

{ JAVA }

선린인터넷

웹 운영 과

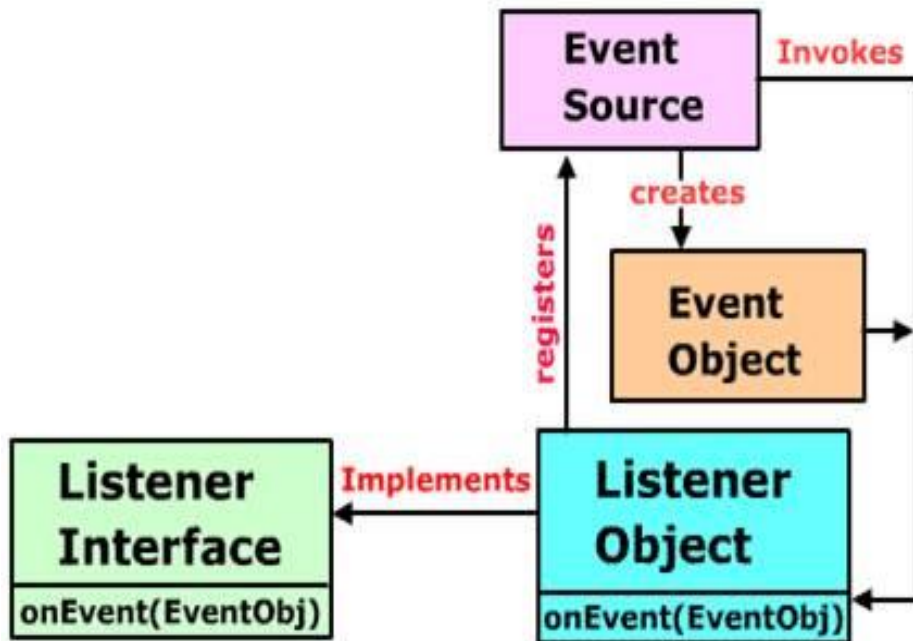
그래픽 사용자

인터페이스 : GUI-5

- 어댑터 클래스
- 컬렉션 개요
- 내부클래스 개요
- 스레드 개요

복습 Event-driven Programming : Delegation model

- ❖ 사용자의 행동에 따라 컴포넌트에서 발생할 수 있는 이벤트 종류가 결정되어 있음 → 교과서 202쪽 표 참조
- ❖ 각 이벤트마다 리스너 인터페이스가 지정되어있음 → 교과서 203쪽 표 참조
- ❖ 리스너 인터페이스를 구현하여 이벤트 소스 객체에 등록하는 방법으로 이벤트 프로그래밍을 함



복습 이벤트객체 - 이벤트소스

이벤트 객체	이벤트 소스 (이벤트 발생 근원)	이벤트가 발생하는 경우
ItemEvent	JCheckBox	체크박스의 선택 혹은 해제
	JCheckBoxMenuItem	체크박스 메뉴 아이템의 선택 혹은 선택
	JList	리스트 아이템 선택
ActionEvent	JButton	마우스나 키로 버튼 선택
	JList	리스트 아이템을 더블클릭하여 아이템 선택
	JMenuItem	메뉴 아이템 선택
	JTextField	텍스트 입력 중 [Enter]키 입력
KeyEvent	Component	키가 눌러지거나 눌러진 키가 떨어질 때
MouseEvent	Component	마우스 버튼이 눌러지거나 떨어질 때 마우스 버튼 이 클릭될 때 컴포넌트 위에 마우스가 올라갈 때/올라간 마우스가 내려올 때 마우스 드래그가 실행될 때 마우스가 단순히 움직일 때
FocusEvent	Component	컴포넌트가 포커스를 받거나 잃을 때
TextEvent	TextField	텍스트 변경
	TextArea	
WindowEvent	Window	컴포넌트에 대한 윈도우 활성화/비활성화 아이콘 화/아이콘에서 복구 윈도우 열기/닫기/종료
AdjustmentEvent	JScrollBar	스크롤바 이동시
ComponentEvent	Component	컴포넌트가 사라지거나/나타나거나/변경되거나/이동할 때
ContainerEvent	Container	Container에 컴포넌트가 추가/삭제될 때

복습 이벤트객체 - 리스너 인터페이스(추상메소드: 이벤트 핸들러)

이벤트 종류	리스너 인터페이스	리스너의 추상 메소드
ItemEvent	ItemListener	void itemStateChanged(ItemEvent)
ActionEvent	ActionListener	void actionPerformed(ActionEvent)
KeyEvent	KeyListener	void keyPressed(KeyEvent)
		void keyReleased(KeyEvent)
		void keyTyped(KeyEvent)
MouseEvent	MouseListener	void mousePressed(MouseEvent)
		void mouseReleased(MouseEvent)
		void mouseClicked(MouseEvent)
		void mouseEntered(MouseEvent)
		void mouseExited(MouseEvent)
FocusEvent	FocusListener	void focusGained(FocusEvent)
		void focusLost(FocusEvent)
TextEvent	TextListener	void textValueChanged(TextEvent)
WindowEvent	WindowListener	void windowOpened(WindowEvent)
		void windowClosing(WindowEvent)
		void windowIconified(WindowEvent)
		void windowDeiconified(WindowEvent)
		void windowClosed(WindowEvent)
		void windowActivated(WindowEvent)
		void windowDeactivated(WindowEvent)
AdjustmentEvent	AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent)
ComponentEvent	ComponentListener	void componentHidden(ComponentEvent)
		void componentShown(ComponentEvent)
		void componentResized(ComponentEvent)
		void componentMoved(ComponentEvent)
ContainerEvent	ContainerListener	void componentAdded(ContainerEvent)
		void componentRemoved(ContainerEvent)

복습 Generic

❖ **Generic** : 꺾쇠 가 바로 제네릭...??? \Longrightarrow $< >$

- "데이터 타입의 일반화"
- 꺾쇠 $< >$ 안에 저장할 클래스 타입을 명시하는 문법
- 제네릭 클래스, 제네릭 메소드
- 제네릭 타입에 제한 걸기
 - extends
 - &
 - super
 - ?

복습 Generic Class

❖ Generic Class

- 클래스 데이터 타입 자리에 T 로 치환하여 표기한다
- T로 치환된 자리의 타입은 인스턴스 생성시 결정된다
- T 자리에는 반드시 클래스 타입만 들어올 수 있다. : 기본자료형 불가
- 코드의 간결성 증대
 - 하나의 클래스로 다양한 타입을 가지는 클래스로 대응 가능
- 데이터 타입에 대해 프로그램 코드의 안전성 증대
 - 인스턴스 생성 시 결정된 타입과 다른 타입이 사용될 경우 컴파일 타임 에러 발생

어댑터(Adapter) 클래스

❖ 어댑터(Adapter) 클래스

- 목적: 이벤트 리스너 인터페이스 구현에 따른 부담 해소
 - 리스너 작성시 추상 메소드들을 모두 구현해야 하는 부담
- 어댑터 클래스
 - 리스너의 모든 메소드가 단순 리턴하도록 구현해놓은 클래스
 - 추상 메소드가 단 하나뿐인 리스너 인터페이스는 어댑터클래스가 없음 : ActionListener
 - 예: MouseAdapter클래스

```
class MouseAdapter implements MouseListener, MouseMotionListener,  
                                MouseWheelListener {  
    public void mousePressed(MouseEvent e) { }  
    public void mouseReleased(MouseEvent e) { }  
    public void mouseClicked(MouseEvent e) { }  
    public void mouseEntered(MouseEvent e) { }  
    public void mouseExited(MouseEvent e) { }  
    public void mouseDragged(MouseEvent e) { }  
    public void mouseMoved(MouseEvent e) { }  
    public void mouseWheelMoved(MouseWheelEvent e) { }  
}
```


❖ 어댑터(Adapter) 클래스

리스너 인터페이스	대응하는 어댑터 클래스
ActionListener	없음
ItemListener	없음
KeyListener	KeyAdapter
MouseListener	MouseListener
MouseMotionListener	MouseMotionAdapter 혹은 MouseAdapter
FocusListener	FocusAdapter
WindowListener	WindowAdapter
AdjustmentListener	없음
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter

❖ 어댑터(Adapter) 클래스

(이전 WhiteBoard 클래스에서...)

```
23 //아래는 어댑터 클래스(리스너 클래스의 일종)를 상속받은 내부클래스
24 class MouseEventHdl extends MouseListener {
25     public void mousePressed(MouseEvent e) {
26         setStartPoint(e.getX(),e.getY());
27     }
28     public void mouseReleased(MouseEvent e) {
29         setEndPoint(e.getX(),e.getY());
30         repaint();
31     }
32 }
33 class MouseMotionHdl extends MouseMotionAdapter {
34     public void mouseDragged(MouseEvent e) {
35         setEndPoint(e.getX(),e.getY());
36         repaint();
37     }
38 }
39 }
```

Collection Framework

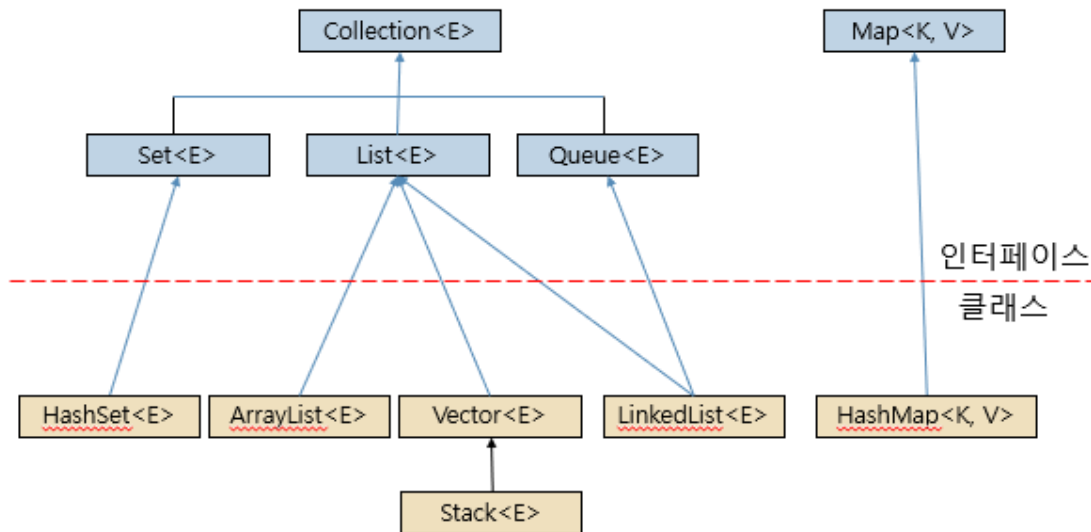
❖ Collection Framework

- 여러가지 **자료구조**를 미리 구현하여 제공하는 클래스 (이미 만들어진 기능?)
- 객체(인스턴스)만 저장할 수 있음

■ 계층 구조

- Collection 계열
- Map 계열

■ 참고자료:예제들



❖ Collection Framework: 각 인터페이스의 특징 <http://hackersstudy.tistory.com/26>

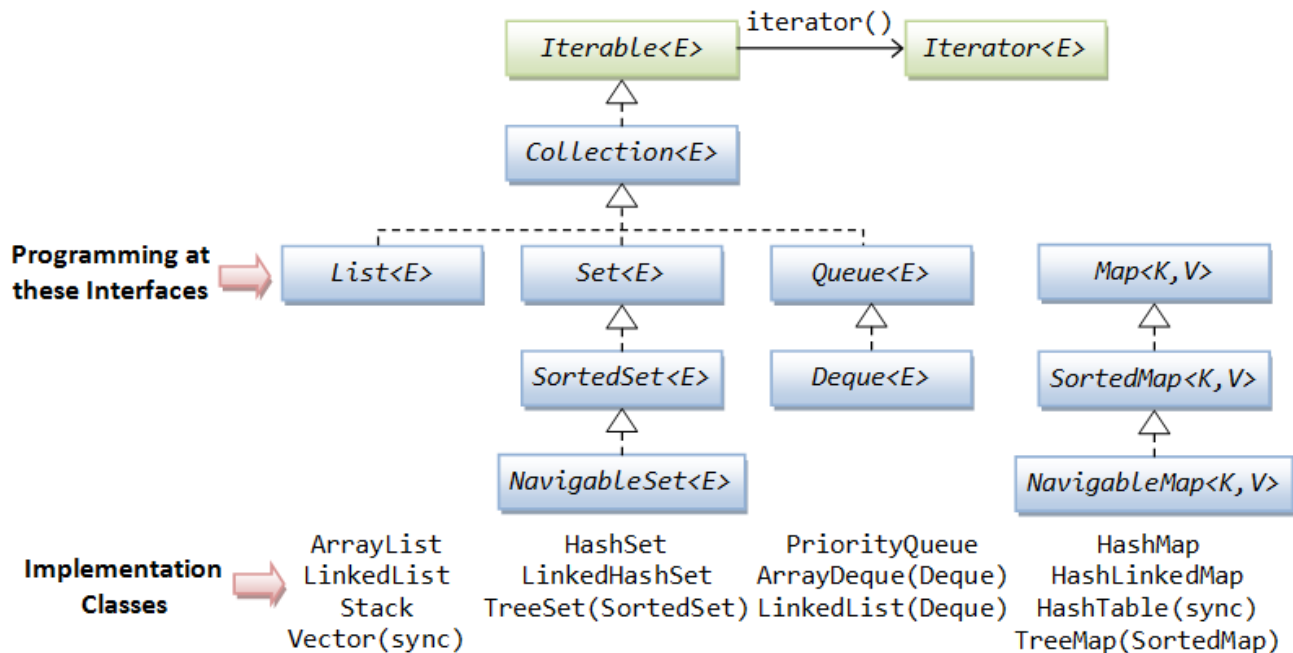
인터페이스	구현 클래스	특징
List	LinkedList Stack Vector ArrayList	순서가 있는 데이터의 집합, 데이터의 중복을 허용한다.
Set	HashSet TreeSet	순서를 유지하지 않는 데이터의 집합, 데이터의 중복을 허용하지 않는다.
Map	HashMap TreeMap HashTable Properties	키(key)와 값(value)의 쌍으로 이루어진 데이터의 집합이다. 순서는 유지되지 않고, 키는 중복을 허용하지 않으며 값의 중복을 허용한다.

❖ Collection Framework : Collection 인터페이스에 선언된 주요 추상 메소드

추상메소드	설명
<code>boolean add(E e)</code>	객체(데이터)를 콜렉션에 추가
<code>void clear()</code>	현재 콜렉션의 모든 객체(데이터)를 제거
<code>boolean isEmpty()</code>	현재 콜렉션에 객체(데이터)가 있는지 여부
<code>Iterator<E> iterator()</code>	현재 콜렉션의 객체들을 iterator로 반환
<code>boolean remove(Object o)</code>	현재 콜렉션에서 o로 전달된 객체를 제거
<code>int size()</code>	현재 콜렉션이 가지고 있는 객체(데이터) 개수
<code>Object[] toArray()</code>	현재 콜렉션의 객체들을 배열로 만들어 반환

❖ Collection Framework : **Iterator** 이터레이터

- https://www.ntu.edu.sg/home/ehchua/programming/java/J5c_Collection.html



Stack 콜렉션 예제

```
3 import java.util.Stack;
4
5 public class StackEx {
6     public static void main(String[] args) {
7         Stack<String> col = new Stack<>();
8         col.push("홍길동");
9         col.push("이몽룡");
10        col.push("성춘향");
11        System.out.println(col.pop());
12        System.out.println(col.pop());
13        System.out.println(col.pop());
14    }
15 }
```

Console

<terminated>

성춘향

이몽룡

홍길동

HashSet 컬렉션 예제

```
3 import java.util.HashSet;
4 import java.util.Iterator;
5
6 public class HashSetEx {
7     public static void main(String[] args) {
8         HashSet<String> ex = new HashSet<>();
9         ex.add("빨간색");
10        ex.add("파란색");
11        ex.add("노란색");
12        System.out.println("컬렉션 크기: " + ex.size());
13
14        System.out.println("== 이테레이터로 출력 ==");
15        Iterator<String> myIterator = ex.iterator();
16        while(myIterator.hasNext())
17            System.out.println(myIterator.next());
18    }
19
20 }
```

Console Problems

<terminated> HashSetEx [Java App]

컬렉션 크기: 3

== 이테레이터로 출력 ==

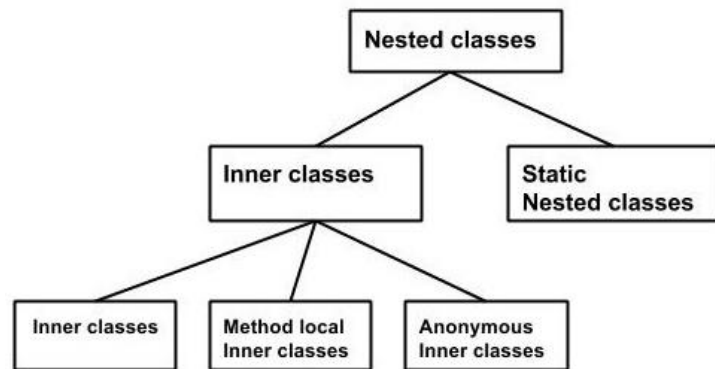
노란색

파란색

빨간색

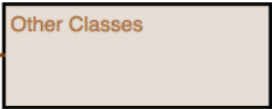
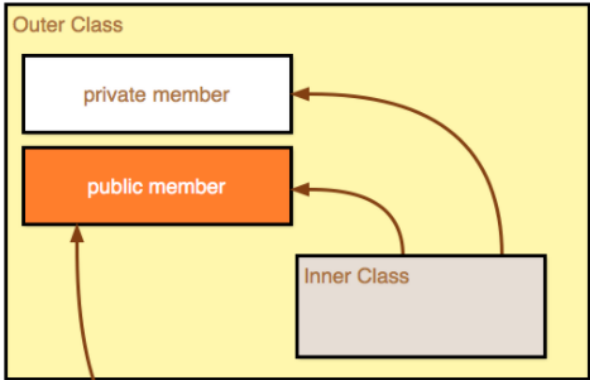
내부 클래스

```
public class Class_A { // 외부클래스  
    public class Class_B{ // 내부클래스  
    }  
}
```



❖ 내부클래스

- 특징: 외부클래스의 멤버에 자유롭게 접근 가능
- 용도
 - 클래스 자체를 멤버로 사용
 - 일반적으로 화면의 이벤트를 처리하는 **이벤트 핸들러**로 사용



You can see and use both public and private members, but you don't inherit them.

종류	특징
Member	외부 클래스의 멤버로 정의되는 내부 클래스
Local	외부 클래스의 메서드 내에서 정의되는 내부 클래스
Static	static으로 선언된 내부 클래스
Anonymous	이름이 없이 정의된 내부 클래스

멤버 내부 클래스

❖ 멤버 내부 클래스

- 일반적인 내부 클래스
- **외부 클래스의 멤버**처럼 정의된 클래스
- 반드시 외부 클래스가 먼저 생성이 된 후 내부 클래스를 생성해야 함
- 장점 : 클래스의 캡슐화 강도가 최대가 됨

멤버 내부 클래스

```
3 class Outer { // 외부클래스
4     private String name;
5     public Outer(String name) {
6         this.name = name;
7     }
8     class Inner { // 내부클래스 (멤버 내부 클래스)
9         private int age;
10        public Inner(int age) {
11            this.age = age;
12        }
13        public void printInfo() {
14            System.out.println("name: "+name+", age: "+age);
15        }
16    }
17 }
18 public class InnerApp {
19     public static void main(String[] args) {
20         Outer out = new Outer("홍길동"); // 외부클래스
21         Outer.Inner in = out.new Inner(19); // 내부클래스
22         // Outer.Inner in = new Outer("홍길동").new Inner(19);
23         in.printInfo();
24     }
25 }
```

정적 내부 클래스

❖ 정적 내부 클래스

- 내부 클래스가 **static**으로 지정된 클래스
- 내부 클래스 멤버가 static으로 지정되었으면 내부 클래스도 반드시 static으로 지정해야 함
- 외부 클래스를 인스턴스로 생성하지 않아도 내부 클래스를 생성할 수 있음
- 정적 내부 클래스에서는 외부 클래스의 static 변수와 static 메소드만 접근 가능

GUI

정적 내부 클래스

```
3 class Outter {
4     private String name;
5     public Outter(String name) {
6         this.name = name;
7     }
8     static class Inner { // 내부클래스 (정적 내부 클래스)
9         private int age;
10        static String dept = "웹운영과";
11        public Inner(int age) {
12            this.age = age;
13        }
14        public void printInfo() {
15            System.out.println("소속: "+dept+", age: "+age);
16        }
17    }
18 }
19 public class StaticApp {
20     public static void main(String[] args) {
21         Outter.Inner in = new Outter.Inner(20);
22         in.printInfo();
23         System.out.println(Outter.Inner.dept);
24     }
25 }
```

지역 내부 클래스

❖ 지역 내부 클래스

- 외부 클래스의 메소드 내에서 정의된 클래스
- 메서드 호출시 생성되고, 복귀하면 소멸됨 : 메서드 종료시 메모리에서 삭제됨
- 지역변수와 같은 특성을 가지기에, 선언된 메서드 내부에만 인스턴스 생성 가능

GUI

지역 내부 클래스

```
3 class Outer {
4     private String name;
5     public Outer(String name) {
6         this.name = name;
7     }
8     public void method() {
9         class Inner { // 내부클래스 (지역 내부 클래스)
10             private int age;
11             public Inner(int age) {
12                 this.age = age;
13             }
14             public void printInfo() {
15                 System.out.println("name: "+name+", age: "+age);
16             }
17         }
18         Inner in = new Inner(19);
19         in.printInfo();
20     }
21 }
22 public class LocalApp {
23     public static void main(String[] args) {
24         Outer out = new Outer("홍길동");
25         out.method();
26     }
27 }
```

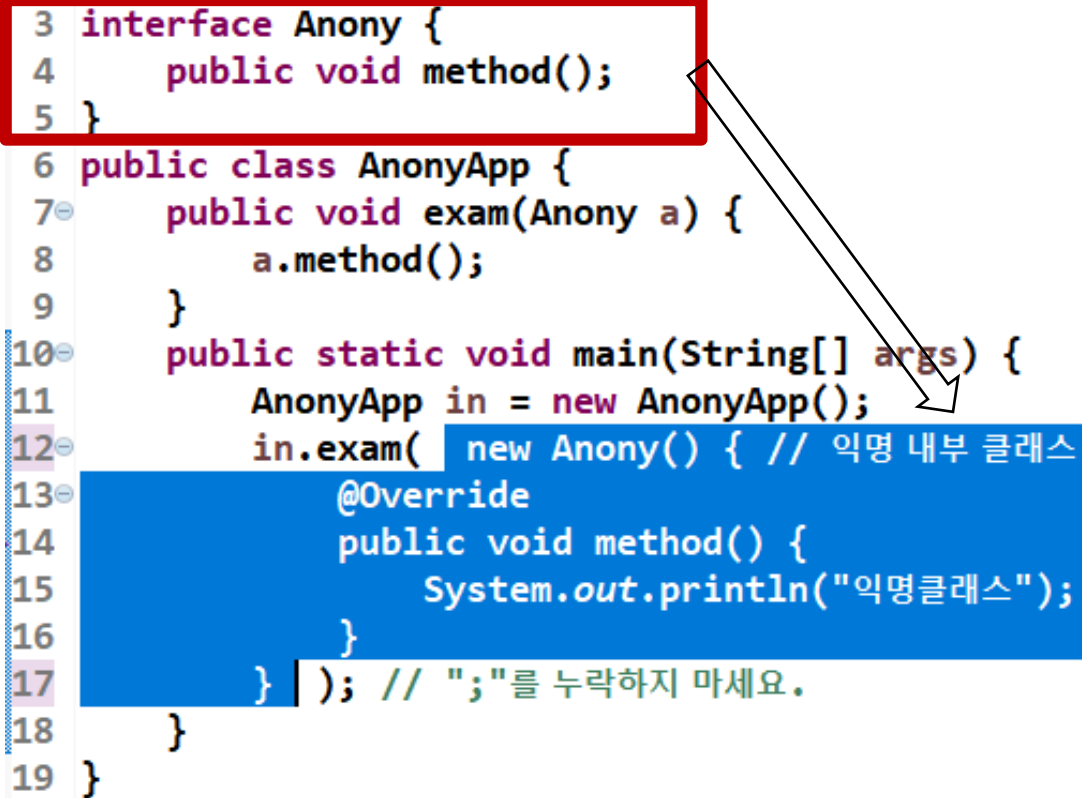
익명(무명) 내부 클래스

❖ 익명(무명) 내부 클래스 : 지역 내부 클래스의 변형된 형태(?!)

- 외부 클래스의 메소드 내에서 선언되지만, 클래스 선언이 아예 없는 특이한 구조
 - 클래스 선언이 되지 않기 때문에, **클래스의 이름이 없음**
 - **인스턴스 생성**과 **구현 부분**만 가짐
- 사용 예
 - 인터페이스를 구현하는 클래스의 사용빈도가 극히 적을 경우
 - 한번만 사용되는 인스턴스의 경우

익명(무명) 내부 클래스

```
3 interface Anony {  
4     public void method();  
5 }  
6 public class AnonyApp {  
7     public void exam(Anony a) {  
8         a.method();  
9     }  
10    public static void main(String[] args) {  
11        AnonyApp in = new AnonyApp();  
12        in.exam( new Anony() { // 익명 내부 클래스  
13            @Override  
14            public void method() {  
15                System.out.println("익명클래스");  
16            }  
17        } ); // ";"를 누락하지 마세요.  
18    }  
19 }
```



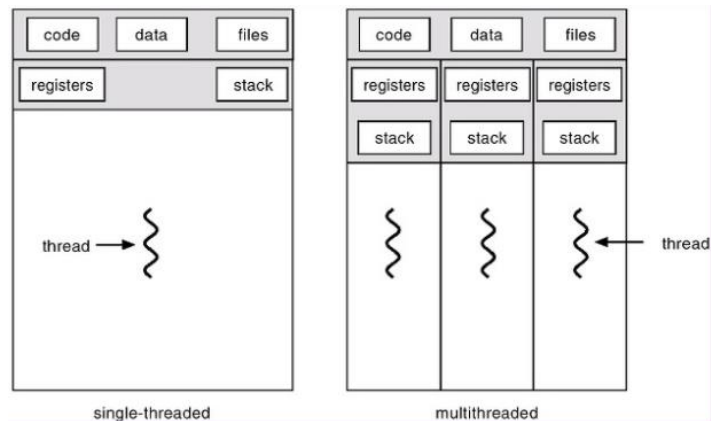
GUI

익명(무명) 내부 클래스

이벤트 핸들러 사용 예시

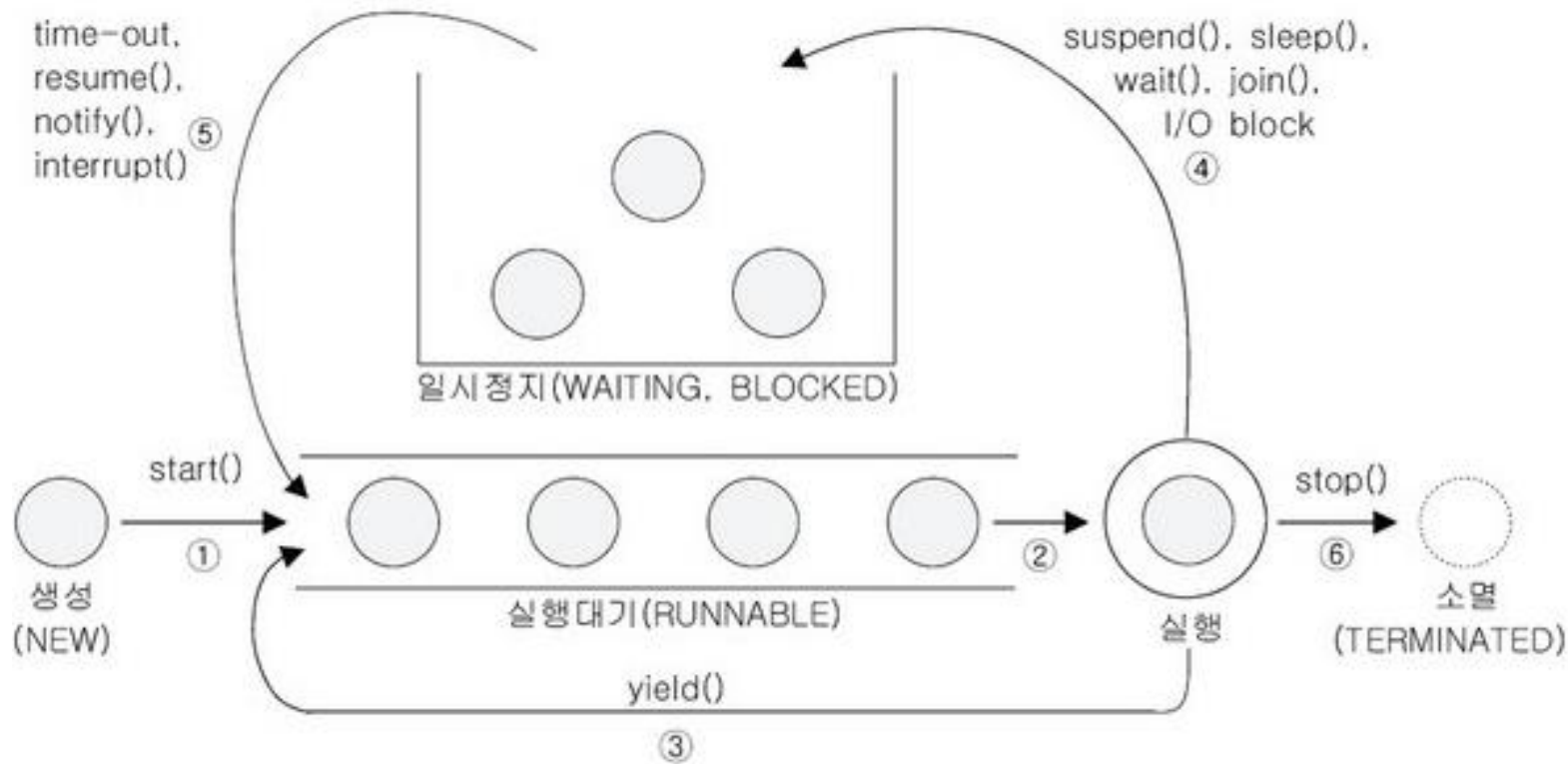
```
3 import java.awt.event.*;
4 import javax.swing.*;
5 public class Exam extends JFrame {
6     public Exam() {
7         setSize(200, 200);
8         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9         JButton button = new JButton("누르세요");
10        add(button);
11        button.addActionListener( new ActionListener() {
12            @Override
13            public void actionPerformed(ActionEvent e) {
14                if(button.getText().equals("누르세요"))
15                    button.setText("Click now");
16                else button.setText("누르세요");
17            }
18        });
19        setVisible(true);
20    }
21    public static void main(String[] args) {
22        new Exam();
23    }
24 }
```

Thread



- * 스레드 = 프로세스 내에서 실행되는 세부 실행 단위
- * 코드의 재사용성과 데이터 공유 가능
- * 여러 스레드가 동시 실행시 공유 자원에 대한 동기화를 고려해야 함

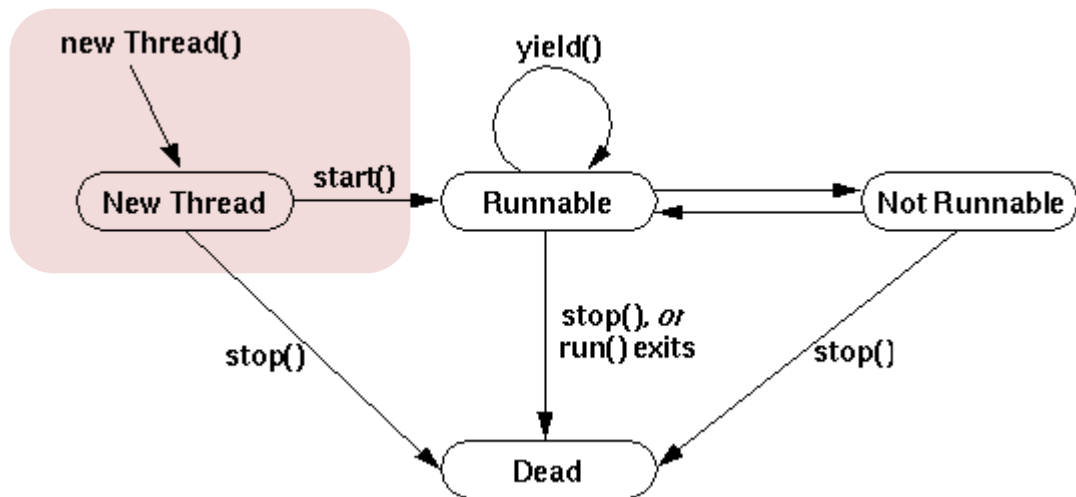
스레드의 상태



스레드

❖ 스레드 생성 방법

- java.lang.Thread 클래스 상속
- java.lang.Runnable 인터페이스 구현



GUI

스레드 생성1 : Thread 상속

① Thread 상속 : run()에 코딩

```
class 클래스명 extends Thread {  
  
    public void run( ) {  
  
        // 스레드 실행시 코드  
  
    }  
  
}
```

② 생성한 객체의 start() 호출

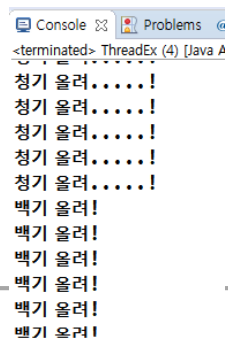
```
3 class Blue extends Thread { // Thread클래스 상속  
4     public void run() { //스레드 스케줄러가 스레드 실행시 호출  
5         while(true) {  
6             System.out.println("청기 올라.....!");  
7         }  
8     }  
9 }  
10 class White extends Thread { // Thread클래스 상속  
11     public void run() { //스레드 스케줄러가 스레드 실행시 호출  
12         while(true) {  
13             System.out.println("백기 올라!");  
14         }  
15     }  
16 }  
17 public class ThreadEx {  
18     public static void main(String[] args) {  
19         // --스레드를 상속한 클래스에서 run()을 구현하면,  
20         // --객체가 스레드 기능 수행 시 run()이 호출돼 실행된다.  
21         // 각 객체의 스레드 실행 : run메소드 X. start메소드 0  
22         new Blue().start();  
23         new White().start();  
24     }  
25 }
```

Console Problems
<terminated> ThreadEx (4) [Java A
청기 올라.....!
청기 올라.....!
청기 올라.....!
청기 올라.....!
청기 올라.....!
백기 올라!
백기 올라!
백기 올라!
백기 올라!
백기 올라!
백기 올라!

GUI

스레드 생성2 : Runnable 구현

- ① Runnable 인터페이스 구현
- ② 구현된 클래스의 run() 코딩
- ③ 위 클래스의 객체를 생성함
- ④ Thread 객체를 별도 생성하면서
매개변수로 ③번 객체를 넣음
- ⑤ ④번 Thread 객체의 start()호출



```
<terminated> ThreadEx (4) [Java A]
청기 올라.....!
청기 올라.....!
청기 올라.....!
청기 올라.....!
청기 올라.....!
백기 올라!
백기 올라!
백기 올라!
백기 올라!
백기 올라!
```

```
3 class Blue implements Runnable { // Runnable 구현
4     public void run() { //스레드 스케줄러가 스레드 실행시 호출
5         while(true) {
6             System.out.println("청기 올라.....!");
7         }
8     }
9 }
10 class White implements Runnable { // Runnable 구현
11     public void run() { //스레드 스케줄러가 스레드 실행시 호출
12         while(true) {
13             System.out.println("백기 올라!");
14         }
15     }
16 }
17 public class ThreadEx {
18     public static void main(String[] args) {
19         // --스레드를 상속한 클래스에서 run()을 구현하면,
20         // --객체가 스레드 기능 수행 시 run()이 호출돼 실행된다.
21         // 각 객체의 스레드 실행 : run메소드 X. start메소드 0
22         new Thread(new Blue()).start();
23         new Thread(new White()).start();
24     }
25 }
```