int, char 등과 같은 기본 데이터 타입은 불가능하다
  - ❑ Hashtable은 내부에 키와 값을 저장하는 **자료구조를 각각 가지고 있으며** 키를 입력으로 받는 해시 함수를 통해 내부 자료구조에서의 키와 값의 위치를 결정한다
-



# Hashtable의 내부구성과 put(), get() 메소드



# 클래스

---

- 자바의 가장 핵심적인 내용
  - 자바 프로그램을 작성하는 기본단위
-

# 객체지향

---

- 데이터와 이를 처리하는 기능이 하나로 이루어져 있는 객체를 모델링하고 이들간의 관계를 정의하는 것
  - 즉 객체 자신 안에 데이터와 데이터 처리 기능의 모든 것이 포함되어 있는 것
-

# 객체지향의 중요 개념

---

## □ ▶ 캡슐화(encapsulation)

클래스의 멤버변수나 메소드들을 외부에서 사용가능한것과 그렇지 않은 것을 구분할 수 있게 하는 것으로, 정보은폐를 구현하는 개념이다. 이러한 개념을 구현하기 위해, 자바에서는 private, protected, public 등의 세가지로 구분하여 가시성을 나누고 있다.

## □ ▶ 상속성(inheritance)

코드를 **재사용(reuse)**하여 개발의 효율성을 증대하기 위한 중요한 개념이다. 자바에서 하위 클래스를 정의 할 때, **extends**라는 키워드를 함께 상위 클래스 이름을 써주면 간단하게 상속이 이루어진다.

## □ ▶ 다형성(polymorphism)

같은 메소드 이름을 중복해서 사용할 수 있게 함으로써, 프로그램 개발자에게 편리함을 주는 것이다.

자바에서는 메소드 **오버로딩과 메소드 오버라이딩**을 통해 다형성을 구현하고 있다.

---

# 접근 지정자

---

- 객체 지향 언어에는 기본적으로 접근 지정자의 개념이 있다. 객체를 캡슐화하기 때문에 한 객체를 다른 객체가 접근하는 것을 허용할 것인지 말지를 지정할 필요가 있다.

[자바에서는 4가지 접근 지정 방식 정의]

- **private** (비공개)
  - **protected** (보호된 공개)
  - **public** (공개)
  - **default** (접근 지정자 생략)
-

# 클래스 접근 지정자

---

- public – 어떤 다른 클래스에서도 사용 가능
  - default – 같은 패키지 내에 있는 클래스만이 접근 허용
-

# 멤버 접근 지정자

---

- 클래스의 멤버인 필드와 메소드의 접근 지정자

| 멤버에 접근하는 클래스 | 멤버의 접근 지정자 |           |        |         |
|--------------|------------|-----------|--------|---------|
|              | private    | protected | public | default |
| 같은 패키지의 클래스  | X          | O         | O      | O       |
| 다른 패키지의 클래스  | X          | X         | O      | X       |

---

# 자바에서의 객체 설계 방법(3단계)

---

객체 모델링

프로그래밍 하고자 하는 객체의 속성과 필요한 기능을 정리하는것



클래스 정의

객체를 실제로 사용하기 위해서 클래스라는 형태로 객체를 표현한다



클래스 생성과 사용

정의된 클래스를 이용해서, 메모리상에 객체(Object)를 생성하고 사용한다

---



# 1단계 : 객체 모델링

---

## □ 자동차의 속성과 기능

| 속성     | 기능      |
|--------|---------|
| 현재속도   | 속도를 올린다 |
| 바퀴의 수  | 속도를 내린다 |
| 자동차 이름 | 멈춘다     |

---

## 2단계 : 클래스 정의 => 멤버변수 정의

---

### 1) 클래스 이름 결정

```
class Car  
{  
}
```

### 2) 멤버변수의 정의

```
class Car {  
    int speed;           //현재속도  
    int wheelNum;        //바퀴의수  
    String carName;      //자동차이름  
}
```

---

## 2단계 : 클래스 정의 => 메소드 정의

---

```
class Car {  
    int speed;  
    int wheelNum;  
    String carName;  
  
    // 속도를 올린다  
    public void speedUp( ){  
        speed = speed + 1;  
    }  
    // 속도를 내린다  
    public void speedDown( ){  
        speed = speed - 1;  
        if(speed < 0)  
            speed = 0;  
    }  
    // 멈춘다  
    public void stop( ){  
        speed = 0;  
    }  
}
```

---

## 3단계 : 클래스 생성과 사용

---

- 클래스는 new 연산자를 이용해서 생성한다

<예> `Car myCar = new Car("제너시스");`  
`Car yourCar = new Car("소나타");`

- new 연산자의 결과로서 클래스 "객체"를 가리키는 레퍼런스 변수를 반환하게 된다

- 이 레퍼런스 변수로 해당 클래스를 마음대로 사용할 수 있다

<사용 예>

레퍼런스변수.멤버변수 => `myCar.carName`

레퍼런스변수.메소드( ) => `myCar.speedUp( )`

---

# 메소드 오버로딩(Method Overloading)

---

## (1) 정의

“같은 클래스내에서 같은 이름을 가진 메소드를 여러 개 정의 (즉, 구현)할 수 있는 것”

즉, 인자가 다를지라도 같은 동작을 해야 하는 메소드를 같은 이름을 가질 수 있도록 지원해 주는 것을 의미한다.

## (2) 오버로딩의 필요성

“생성자의 이름은 클래스의 이름과 반드시 같아야 한다”라는 규칙이 있었다.

이 규칙대로라면 생성자는 하나밖에 만들 수 없다.

이런 문제를 해결하기 위해서, 자바에는 메소드 오버로딩이라는 방식이 제공된다.

메소드 오버로딩이 가능하면 자동차의 생성자가 여러 개 필요한 경우에는 생성자를 여러 개 정의하고, 그때 그때 필요한 생성자를 호출하면 된다.

또한 생성자 오버로딩을 이용하면, 해당 객체에 가장 적합한 초기화를 할 수 있다.

---

# 메소드 오버로딩(Method Overloading)

---

## (3) 메소드 오버로딩 사용시 주의할 점

메소드의 구분은 인자의 개수와 데이터형(인자의 개수가 같은 경우)에 의해서만 구분된다는 점이다

(예) String 클래스의 `valueOf()` 메소드 경우 - 9개 존재

`valueOf`(boolean b)

`valueOf`(char c)

`valueOf`(char[] data)

`valueOf`(char[] data, int offset, int count)

`valueOf`(double d)

`valueOf`(float f)

`valueOf`(int i)

`valueOf`(long l)

`valueOf`(Object obj)

---

# 메소드 오버라이딩(Method Overriding)

---

- 상속과 관련하여 상위 클래스의 메소드를 하위 클래스에서 재정의 하는 것

# 디폴트 생성자와 상속관계

---

## □ 디폴트 생성자란 ?

클래스 안에 아무런 생성자가 없는 경우에, 컴파일러가 자동으로 생성해주는 생성자를 의미한다.

이것 때문에, 클래스 정의시 특별한 생성자를 코딩하지 않아도 괜찮았던 것이다  
하지만, 클래스 상속을 사용하는 경우에는 생성자 문제를 조금 고려해야 한다

특별히 문제가 되는 경우는 상위 클래스에서 인자가 있는 생성자만이 정의되어  
을 때 문제가 발생하게 된다.

부모 클래스에서 인자가 있는 생성자만을 정의하고 있으면, 컴파일러는 그  
클래스에게 디폴트 생성자를 제공하지 않는다  
결과적으로 인자가 없는 생성자는 자동으로 만들어지지 않는다

---



# 인터페이스(interface) => **spring framework** 적용시 중요

---

- 자바에서는 다중 상속을 가능하게 하기 위해서 인터페이스를 제공한다

<형식>

```
interface 인터페이스이름{  
    상수;  
    추상메소드이름(인자들);  
}
```

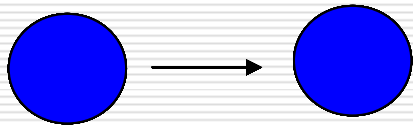
```
interface 인터페이스이름{  
    public static final 자료형 상수이름 = 값;  
    public abstract 추상메소드이름(매개변수 리스트);  
}
```

- interface는 예약어로 선언하며 상수 또는 추상 메소드들만 포함할 수 있다
  - interface 내의 메소드는 서브 클래스에서 어떻게 동작해야 할지를 구현(새롭게 정의)해 주어야 한다
  - interface는 추상 메소드를 멤버로 갖기 때문에 인스턴스를 생성할 수는 없지만 레퍼런스 변수는 선언이 가능하다
-

# 자바에서 애플리케이션 설계 방식

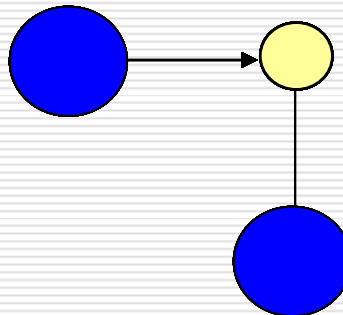
---

sample1

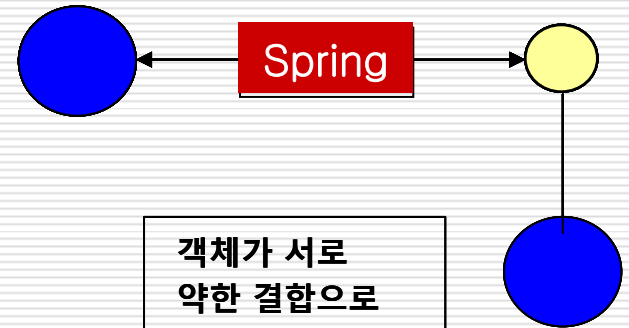


객체가 서로  
강한 결합으로  
묶여 있다

sample2



sample3



객체가 서로  
약한 결합으로  
연결된다

# 예외처리(Exception)

---

□ 프로그래밍에서 발생하는 에러

## (1) 컴파일 에러

: 문법적 에러

## (2) 예외(Exception)

: 프로그램 실행 중에 발생하는 런타임 오류는 미리 걸러낼 수가 없어 자바에서는 예외라는 것을 사용하여 처리한다

---

# 예외가 발생하는 경우

---

- 정수를 0으로 나눌 때 발생
  - 배열의 범위를 벗어난 접근 시 발생
    - ▶ **ArrayIndexOutOfBoundsException**
  - 유효하지 않은 형변환
  - 파일을 열어서 읽거나 쓸 때 파일이 존재하지 않은 경우
  - 잘못된 인자 전달 시 발생
    - ▶ **IllegalArgumentException**
  - 레퍼런스 변수가 null인 상태에서 객체를 참조할 경우
    - ▶ **NullPointerException**
  - 문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환시 발생
    - ▶ **NumberFormatException**
-

# 예외처리, try-catch-finally문

---

```
try {  
    예외가 발생할 가능성이 있는 실행문(try 블록)  
}  
catch (처리할 예외 타입 선언) {  
    예외 처리문(catch 블록)  
}  
finally { //finally는 생략 가능  
    예외 발생 여부와 상관없이 무조건 실행되는 문장(finally 블록)  
}
```

# 이벤트 처리

---

## □ 정의

- 자바는 이벤트 주도형 프로그래밍 방식이기 때문에, 이벤트 처리에 대해서 필수적으로 이해하고 있어야 한다.  
GUI의 컴포넌트가 이벤트를 발생시키고, 이 이벤트에 의해 기능이 수행되는 프로그래밍 방식을 이벤트 주도형 프로그래밍이라 한다.
  - GUI 프로그램에서 GUI 컴포넌트와 실제의 기능(즉 이벤트 핸들러)을 연결하는 부분을 이벤트(Event)라는 구조로 지원된다.
  - AWT 컴포넌트가 GUI의 겉모습을 이룬다면, 이벤트는 컴포넌트와 실제 기능을 연결하는 GUI 프로그램의 신경계라고 할 수 있다.
-

# 자바 에서의 Event 처리 구조(3가지 구성요소)

---

## (1) 이벤트 소스(Event Source)

- 이벤트가 발생하는 컴포넌트 즉, AWT나 스윙에서 볼 수 있는 버튼, 스크롤바 등 사용자가 마우스나 키보드로 입력을 가할 수 있는 모든 컴포넌트가 이벤트 소스가 된다.

## (2) 이벤트 클래스

- 자바가상머신에서 정의되어 있기 때문에, 개발자는 이벤트 소스와 이벤트 핸들러의 두 가지 요소만을 프로그래밍할 수 있다.

## (3) 이벤트 핸들러(Event Handler)

- 이벤트를 처리하는 기능
-

# 이벤트 처리 각 단계별

---

## 1단계 : 이벤트 클래스의 선정

- 이벤트 소스와 처리하려는 이벤트 선정

## 2단계 : 이벤트 핸들러의 구성

- 이벤트를 처리하기 위한 클래스 생성
- 이벤트 핸들러를 작성하기 위해서는 각 이벤트를 처리하기 위해, 준비된 메소드를 선언한 인터페이스를 이용해야 한다. 이러한 인터페이스를 리스너(Listener) 인터페이스라고 한다.
- 이 인터페이스의 이름은 대부분 처리하려는 이벤트의 이름과 Listener라는 이름을 조합하여 만들어진다.  
(예, ActionListener, FocusListener, ItemListener, MouseListener 등)
- 각 이벤트리스너에 해당하는 메소드 구현

## 3단계 : 이벤트 핸들러와 이벤트의 연결

- 이벤트 핸들러가 만들어지면, 이벤트 소스에 해당하는 컴포넌트와 연결되어야 한다.
  - 이 연결 작업을 해주는 메소드가 바로 **addXxxListener()** 이다.  
(예, addActionListener(), addFocusListener(), addItemListener(), addMouseListener() 등)
-



# 내부 클래스(Inner Class)

---

- 내부 클래스를 사용하는 경우  
이벤트 핸들러를 만드는데 많이 사용되는 방식 - **안드로이드 APP 개발시**  
(즉 이벤트 핸들링을 효율적으로 처리하기 위해서)
- <내부 클래스 작성 형식>

```
class 외부_클래스
{
    .....
    class 내부_클래스
    {
        .....
    }
}
```

# 내부 클래스의 주요 특징

---

1. 내부 클래스도 사용시 반드시 클래스 생성
2. 외부 클래스의 멤버변수와 메소드를 마음대로 사용 가능
3. 내부 클래스는 외부에서 단독으로 사용 불가
4. 따로 컴파일하지 않아도 자동으로 생성
5. 클래스 파일이 외부\_클래스\_이름\$내부\_클래스\_이름.class로 생성됨

## 내부 무명 클래스(inner anonymous class)

---

- 무명 클래스는 어떤 class나 interface를 상속/구현 해야만 그 instance를 사용할 수 있는 것이다
- 이처럼 무명 클래스를 사용하면 어떤 절차(수행)를 다른 method의 인수로 건네줄 수 있게 된다. 하지만 간단한 로직만 구현 처리해야 한다.
- 무명 클래스는 조금만 복잡해져도 급격히 소스의 '가독성'이 떨어지게 되므로 남용하지 않는 것이 바람직하다"

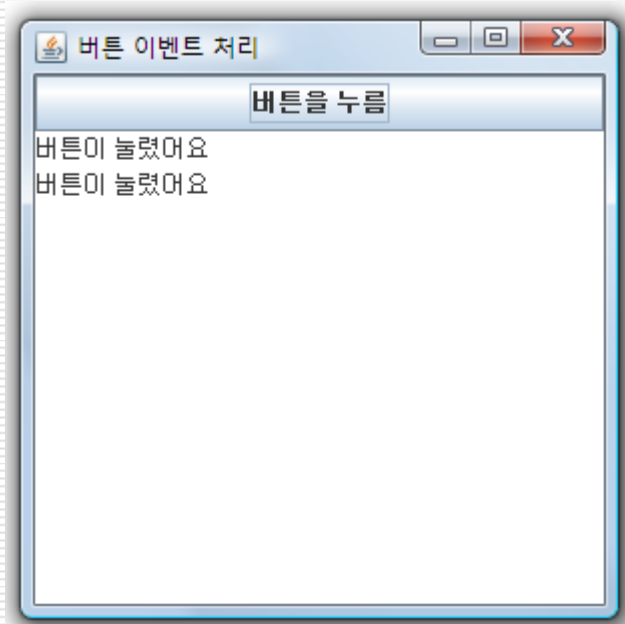
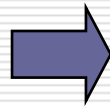
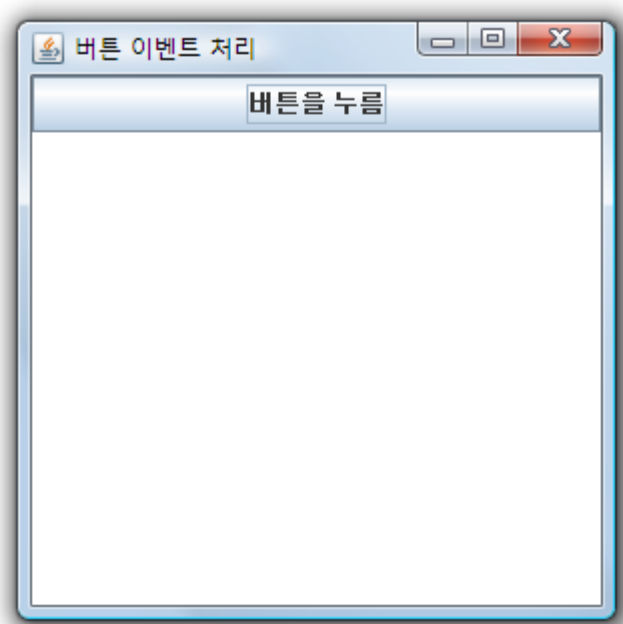
# Adapter의 사용

---

- 한 개의 메소드만을 가진 리스너 인터페이스도 있으나, WindowListener 인터페이스의 경우에는 7개의 메소드를 가지고 있다
  - 이러한 경우 인터페이스를 구현하는 이벤트 클래스는 인터페이스가 갖고 있는 모든 메소드를 구현해주어야 한다.  
(예) 윈도우 닫기에 사용되는 Close Event 만을 처리하고 싶은 경우에도 windowClosed() 메소드 외에 다른 6개의 메소드를 모두 처리해주어야 한다.  
<문제 해결책으로>  
해당 인터페이스를 모두 구현해 놓은 [Adapter라는 클래스](#) 제공  
해당 Adapter 클래스를 상속받아 필요한 메소드만을 오버라이딩 해주면 된다.
-

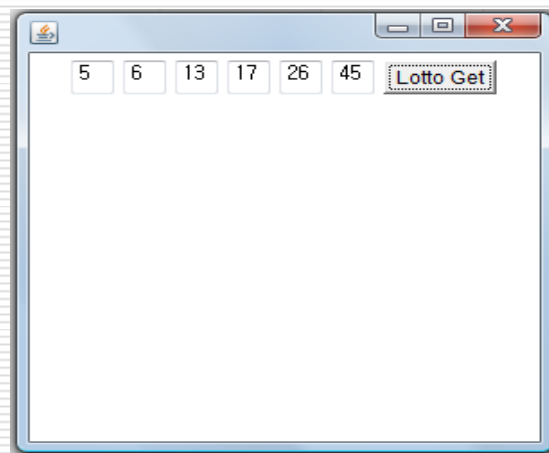
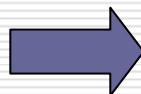
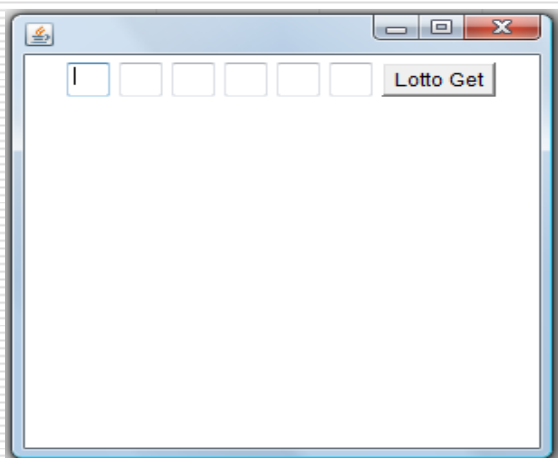
# ButtonEvent 프로젝트 - 실행결과

---



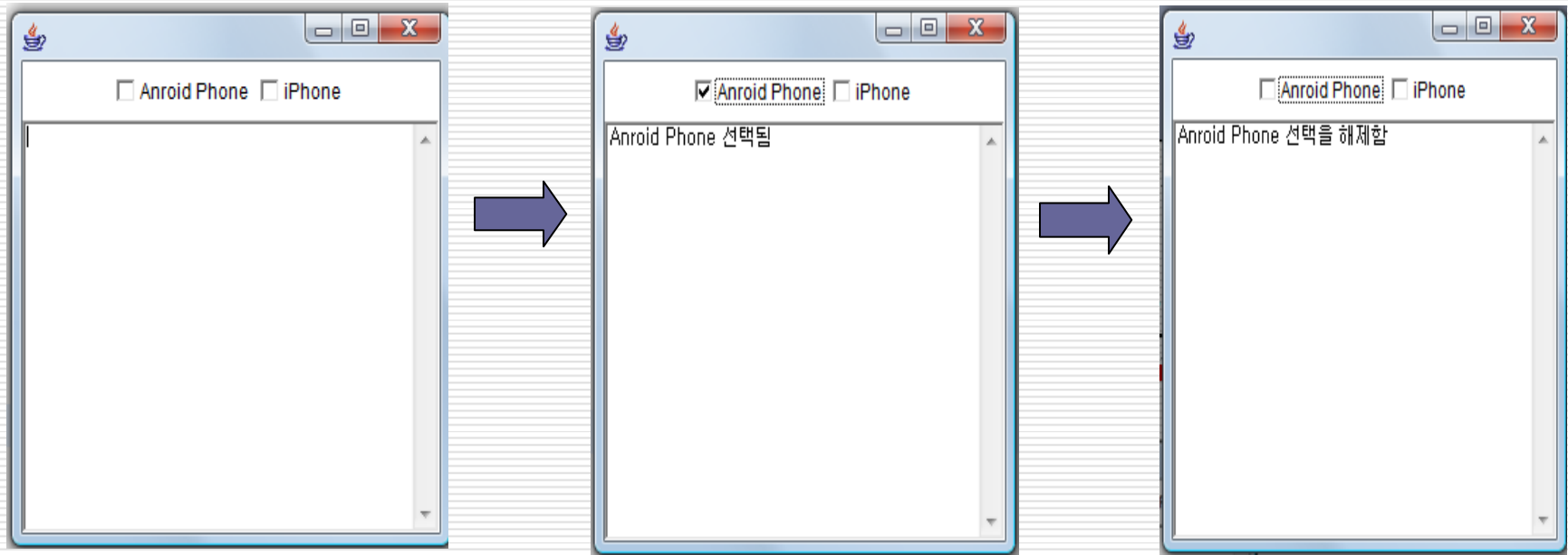
# 로또번호 발생기

---



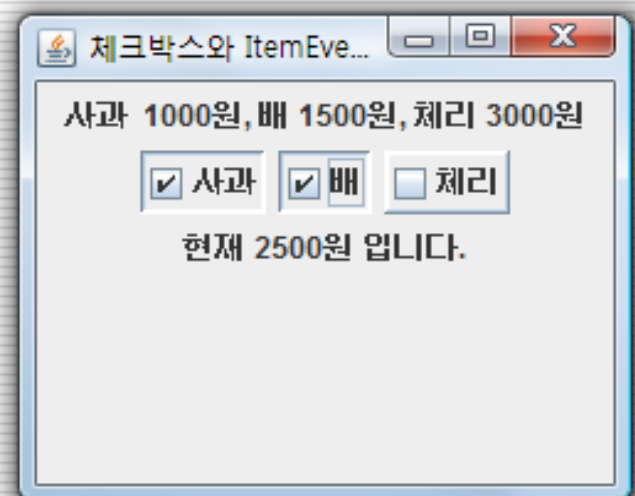
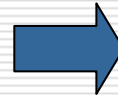
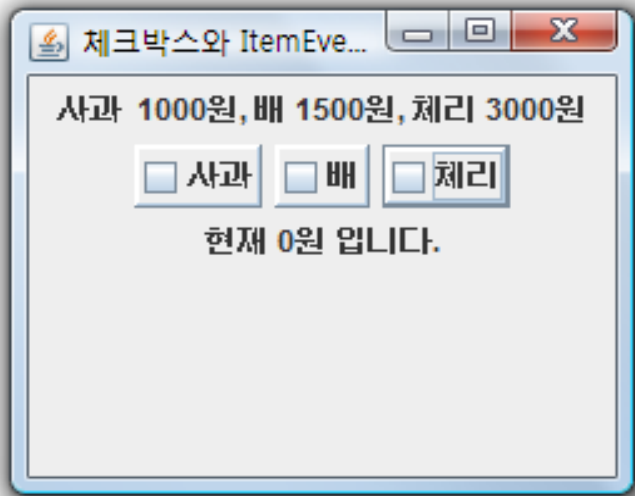
# CheckboxEventTest 프로젝트 - 실행결과

---



# JCheckBox와 ItemEvent 처리 프로젝트

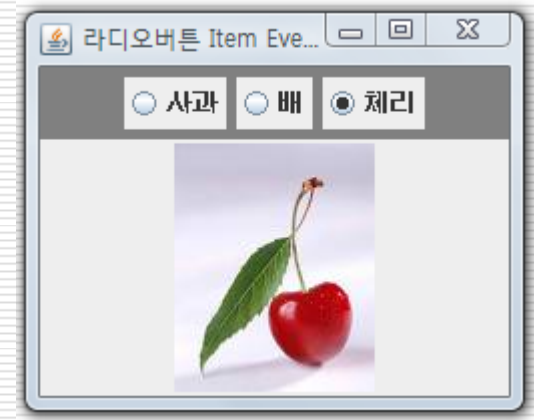
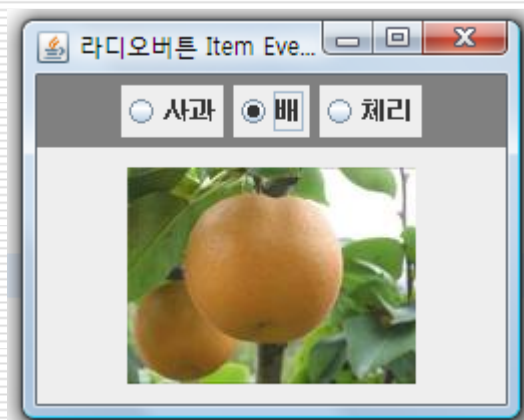
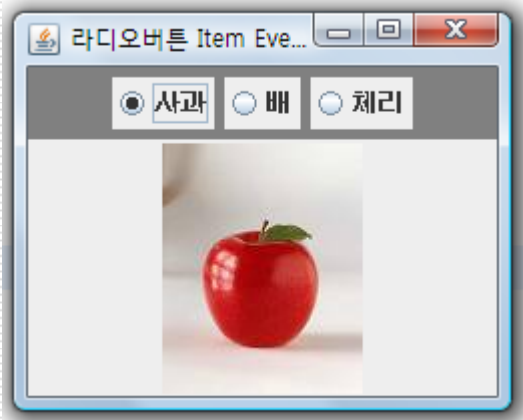
---





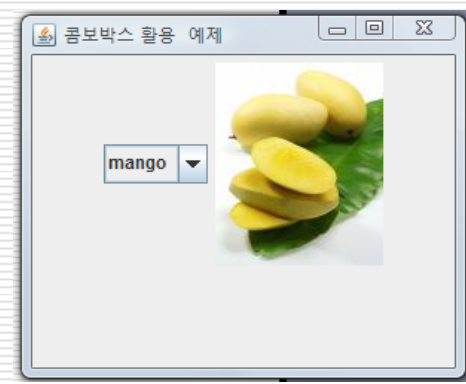
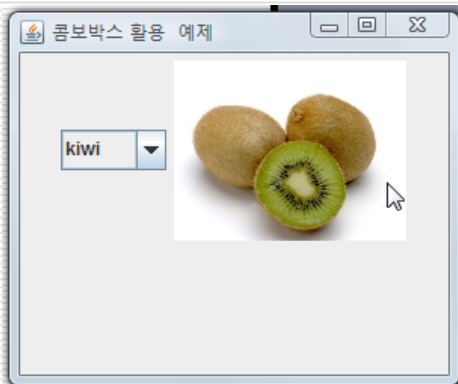
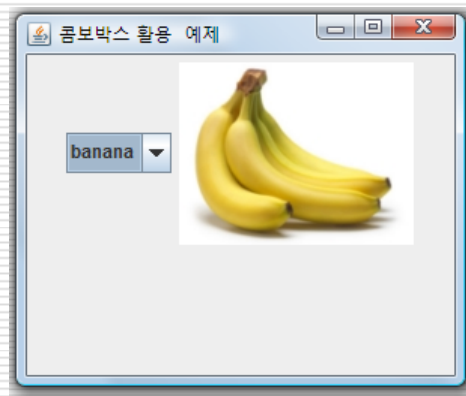
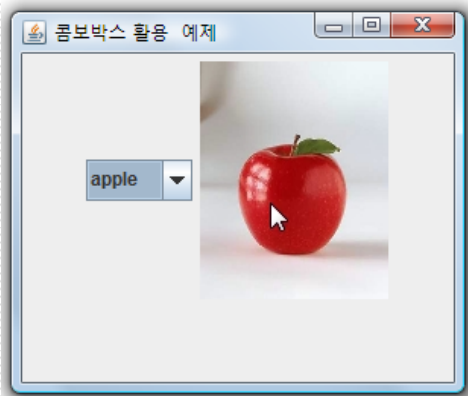
# JRadioButton과 ItemEvent 처리 프로젝트

---



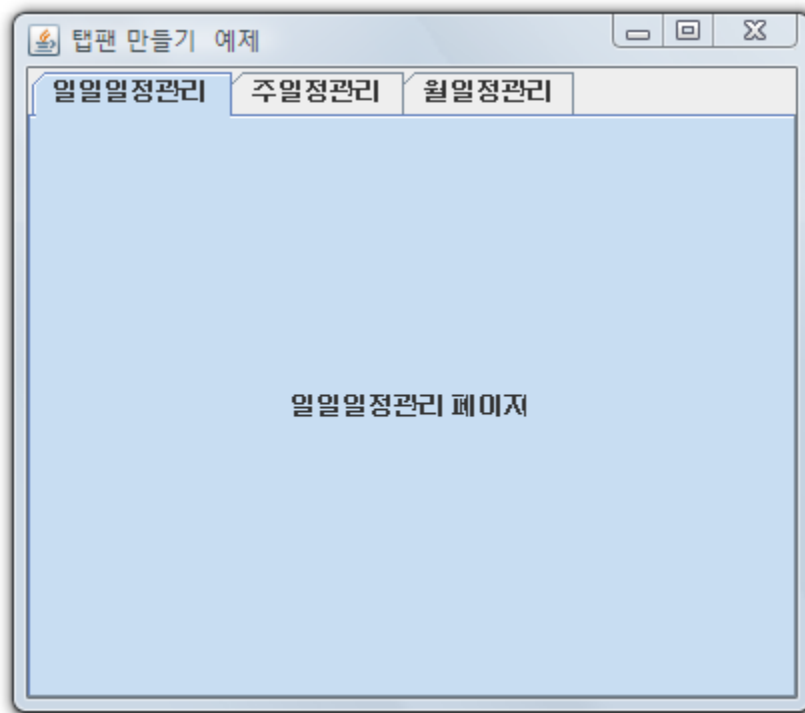
# JComboBox와 ActionEvent 처리 프로젝트

---



# 탭판 만들기

---



# 팝업 다이얼로그 만들기

---

- JOptionPane 클래스  
팝업 다이얼로그를 생성할 수 있는 static 메소드를 여러 개 지원한다.
  - 팝업 다이얼로그는 모두 다 '모달' 타입이다.  
그러므로 팝업 다이얼로그를 닫기 전에는 프레임 혹은 다른 창으로 이동 할 수 없다.
-

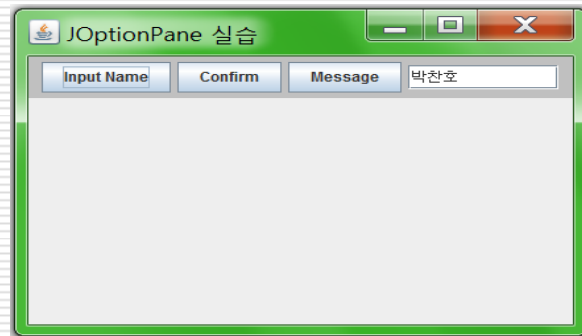
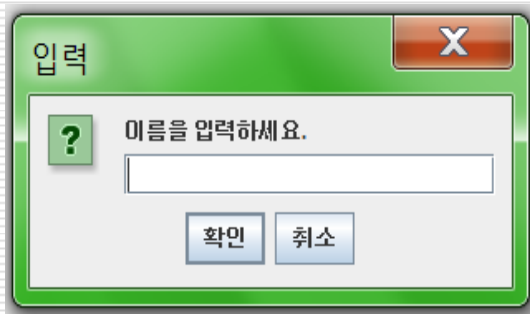
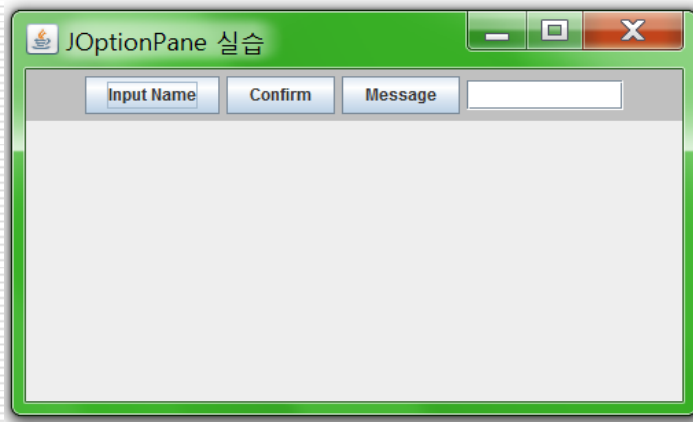
# 필요 메소드

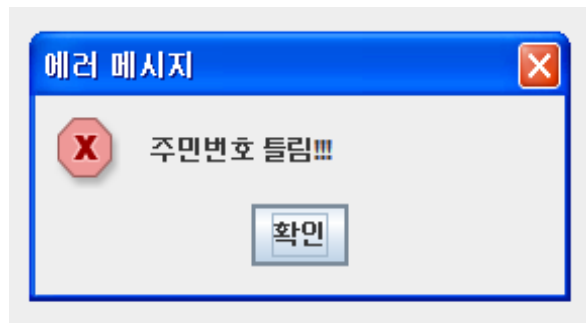
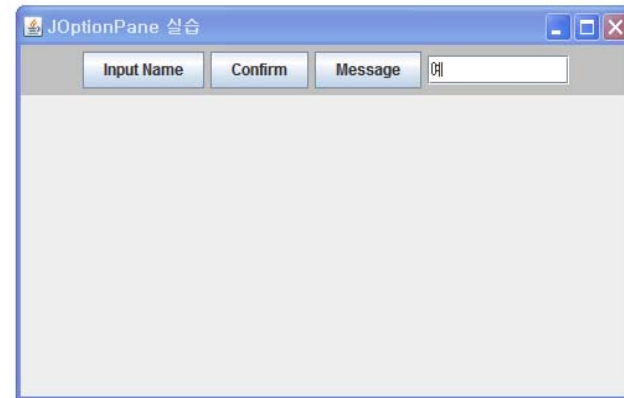
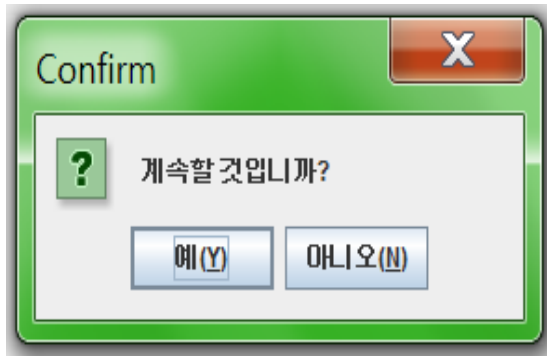
---

- (1) static String `showInputDialog`(Object message)
  - (2) static int `showConfirmDialog`(Component parentComponent,  
Object message,  
String title  
int optionType)
  - (3) static void `showMessageDialog`(Component parentComponent,  
Object message,  
String title,  
int messageType)
  - (4) static void `showMessageDialog`(Component parentComponent,  
Object message)
-

# JOptionPane 클래스 이용하여 구현

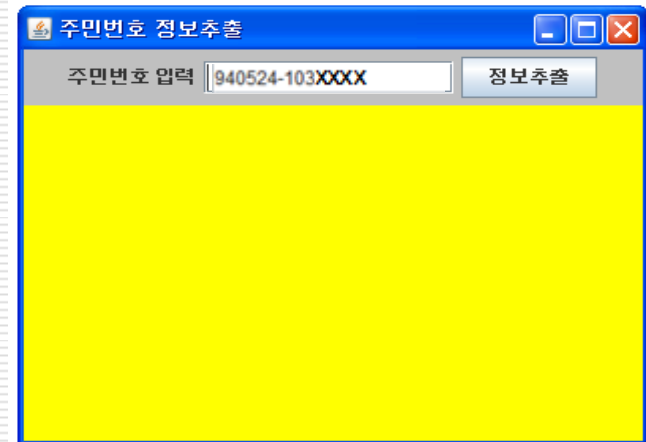
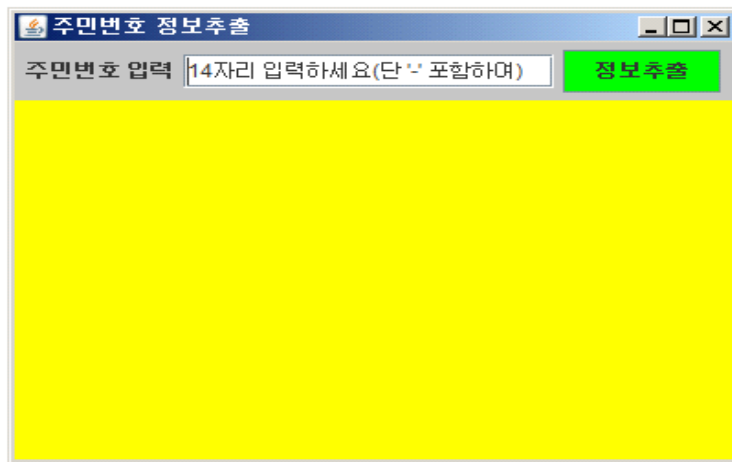
---





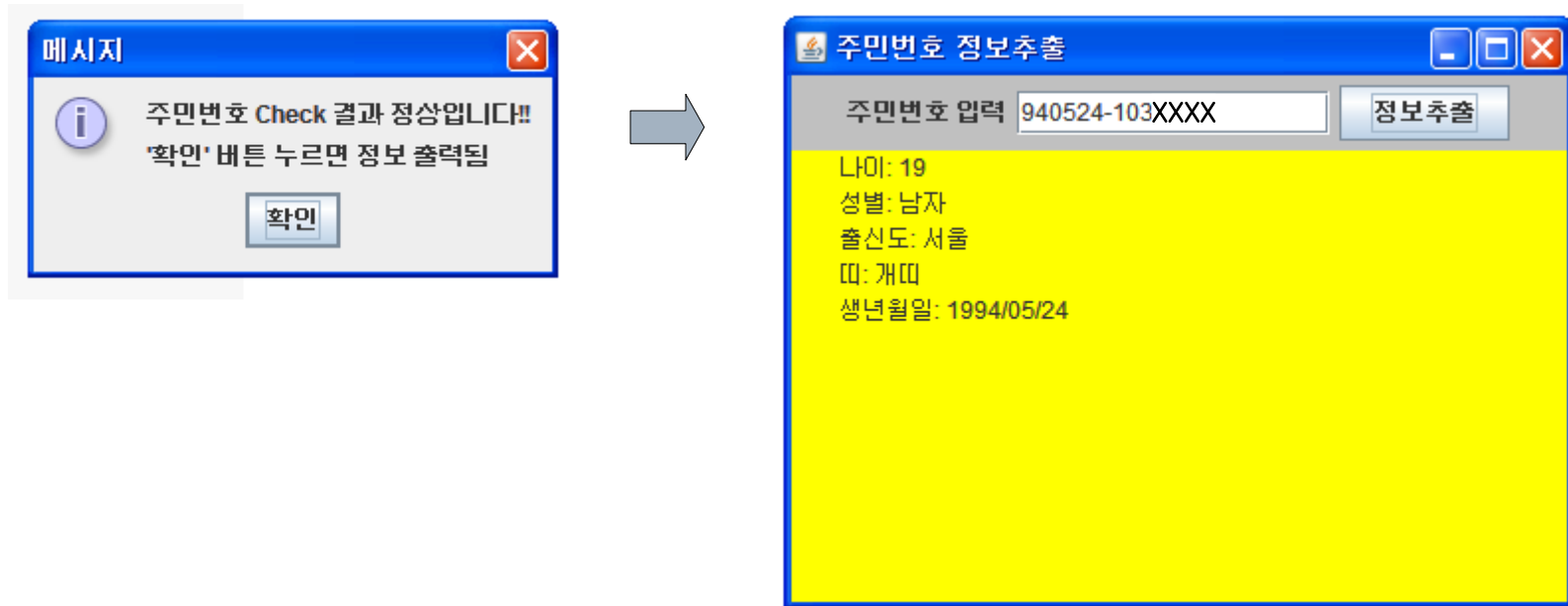
# 주민번호\_정보추출(GUI) 프로그램

---

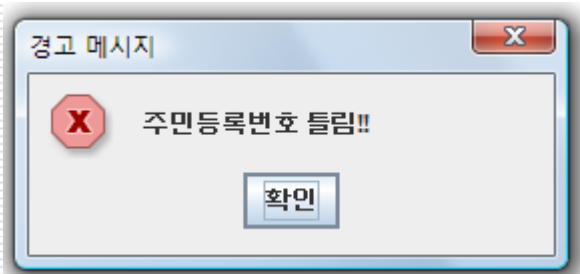
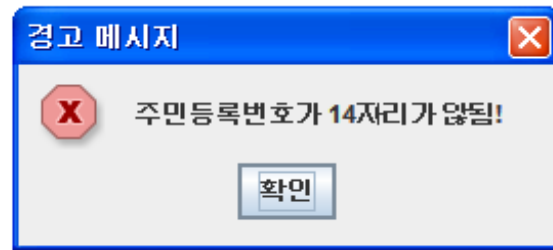
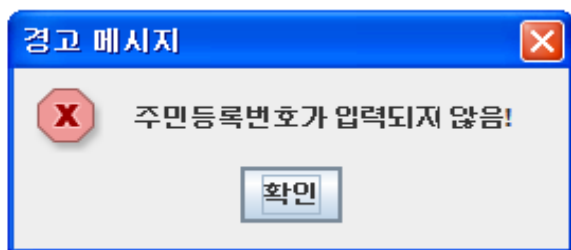




- 주민번호 데이터 입력 후 “정보추출” 버튼 클릭 후  
주민번호가 정상이면 메시지 다이얼로그 창을 띄운 후  
개인정보를 출력한다



- ▶ 주민번호를 입력하지않고 “정보추출” 버튼을 클릭한 경우
  - ▶ 주민번호가 14자리가 안되는 경우
  - ▶ 주민번호가 틀린 경우 – 경고 메시지 다이얼로그 창 띄우기
- 



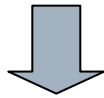
# MouseEvent 처리

마우스 이벤트 테스트

로그인

아이디 or 이메일

비밀번호



마우스 이벤트 테스트

로그인

비밀번호

마우스 이벤트 테스트

로그인

아이디 or 이메일

## ■ 고객정보 관리 시스템 실행화면

**고객정보 관리시스템**

파일

정렬

도움말

입력

번호

이름

핸드폰 번호

이메일

주민등록번호

직업

선택

신상정보

나이 :

성별 :

출신도 :

생일 :

| 번호 | 이름 | 핸드폰번호 | E-Mail | 주민등록번호 | 나이 | 성별 | 출신도 | 생일 | 직업 |
|----|----|-------|--------|--------|----|----|-----|----|----|
|    |    |       |        |        |    |    |     |    |    |

추가

삭제

이전

다음

수정

검색

## ■ '검색' 버튼 클릭시 실행화면

고객정보 관리시스템

파일 정렬 도움말

입력

번호

이름

핸드폰 번호

이메일

주민등록번호

직업

정보검색

☐ 이름 ☐ 직업 ☐ 출신도

| 번호 | 이름 | 핸드폰번호 | E-Mail | 주민등록번호 | 나이 | 성별 | 출신도 | 생일 | 직업 |
|----|----|-------|--------|--------|----|----|-----|----|----|
|----|----|-------|--------|--------|----|----|-----|----|----|

추가 삭제 이전 다음 수정 **검색**

← CardLayout

■ '파일' 메뉴 클릭시 실행화면

고객정보 관리시스템

파일 정렬 도움말

열기  
저장  
닫기

JMenuItem

| 번호 | 이름 | 핸드폰번호 | E-Mail | 주민등록번호 | 나이 | 성별 | 출신도 | 생일 | 직업 |
|----|----|-------|--------|--------|----|----|-----|----|----|
|----|----|-------|--------|--------|----|----|-----|----|----|

이름

핸드폰 번호

이메일

주민등록번호

직업

선택

신상정보

나이 :

성별 :

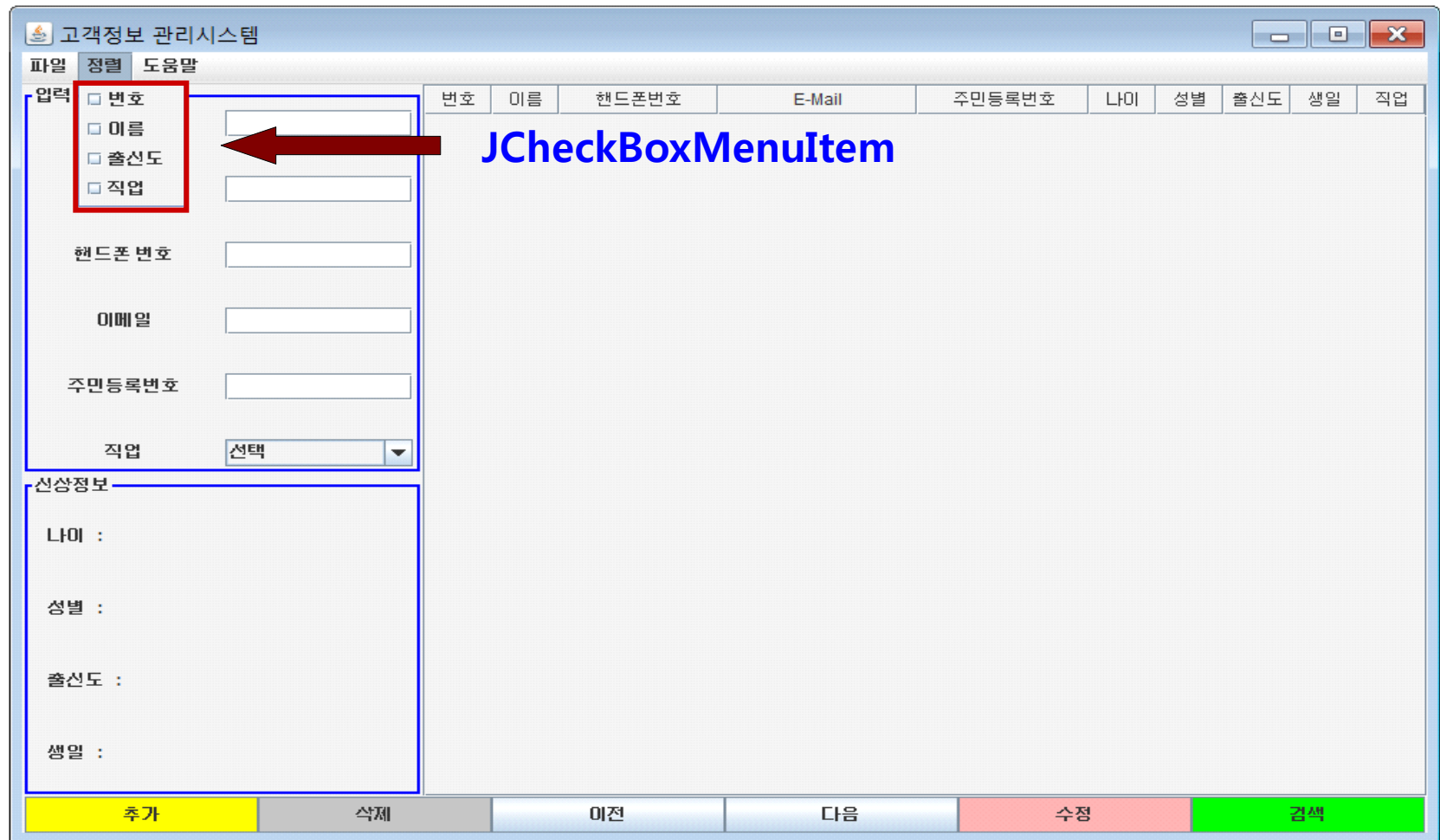
출신도 :

생일 :

JButton

추가 삭제 이전 다음 수정 검색

## ■ '정렬' 메뉴 클릭시 실행화면



고객정보 관리시스템

파일 정렬 도움말

입력

- ☐ 번호
- ☐ 이름
- ☐ 출신도
- ☐ 직업

핸드폰 번호

이메일

주민등록번호

직업

신상정보

나이 :

성별 :

출신도 :

생일 :

번호 이름 핸드폰번호 E-Mail 주민등록번호 나이 성별 출신도 생일 직업

추가 삭제 이전 다음 수정 검색

JCheckBoxMenuItem

고객 정보 관리 프로그램

파일 정렬 색상 도움말

입력

번호

성명

핸드폰 번호

이메일

주민등록번호  -

직업

정보

나이

성별

출신동

생일

| 번호 | 성명  | 핸드폰번호         | E-Mail        | 주민등록번호         | 나이 | 성별 | 출신도 | 생일     | 직업  |
|----|-----|---------------|---------------|----------------|----|----|-----|--------|-----|
| 1  | 김영은 | 010-0000-0000 | nate@nate.com | 880403-2125465 | 25 | 여성 | 인천  | 04월03일 | 학생  |
| 2  | 김원식 | 010-0000-0000 | nate@nate.com | 870403-1125465 | 26 | 남성 | 인천  | 04월03일 | 학생  |
| 3  | 조용표 | 010-0000-0000 | nate@nate.com | 851112-1678456 | 28 | 남성 | 전남  | 11월12일 | 회사원 |
| 4  | 손민철 | 010-0000-0000 | nate@nate.com | 850911-1325874 | 28 | 남성 | 강원  | 09월11일 | 자영업 |
| 5  | 김미영 | 010-0000-0000 | nate@nate.com | 920505-2210101 | 21 | 여성 | 부산  | 05월05일 | 학생  |
| 6  | 나영은 | 010-0000-0000 | nate@nate.com | 940701-2828123 | 19 | 여성 | 경남  | 07월01일 | 학생  |
| 7  | 오국환 | 010-0000-0000 | nate@nate.com | 910721-1234123 | 22 | 남성 | 부산  | 07월21일 | 공무원 |

추가 삭제 이전 다음 출력 수정 검색



## DefaultTableModel 적용한 JTable

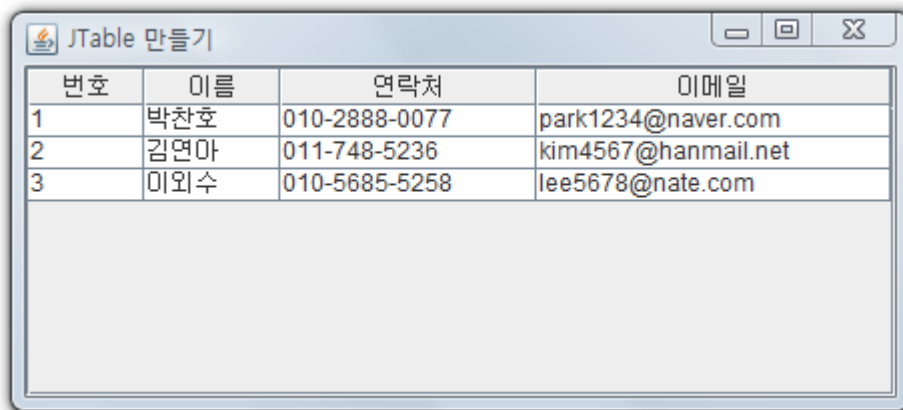
생성자

DefaultTableModel(Vector data, Vector columnNames)



# JTable 만들기

---



| 번호 | 이름  | 연락처           | 이메일                 |
|----|-----|---------------|---------------------|
| 1  | 박찬호 | 010-2888-0077 | park1234@naver.com  |
| 2  | 김연아 | 011-748-5236  | kim4567@hanmail.net |
| 3  | 이외수 | 010-5685-5258 | lee5678@nate.com    |

# JTable 클래스 중요 메소드

---

- ❑ `int getRowCount()`
  - ❑ `Object getValueAt(int row, int column)`
  - ❑ `void setValueAt(Object aValue, int row, int column)`
  - ❑ `int getSelectedRow()`
  - ❑ `void changeSelection(int rowIndex, int columnIndex,  
boolean toggle, boolean extend)`
-

# JTable을 원할히 사용하기 위해서 3개의 관련요소를 사용

---

- (1) 먼저 JTable에 사용할 모델 결정
  - (2) 위에서 만든 모델을 선정하여 JTable 생성
  - (3) JTable에 입력될 data량이 많을 경우를 대비해서 스크롤바를 붙인다
-

# DefaultTableModel 이용하여 구현

---




Table Test window showing an empty table with columns: 이름, 나이, 성별. Below the table are input fields for 이름, 나이, and 성별, and buttons for 추가 and 삭제.

| 이름 | 나이 | 성별 |
|----|----|----|
|----|----|----|

이름  나이  성별

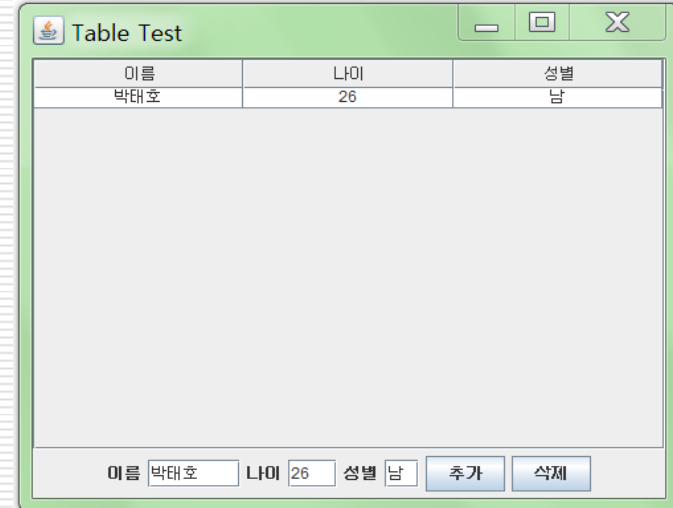
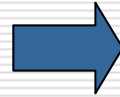


Table Test window showing a table with one row of data: 이름 박태호, 나이 26, 성별 남. Below the table are input fields for 이름, 나이, and 성별, and buttons for 추가 and 삭제.

| 이름  | 나이 | 성별 |
|-----|----|----|
| 박태호 | 26 | 남  |

이름  박태호 나이  26 성별  남

# DefaultTableModel 클래스의 중요 메소드

---

## □ setDataVector(Vector dataVector, Vector columnIdentifiers)

=> 현재의 dataVector 인스턴스 변수를 새로운 행의 Vector인 dataVector에 옮겨놓는다

## □ fireTableDataChanged()

=> 테이블의 행의 모든 셀 값이 변경되고 있을 가능성이 있는 것을 모든 청취자에게 통지한다.  
행수도 변경되고 있는 경우가 있어 JTable은 테이블의 draw를 최초 부터 다시 할 필요가 있다.  
즉, 테이블에 저장된 데이터가 변경되었고 그 내용을 화면에 반영해야할 경우에는 TabeModel  
에서 이 메소드를 호출하여 Table에 데이터가 바뀌었음을 알려주는 이벤트를 발생시켜야 한다.

# "입력" 보더 만들기 프로젝트

---

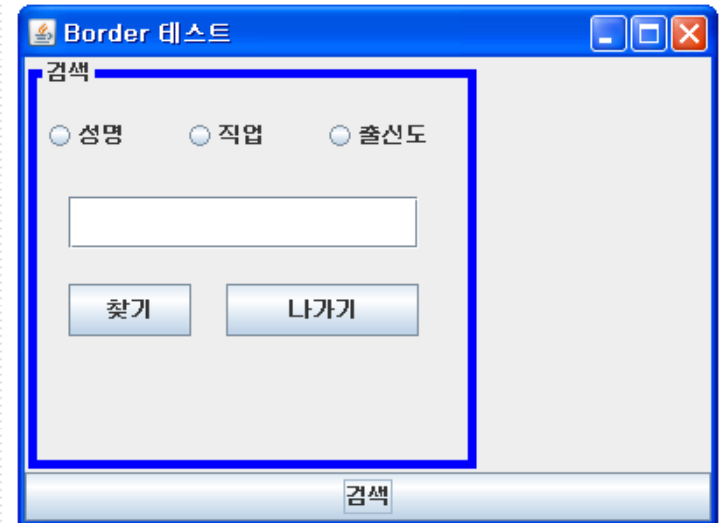
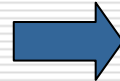
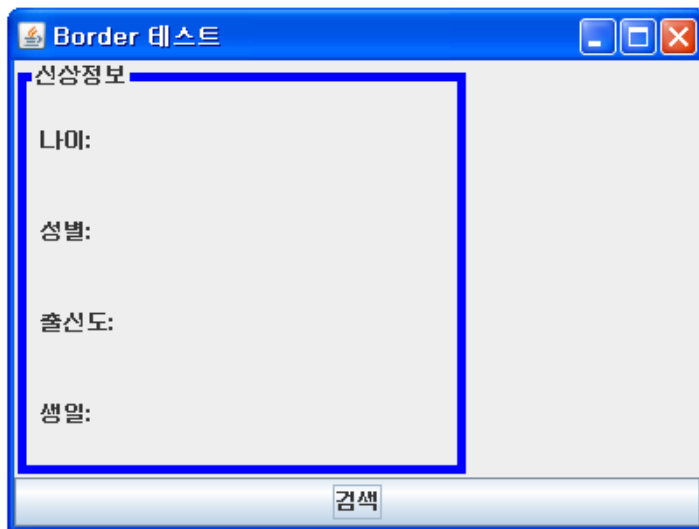
입력 보더 테스트

입력

|        |                                  |
|--------|----------------------------------|
| 번호     | <input type="text"/>             |
| 성명     | <input type="text"/>             |
| 핸드폰번호  | <input type="text"/>             |
| 이메일    | <input type="text"/>             |
| 주민등록번호 | <input type="text"/>             |
| 직업     | <input type="text" value="회사원"/> |

# CardLayout 테스트

---



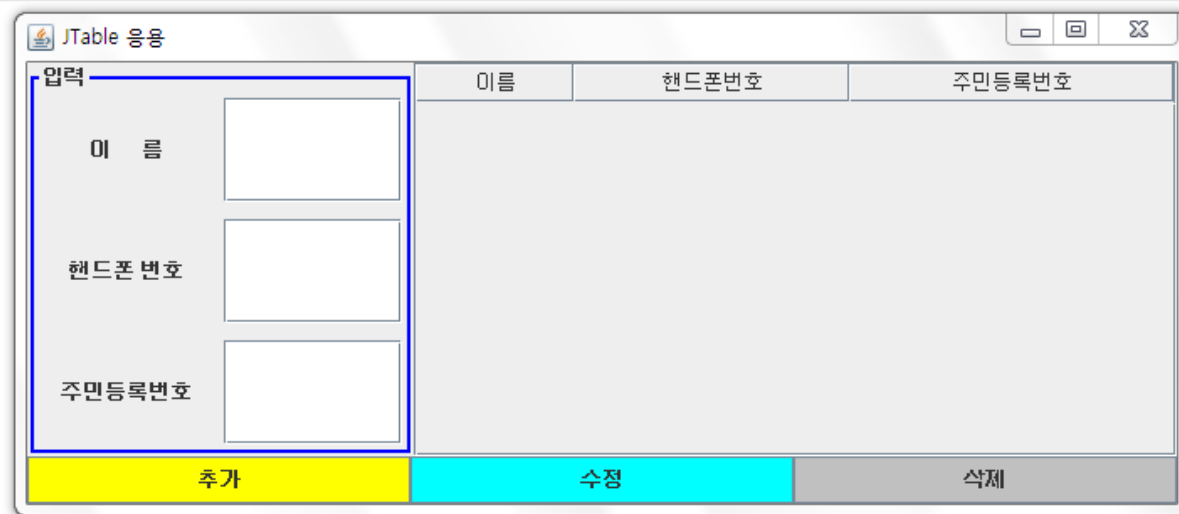
“검색”버튼 클릭하면 [신상정보]보더가 [검색]보더로 바뀐다. (즉 카드 교체)

---

# JTable 응용

## DefaultTableModel 이용하여 구현

---

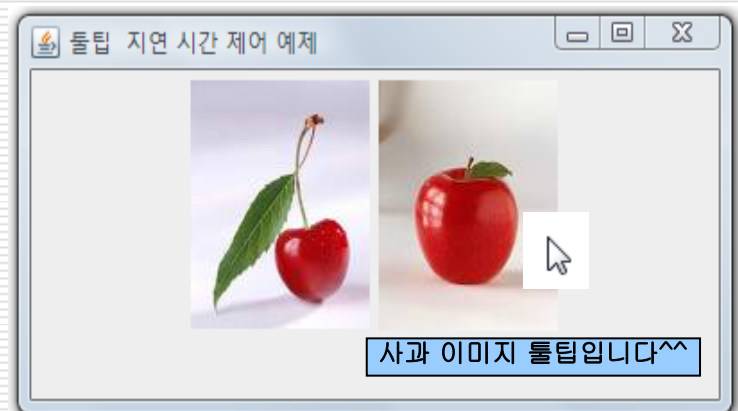
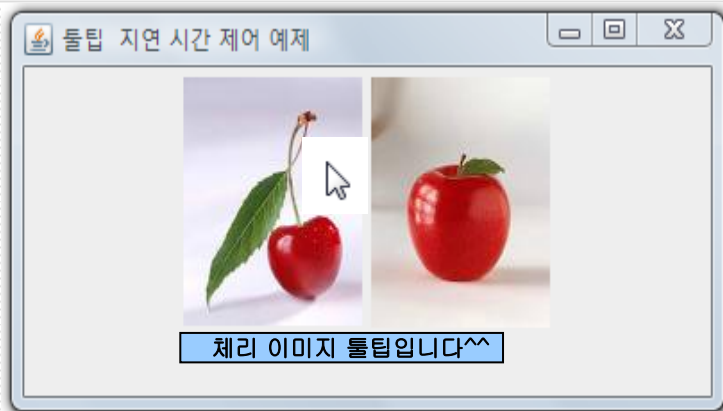
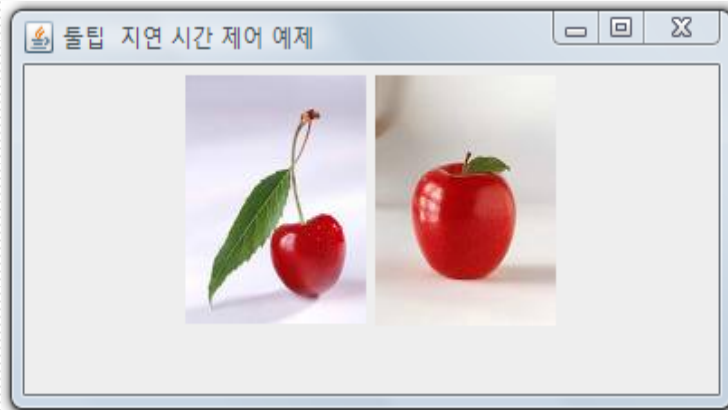


The screenshot shows a Java Swing window titled "JTable 응용". On the left side, there is an input section labeled "입력" (Input) containing three text input fields with labels "이름" (Name), "핸드폰 번호" (Phone Number), and "주민등록번호" (Residential Registration Number). To the right of the input section is a JTable with three columns: "이름", "핸드폰번호", and "주민등록번호". The table is currently empty. At the bottom of the window, there are three buttons: "추가" (Add) in yellow, "수정" (Modify) in cyan, and "삭제" (Delete) in gray.



# ToolTipManager와 툴팁 시간 제어 프로젝트

---



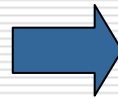
# 금액을 화폐단위로 변환하기

---

Money Changer

금액  계산

|      |                      |
|------|----------------------|
| 오만원  | <input type="text"/> |
| 만원   | <input type="text"/> |
| 천원   | <input type="text"/> |
| 500원 | <input type="text"/> |
| 100원 | <input type="text"/> |
| 50원  | <input type="text"/> |
| 10원  | <input type="text"/> |
| 1원   | <input type="text"/> |



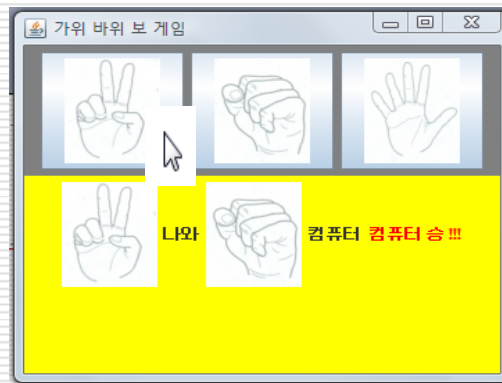
Money Changer

금액  계산

|      |                                |
|------|--------------------------------|
| 오만원  | <input type="text" value="1"/> |
| 만원   | <input type="text" value="1"/> |
| 천원   | <input type="text" value="5"/> |
| 500원 | <input type="text" value="0"/> |
| 100원 | <input type="text" value="3"/> |
| 50원  | <input type="text" value="1"/> |
| 10원  | <input type="text" value="2"/> |
| 1원   | <input type="text" value="0"/> |

# 가위, 바위, 보 게임 프로젝트

---



# JDBC

---

- JDBC(Java Database Connectivity)
  - JDBC는 자바로 작성된 프로그램을, 일반 데이터베이스에 연결하기 위한 응용프로그램 인터페이스 규격
  - 응용프로그램 인터페이스는 데이터베이스 관리시스템에 넘겨질 SQL 형태의 데이터베이스 접근요구 문장을, 각 시스템에 맞도록 바꾸어준다.
-

# JDBC의 출현 배경

---

- JDBC가 SQL을 자바로 객체모델링 한 것에 불과하기 때문에, 몇 가지 스텝만 거치면 누구라도 쉽게 사용 가능
  - JDBC가 나오기 이전에는 자바를 이용해서 DB에 접근하기 위해서는, DB 루틴을 C언어 등으로 네티브 메소드(Native Method)를 만들어 연결하는 수 밖에 없었다.
  - 그러나, 이런 과정은 자바 프로그램의 이식성에 큰 문제점으로 지적되었다. 또한, DB의 종류별로 네티브 메소드를 만드는 것도 쉽지 않은 일이었다.
-

# JDBC 드라이버 타입

---

- Type 1 : JDBC-ODBC Bridge
- Type 2 : Native-API/partly Java driver
- Type 3: Net-protocol/all-Java driver
- Type 4 : Native-protocol/all-Java driver

## [참고]

- (1) 드라이버 관리자는 데이터베이스에 맞는 드라이버를 찾고  
JDBC 초기화 작업을 수행한다  
Type 4는 JDBC 호출을 DBMS에 바로 전달한다.
  - (2) 100% 자바 클래스로 되어 있으며, **오라클의 경우 Thin 드라이버**  
가 있다
-

# JDBC 드라이버

---

| JDBC 드라이버        | Type | 드라이버클래스                         | 연결URL                                 |
|------------------|------|---------------------------------|---------------------------------------|
| 오라클<br>thin 드라이버 | 4    | oracle.jdbc.driver.OracleDriver | jdbc:oracle:thin:@<server-IP>:1521:xe |

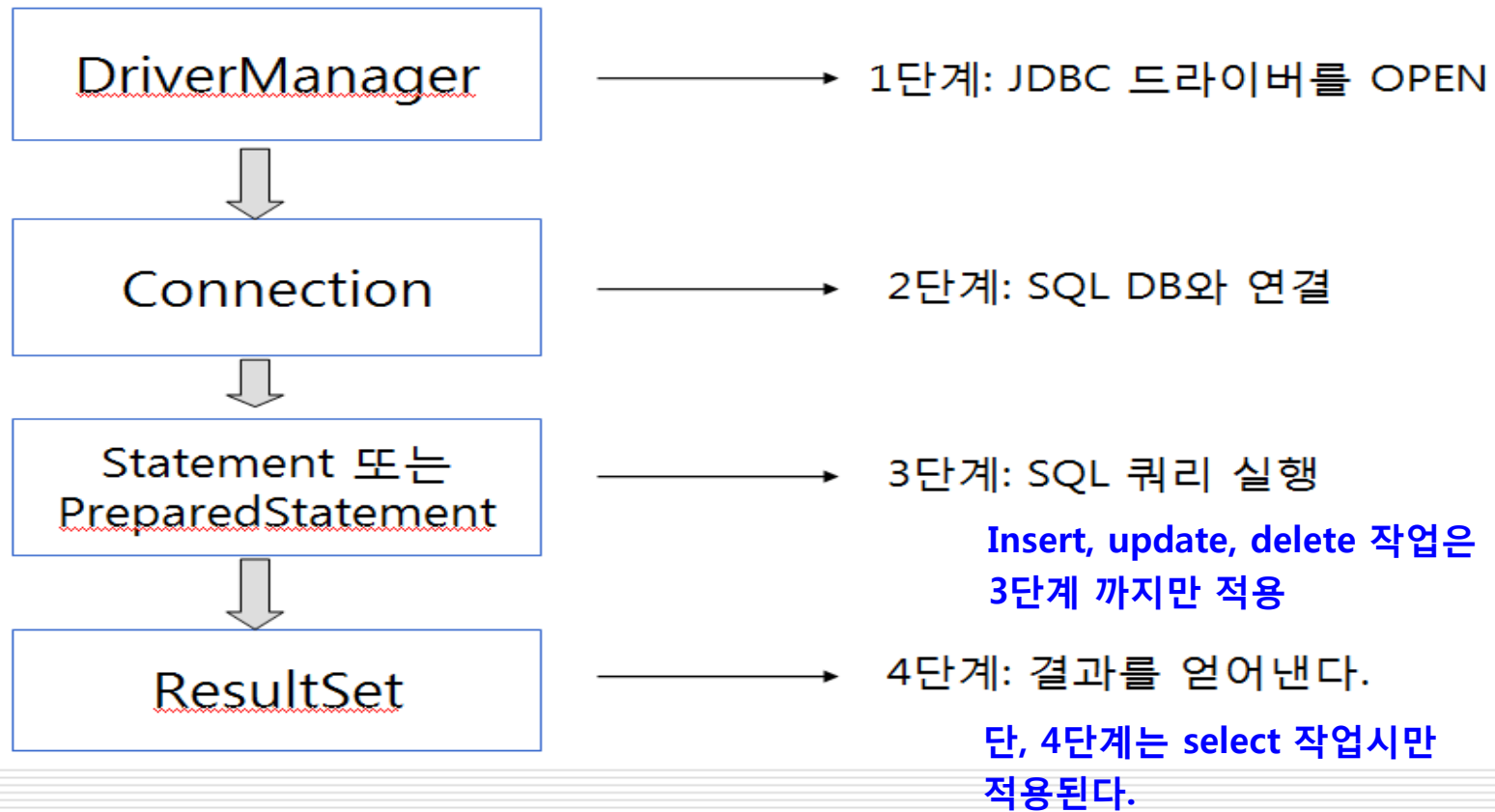
## [참고]

오라클 10g XE(Express Edition) 설치시  
전역 데이터베이스 이름은 'xe'로 설정됨

---

# JDBC의 사용법

---





# 오라클 데이터베이스의 자료형

---

| 자료형            | 설명             | 크기                           |
|----------------|----------------|------------------------------|
| NUMBER(n)      | 숫자 데이터         | 최대 n byte 정수                 |
| CHAR(size)     | 고정길이 문자열       | 최대 255문자                     |
| VARCHAR2(size) | 가변 길이 문자열 데이터  | 영문 4,000 문자<br>(한글 2,000 문자) |
| DATE           | 날짜형(형식은 지정 가능) |                              |

---

# 오라클 SQL 언어의 종류

| 구 분   | 문 장                             | 설 명                 |
|---|---------------------------------|---------------------|
| DQL(데이터 질의어;<br>Data Query Language)              | SELECT                          | 검색시 사용              |
| DML(데이터 조작어;<br>Data Manipulation<br>Language)    | INSERT<br>UPDATE<br>DELETE      | 변경시 사용              |
| DDL(데이터 정의어;<br>Data Definition<br>Language)      | CREATE<br>ALTER<br>DROP         | Object의 생성과 변경시     |
| TCL(트랜잭션 제어어;<br>Transaction Control<br>Language) | COMMIT<br>ROLLBACK<br>SAVEPOINT | Transaction 종료 및 취소 |
| DCL(데이터 제어어;<br>Data Control Language)            | GRANT<br>REVOKE                 | 권한 부여 및 취소          |

# SELECT 문장

---

## <Syntax>

```
SELECT [DISTINCT] {*, column [alias], ...}  
FROM   table_name  
[WHERE condition]  
[GROUP BY group_by_expression]  
[HAVING group_condition]  
[ORDER BY {column, expression} [ASC | DESC]];
```

[참고]

◆ 산술 표현식

: 데이터가 출력되는 방식을 수정하거나 계산을 수행하고자 할 때 사용  
그리고 SELECT 문장에서는 FROM 절을 제외한 SQL 문장의 절에서  
사용 가능

---

# INSERT 문장

---

## <Syntax>

```
INSERT INTO table_name (cname1, . . . , cnameN)  
VALUES (value1, . . . , vlaueN);
```

---

# UPDATE 문장

---

## <Syntax>

```
UPDATE table_name  
SET cname1=value1, . . . , cnameN=valueN  
WHERE condition;
```

---

# DELETE 문장

---

## <Syntax>

```
DELETE table_name  
WHERE condition;
```

---

# Class 클래스

---

- 이 클래스는 일반적으로 클래스의 객체에 대한 정보를 갖고 있는 메타 클래스이다.
  - 프로그램이 실행시에 객체를 생성하려면 JVM은 그 Type의 Class 객체가 메모리에 로드되었는지 먼저 확인하고, 로드되지 않았다면 class 파일을 찾아서 로드한다.
-

# 1. 드라이버의 로드

---

- ❑ `Class.forName("oracle.jdbc.driver.OracleDriver");`
- ❑ `forName()` 메소드는 클래스이름을나타내는 `String` 인자를 받으며, 그 클래스의 정보를 갖고 있는 `Class` 객체 참조를 리턴한다
- ❑ `static Class.forName(String className)` throws **`ClassNotFoundException`**



# OracleDriver.class 파일

---

[참고]

C:\oracle\exe\app\oracle\product\10.2.0\server\jdbc\lib  
폴더에 **ojdbc14.jar** 파일이 존재한다.

압축을 풀면 oracle\jdbc\driver 폴더에  
OracleDriver.class 파일 존재함.

---

## 2. Connection 생성하기

---

□ DriverManager 클래스는

- Connection 클래스 객체 생성 역할 => getConnection()  
메소드 이용

▶ Connection 클래스 객체는 데이터베이스와의 실제 연결을 객체 모델링한 것이기 때문에 매우 중요하다.

이 Connection 클래스 객체가 생성되어야, 비로서 DB에 접근 가능  
<사용 예>

```
Connection conn = null;
```

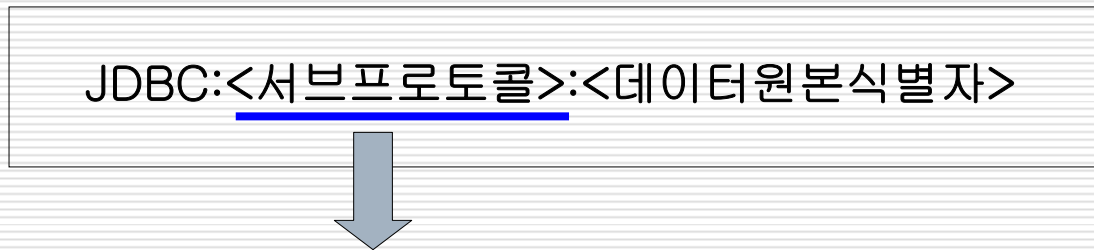
```
conn = DriverManager.getConnection(url,user,passwd);
```

---

# JDBC URL

---

- JDBC URL은 데이터베이스에 대한 다양한 정보를 포함하고 있다.
- JDBC URL 구조



oracle:thin:@IP주소:포트번호

[참고]

포트번호: 오라클 DB에서 네트워크를 통한 접속을 처리하기 위해 실행되어 있는 **리스너 사용포트**로, 기본 값은 1521이다.

---

### 3. 실제 SQL 문의 실행

---

- Statement 클래스에 의해서 수행
- 먼저, Connection 클래스 객체를 이용해서 Statement 객체 생성  
=> createStatement() 메소드 사용

<사용 예>

```
Statement stmt = null;  
stmt = conn.createStatement();
```

---

## 4. 데이터의 검색(Select)

---

- 데이터를 검색하기 위해서는 Select문을 사용하여, Statement 객체에 요청해주어야 한다.
- Select문은 이미 존재하는 데이터를 검색하는 기능이기 때문에 executeQuery( ) 메소드를 실행해 주어야 한다.
- executeQuery( ) 메소드의 결과로 ResultSet 이라는 객체가 얻어진다.

<사용 예>

```
ResultSet rs = null;  
rs = stmt.executeQuery(query);
```

# 데이터의 검색(Select)

- 보통, next() 메소드를 실행하여 레코드간을 움직이고, 각 레코드의 필드에 대한 접근은 getXxx() 메소드를 이용하여 접근한다.

<사용 예>

```
while(rs.next()) {  
    System.out.println(rs.getInt("empno")+"\\t"+  
        rs.getString("ename") + "\\t" +  
        rs.getFloat("sal") + "\\t" +  
        rs.getInt("deptno"));  
}
```

# 실습 전 확인 작업

---

(1) 명령 프롬프트에서

> netstat -a

TCP 컴퓨터이름 : 1521 컴퓨터이름 : 0 LISTENING

(2) [시작] – 제어판- 시스템 및 보안 - 관리도구 – [서비스] 에서 확인

|                     | 상태    | 시작유형  |
|---------------------|-------|-------|
|                     | ----- | ----- |
| OracleXETNSListener | 시작됨   | 자동    |
| OracleServiceXE     | 시작됨   | 자동    |

---

# 오라클 DB에 Data Insert 작업

---

DB 연동 테스트

추가

번호

이름

직책

부서번호

이메일



# 스레드(Thread) 개요

---

- 프로세스 내의 실행 단위가 되는 일련의 작업 흐름을 의미한다
- 하나의 프로세스에서 여러 개의 스레드를 동작 시킬수 있는데 이를 다중 스레드라 한다

[다중 스레드의 예]

- V3 Lite ▶ USB 드라이브 검사(8개의 스레드로 파일을 검사)
  - 파일을 다운 받을 수 있는 FTP 클라이언트 프로그램
-

# 자바 스레드(Thread)

---

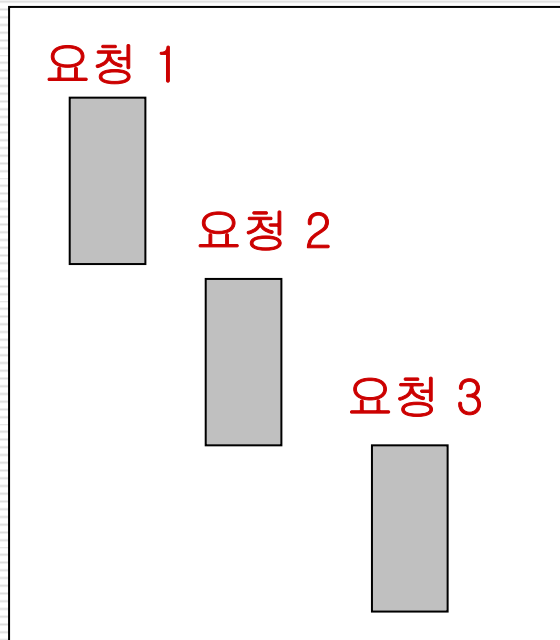
- 하나의 프로그램이 동시에 여러 개의 일을 수행할 수 있도록 해주는 기법
- 자바는 언어차원에서 스레드가 지원되기 때문에 **안전성**과 **효율성**이 보장됨
- Multi-Threading이 지원되면

- 하나의 프로그램내에서 여러 개의 일을 동시에 수행하는 것이 가능하고
- 또 스레드간에 데이터 공유가 가능하므로 멀티프로세스 시스템보다 더 효율적으로 프로그램을 작성 가능하다.

# 인터넷 서버 프로그램 작성시

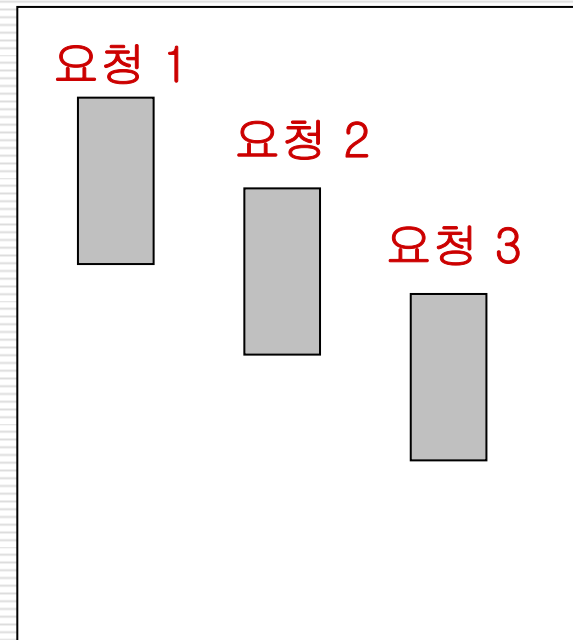
---

“순차적으로 요구 처리”



▲ 스레드를 사용하지 않는 구조

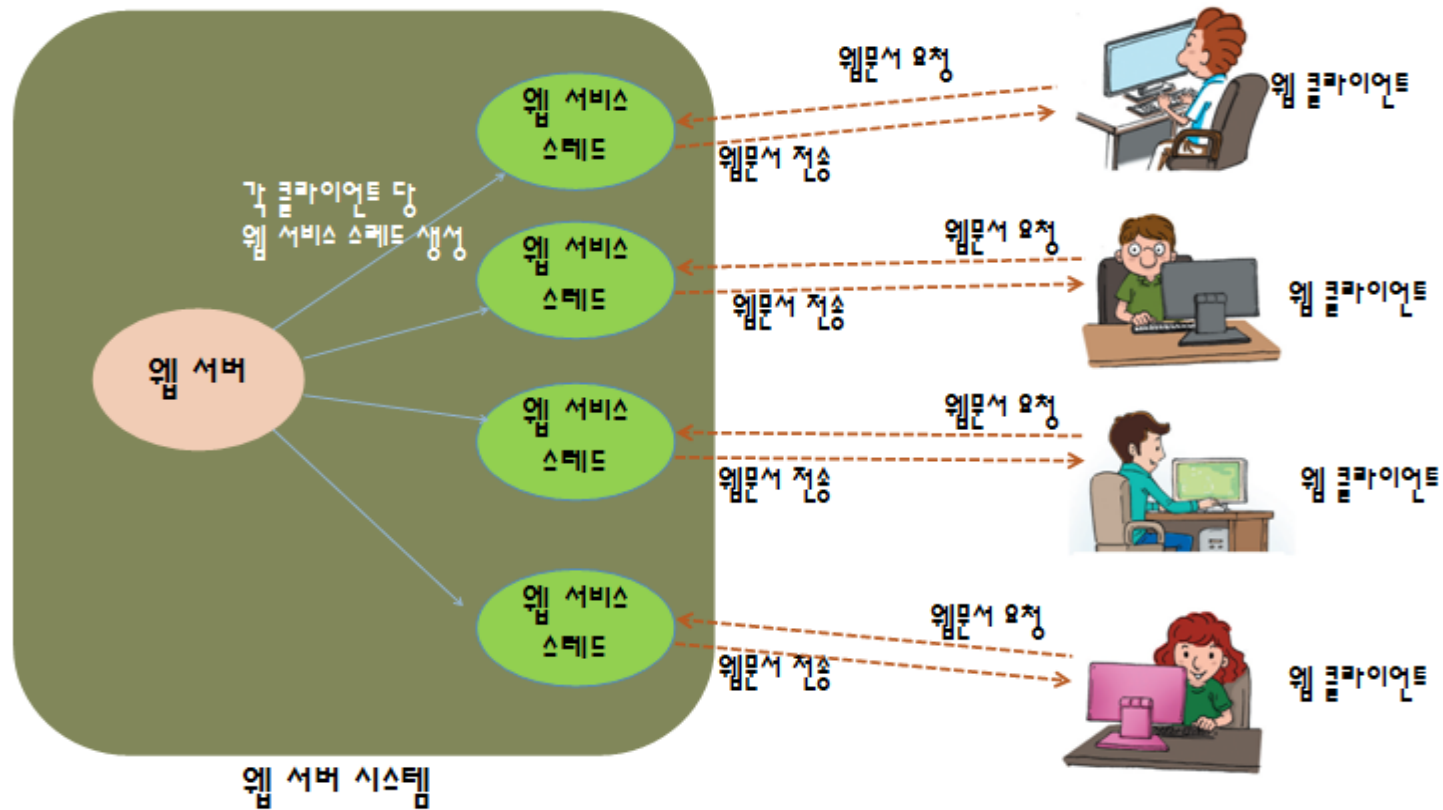
“각 요구를 처리하는 스레드를 만들어 각각을 처리해주면 된다.” – 응답속도 빠름



▲ 스레드를 사용하는 구조

---

# 웹 서버의 멀티스레딩 사례



# 자바의 스레드 지원 방식

---

- (1) `java.lang.Thread` 클래스를 사용하는 방법
- (2) `java.lang.Runnable` 인터페이스를 사용하는 방법

## <차이점>

- (1) 방법: 스레드를 지원하고자 하는 클래스를 생성시  
즉 `Thread`를 상속받는 class를 `start()` 시킨다.
  - (2) 방법: `Runnable`를 구현한 class를 `new Thread()`의 인수로 주어서  
새로 만든 다른 클래스로부터 상속이 필요한 경우 `Thread`  
객체를 `start()` 시키는것
-

# 스레드의 구현과 실행

---

- 스레드를 실행시키기 위해서는 스레드 객체의 `start()` 메소드를 호출함으로써 실행된다
  - `start()` 메소드를 호출하면 `run()` 메소드가 자동 호출되는데 `run()` 메소드에는 다중 스레드로 동작시키고자 하는 내용이 기술되어 있다
-

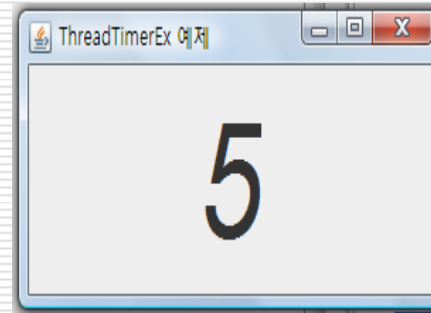
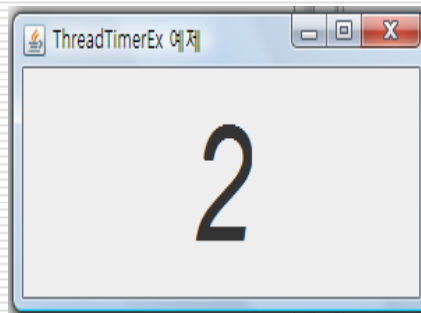
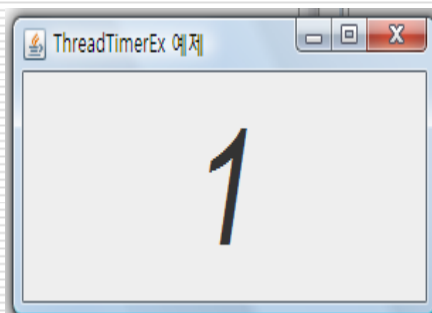
# Thread 클래스를 사용하는 방법

---

- (1) `java.lang.Thread` 클래스를 상속 받아서 처리를 한다
- (2) 상속받은 `Thread` 클래스 `run()` 메서드로 오버라이딩 해준다.
- (3) `run()` 메서드에 스레드가 일할 작업의 내용을 기술한다  
시간이 오래 걸리는 일을 다중 스레드로 구현해야 의미가 있기  
때문에 `run()` 메서드에는 일반적으로 반복문(`while`문)을 기술한다

# Thread를 상속받아 1초 단위의 타이머 레이블 만들기

---





# 소켓(Socket) 인터페이스

---

- 자바에서는 네트워크 프로그램을 작성할 수 있도록 하기 위해서 소켓 인터페이스를 제공한다

## (1) 스트림(Stream)소켓 -> TCP

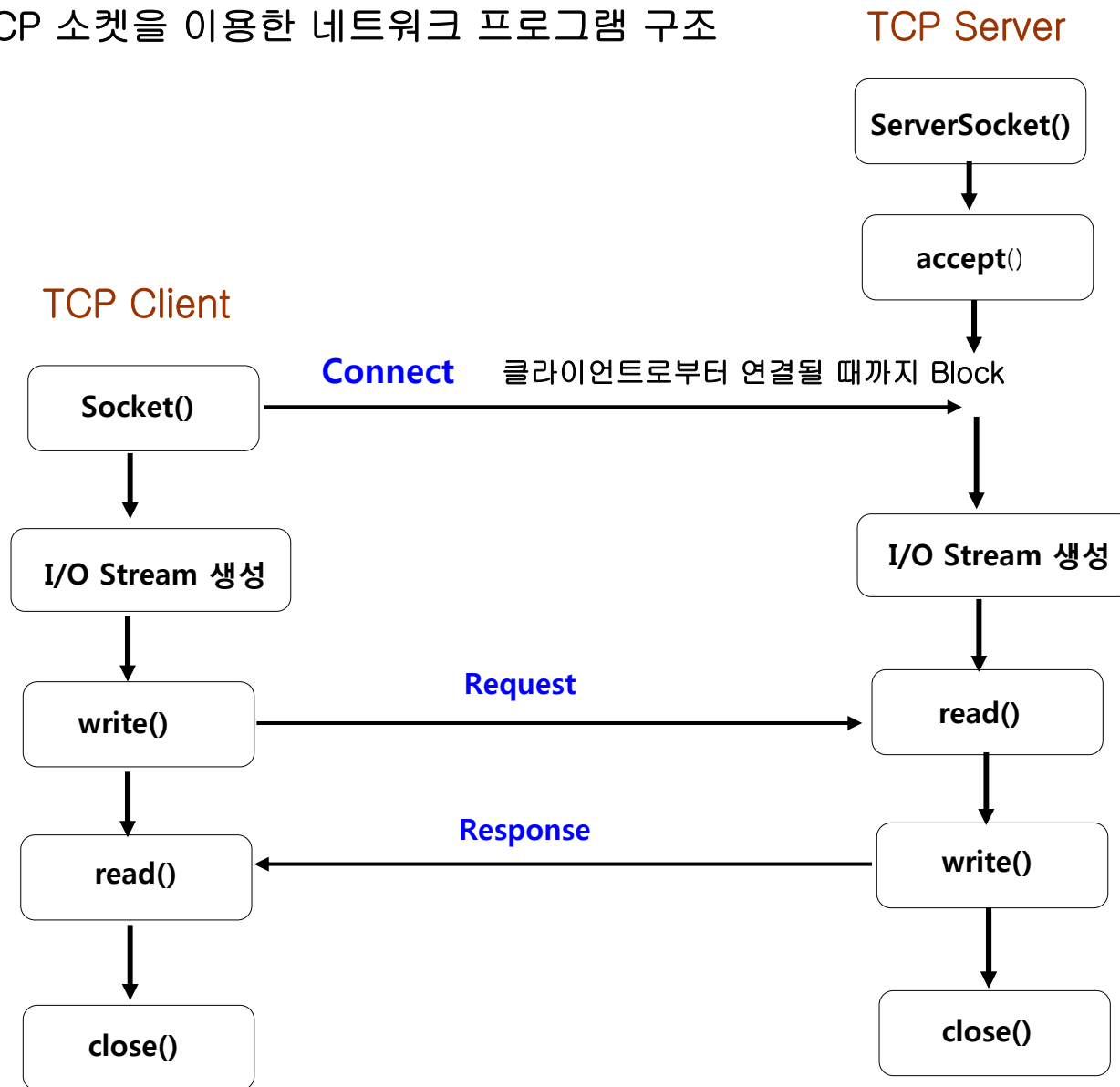
계속 연결하고 있는 상태에서 데이터를 송수신하는 방식  
예) 메신저

## (2) 데이터그램(Datagram)소켓 -> UDP

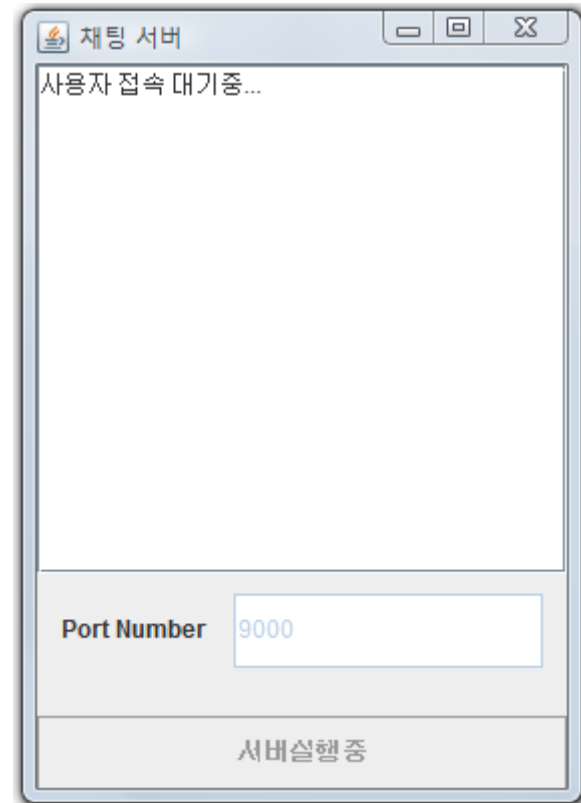
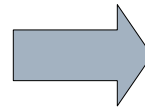
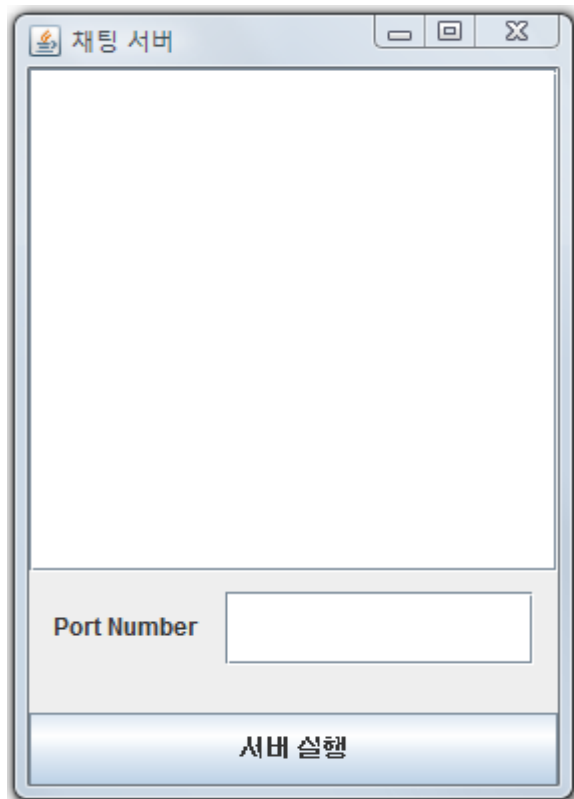
연결 상태와 데이터의 손실 여부를 체크하지 않는다는 특징  
예) 화상 채팅, 동영상 서비스

---

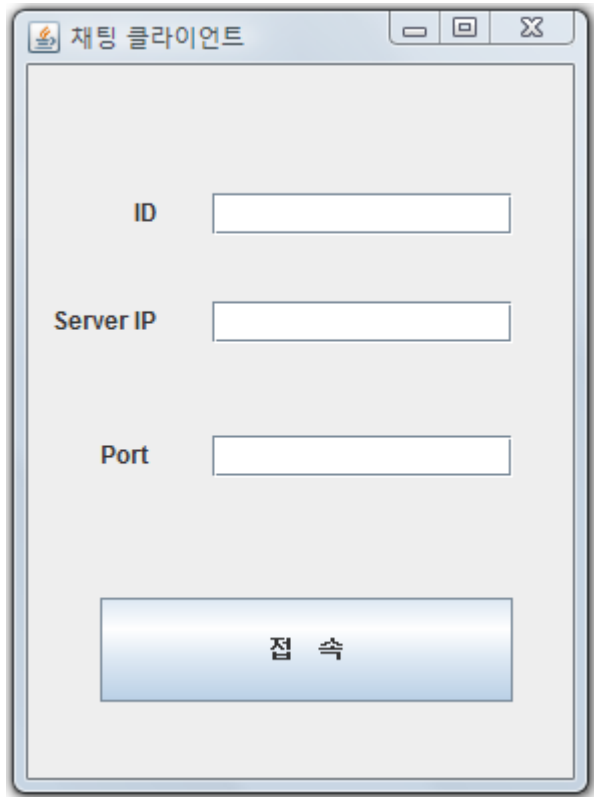
## ■ TCP 소켓을 이용한 네트워크 프로그램 구조



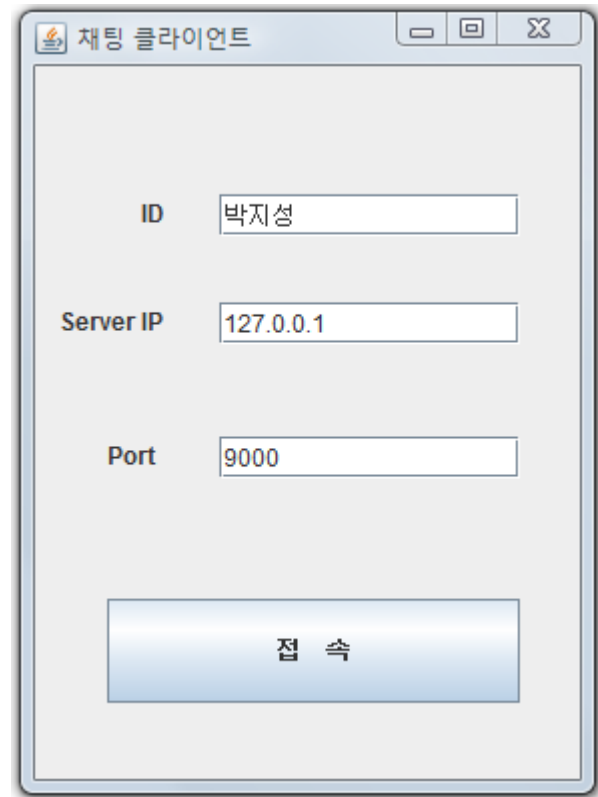
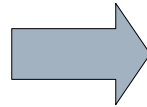
## [채팅 서버 실행]



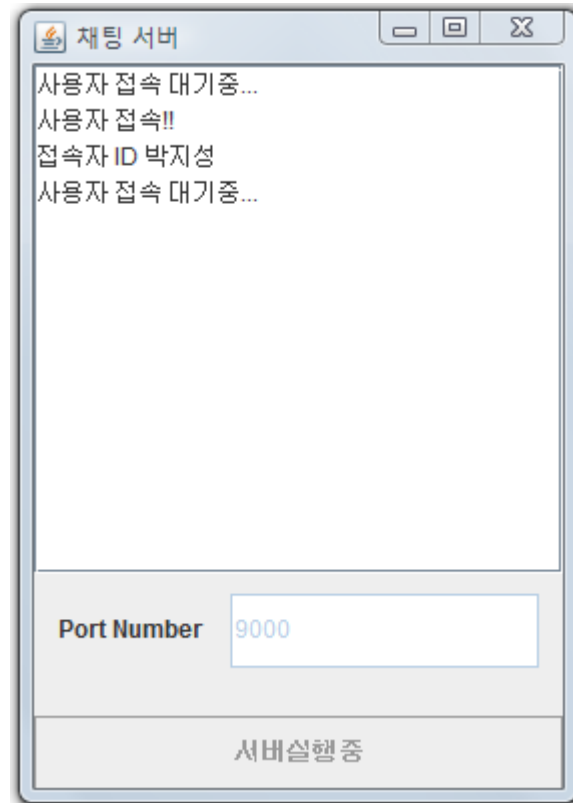
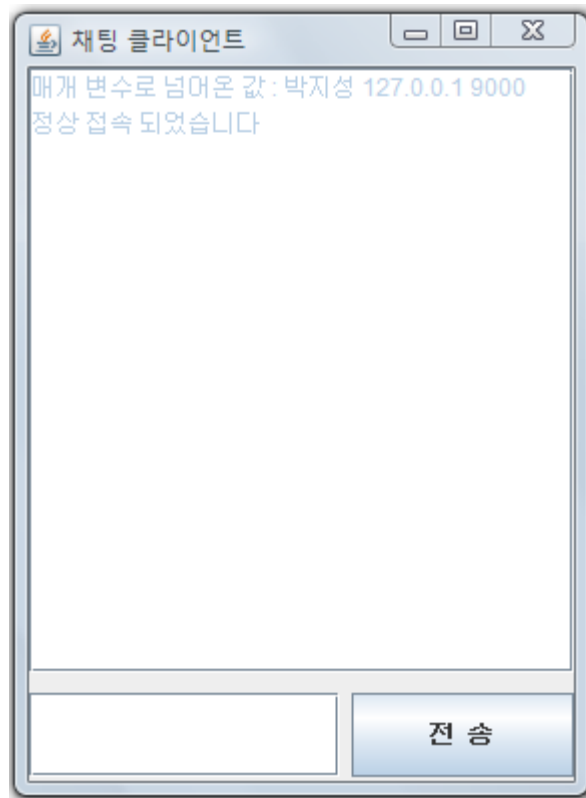
## [채팅 클라이언트 실행]

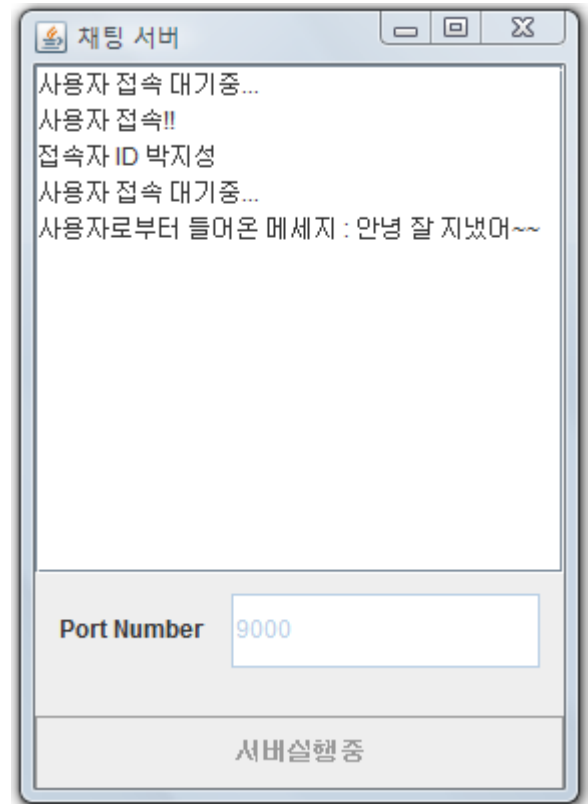
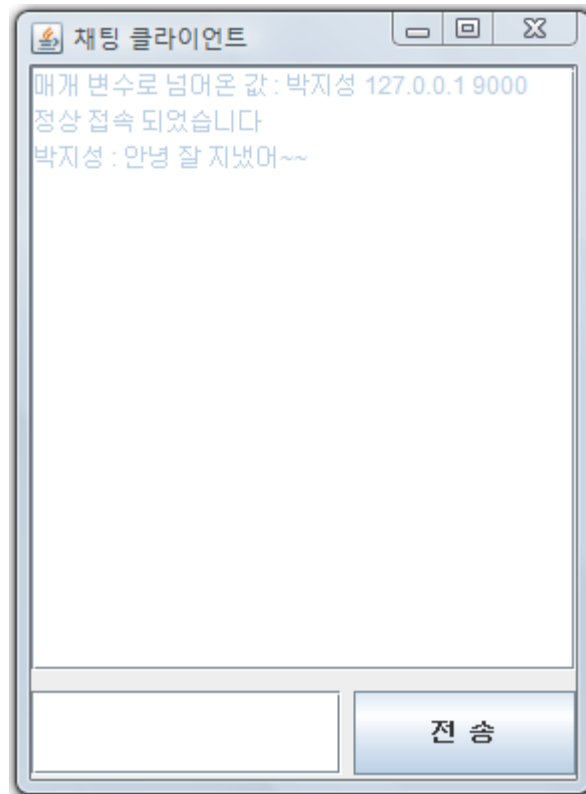
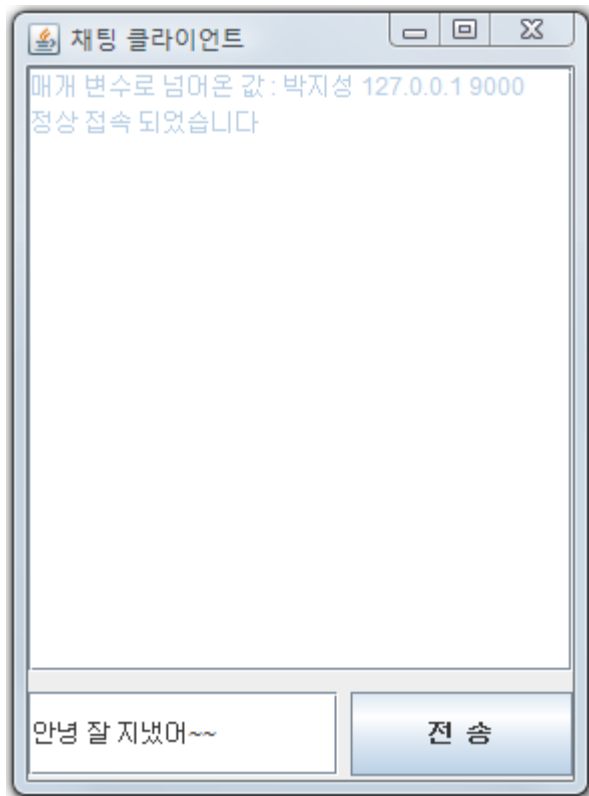


A screenshot of a Windows-style application window titled "채팅 클라이언트" (Chat Client). The window has a light gray background and a blue header bar with standard window controls (minimize, maximize, close). Inside the window, there are three input fields stacked vertically, each with a label to its left: "ID", "Server IP", and "Port". All three input fields are currently empty. At the bottom of the window, there is a large blue button with the text "접속" (Connect) in white.



A screenshot of the same "채팅 클라이언트" (Chat Client) window, but now the input fields are filled with text. The "ID" field contains "박지성", the "Server IP" field contains "127.0.0.1", and the "Port" field contains "9000". The "접속" (Connect) button remains at the bottom.





채팅 클라이언트

ID

Server IP

Port

접속

채팅 클라이언트

매개 변수로 넘어온 값 : 기성용 127.0.0.1 9000  
정상 접속 되었습니다

전송

