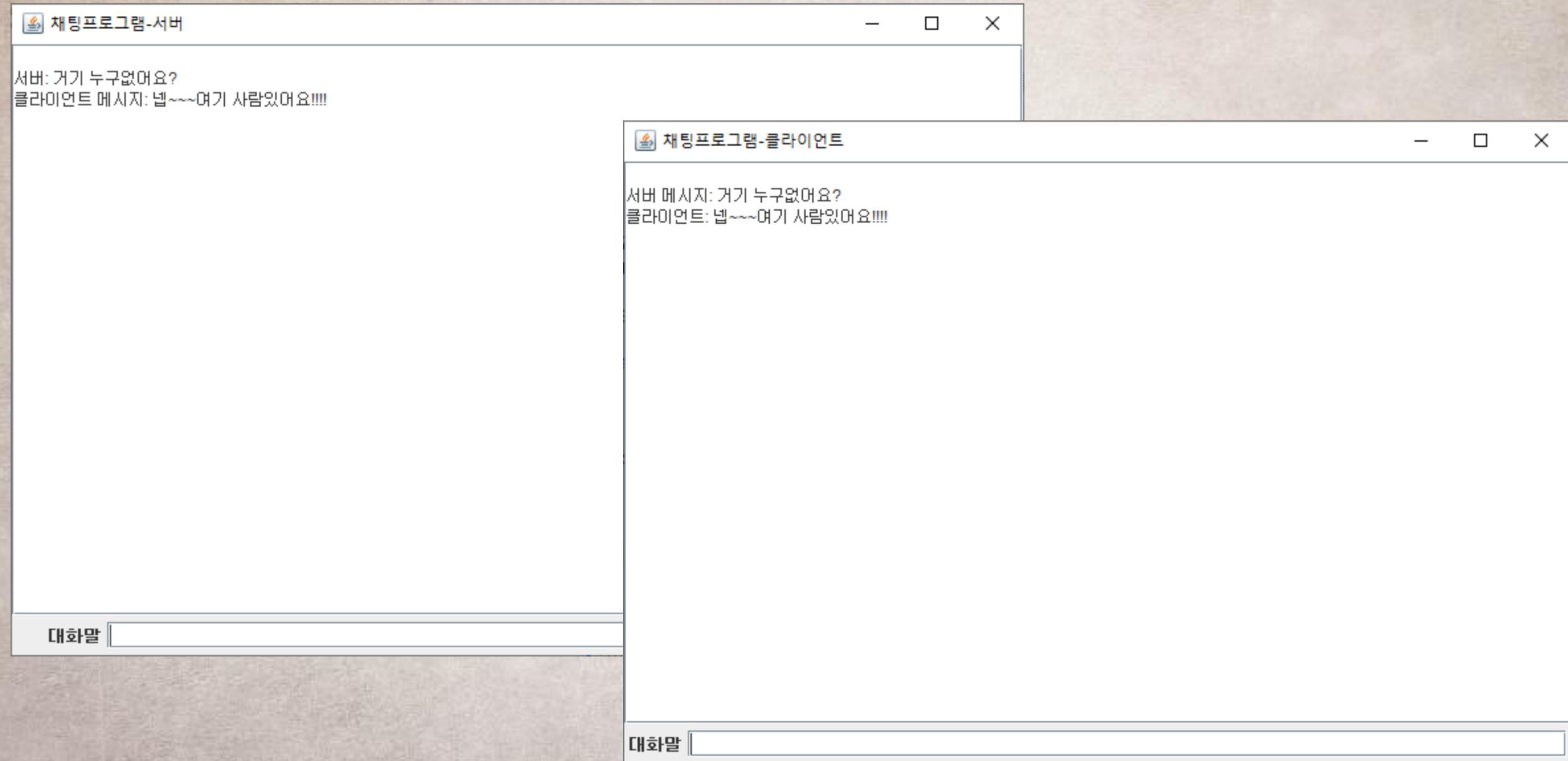


# 쓰레드와 멀티태스킹

comsi.java@gmail.com  
박 경 태

<http://github.com/hopypark>

# 서버와 클라이언트 채팅(1:1)



# 서버-OneToOneS.java

OneToOneS.java OneToOneC.java

```
1⑩ import java.awt.BorderLayout;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import java.io.BufferedReader;
5 import java.io.BufferedWriter;
6 import java.io.IOException;
7 import java.io.InputStream;
8 import java.io.InputStreamReader;
9 import java.io.OutputStream;
10 import java.io.OutputStreamWriter;
11 import java.net.ServerSocket;
12 import java.net.Socket;
13 import java.net.SocketException;
14
15 import javax.swing.JFrame;
16 import javax.swing.JLabel;
17 import javax.swing.JPanel;
18 import javax.swing.JScrollPane;
19 import javax.swing.JTextArea;
20 import javax.swing.JTextField;
21
22 public class OneToOneS extends JFrame{
23
24     private static final long serialVersionUID = 1L;
25
26     Socket conn;
27     BufferedWriter output;
28     BufferedReader input;
29     JTextArea display;
30     JTextField text;
31     String clientdata="";
```

```
32  
33 public OneToOneS() {  
34     // creating the frame  
35     super("채팅프로그램-서버");  
36     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
37     setSize(800, 500);  
38  
39     // creating the panel at bottom  
40     JPanel panel = new JPanel(); //  
41     //  
42     JLabel label = new JLabel("대화말");  
43     text = new JTextField(60); // 15 characters  
44     //  
45     panel.add(label);  
46     panel.add(text);  
47  
48     // TextArea  
49     display = new JTextArea();  
50     display.setEditable(false);  
51     JScrollPane scrollPane = new JScrollPane(display);  
52  
53  
54     getContentPane().add(BorderLayout.SOUTH, panel);  
55     getContentPane().add(BorderLayout.CENTER, scrollPane);  
56     setVisible(true);  
57
```

```
58    text.addActionListener(new ActionListener() {  
59  
60        @Override  
61        public void actionPerformed(ActionEvent e) {  
62            String serverdata = text.getText();  
63            try {  
64                display.append("\n서버: " + serverdata);  
65                output.write(serverdata + "\r\n");  
66                output.flush();  
67                text.setText("");  
68  
69                if(serverdata.equals("quit")) {  
70                    conn.close();  
71                }  
72  
73            }catch(IOException ex) {  
74                ex.printStackTrace();  
75            }  
76        }  
77    });  
78}  
79}  
80}  
81}
```

스트림에서의 개행문자는 "\r\n"이 개행문자이다.  
따라서, 보내는쪽의 데이터 뒤에 "\r\n"을 반드시 붙여야한다.

```
82 public void runServer() {
83     ServerSocket server;
84     try {
85         server = new ServerSocket(5000);
86         conn = server.accept();
87         InputStream is = conn.getInputStream();
88         InputStreamReader isr = new InputStreamReader(is);
89         input = new BufferedReader(isr); // 클라이언트가 전송한 대화말을 수신
90
91         OutputStream os = conn.getOutputStream();
92         OutputStreamWriter osw = new OutputStreamWriter(os); // 클라이언트에 대화말 전송
93         output = new BufferedWriter(osw);
94
95         while((conn.isBound()) && ((clientdata=input.readLine()) != null)) {
96
97             if (clientdata.equals("quit")) {
98                 display.append("\n클라이언트와의 접속이 중단 되었습니다.");
99                 output.flush();
100                break;
101            }else {
102                display.append("\n클라이언트 메시지: " + clientdata);
103                output.flush();
104            }
105        }
106        conn.close();
107    }catch(SocketException ex2) {
108        System.out.println("Connection reset");
109    }catch(IOException e) {
110        e.printStackTrace();
111    }finally {
112        if (!conn.isClosed())
113            try { conn.close(); } catch (IOException e) { e.printStackTrace(); }
114    }
115 }
116 }
```

```
117④ public static void main(String[] args) {  
118     OneToOneS s = new OneToOneS();  
119     s.runServer();  
120 }  
121  
122 }  
123
```

# OneToOneC.java 클라이언트

The screenshot shows a Java IDE interface with two tabs at the top: 'OneToOneS.java' and 'OneToOneC.java'. The 'OneToOneC.java' tab is active, displaying the code for the client application. The code uses Java's Swing library for the user interface and the Java NIO API for network communication via sockets.

```
1① import java.awt.BorderLayout;
2 import java.awt.event.ActionEvent;
3 import java.awt.event.ActionListener;
4 import java.io.BufferedReader;
5 import java.io.BufferedWriter;
6 import java.io.IOException;
7 import java.io.InputStreamReader;
8 import java.io.OutputStreamWriter;
9 import java.net.InetAddress;
10 import java.net.Socket;
11 import java.net.SocketException;
12
13 import javax.swing.JFrame;
14 import javax.swing.JLabel;
15 import javax.swing.JPanel;
16 import javax.swing.JScrollPane;
17 import javax.swing.JTextArea;
18 import javax.swing.JTextField;
19
20 public class OneToOneC extends JFrame {
21
22     private static final long serialVersionUID = 1L;
23
24     JTextArea display;
25     JTextField text;
26     BufferedWriter output;
27     BufferedReader input;
28     Socket client;
29     String serverdata = "";
30 }
```

```
31@ public OneToOneC() {
32     super("채팅프로그램-클라이언트");
33
34     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
35     setSize(800, 500);
36     // creating the panel at bottom
37     JPanel panel = new JPanel(); //
38     JLabel label = new JLabel("대화말");
39     text = new JTextField(60); // 60 characters
40     //
41     panel.add(label);
42     panel.add(text);
43     // TextArea
44     display = new JTextArea();
45     display.setEditable(false);
46     JScrollPane scrollPane = new JScrollPane(display);
47     //
48     getContentPane().add(BorderLayout.SOUTH, panel);
49     getContentPane().add(BorderLayout.CENTER, scrollPane);
50
51     setVisible(true);
```

```
52    text.addActionListener(new ActionListener() {
53
54        @Override
55        public void actionPerformed(ActionEvent e) {
56            String clientdata = text.getText();
57            try {
58                display.append("\n클라이언트: " + clientdata);
59                output.write(clientdata + "\r\n");
60                output.flush();
61                text.setText("");
62                if(clientdata.equals("quit")) {
63                    client.close();
64                }
65            }catch(IOException ex) {
66                ex.printStackTrace();
67            }
68        }
69    });
70}
71}
72}
```

```
73    public void runclient() {
74        try {
75            client = new Socket(InetAddress.getLocalHost(), 5000);
76            input = new BufferedReader(new InputStreamReader(client.getInputStream()));
77            output = new BufferedWriter(new OutputStreamWriter(client.getOutputStream()));
78            while(client.isBound() && ((serverdata=input.readLine()) != null)) {
79
80                if (serverdata.equals("quit")) {
81                    display.append("\n서버와의 접속이 중단 되었습니다.");
82                    output.flush();
83                    break;
84                }else {
85                    display.append("\n서버 메시지: " + serverdata);
86                    output.flush();
87                }
88            }
89            client.close();
90        }catch(SocketException ex2) {
91            System.out.println("Connection reset");
92        }catch(IOException e) {
93            e.printStackTrace();
94        }finally {
95            if (!client.isClosed())
96                try { client.close(); } catch (IOException e) { e.printStackTrace(); }
97        }
98    }
99
100   public static void main(String[] args) {
101       OneToOneC c = new OneToOneC();
102       c.runclient();
103   }
104 }
```

# Thread

# 멀티태스킹(multi-tasking) 개념

- 멀티태스킹
  - 하나의 응용프로그램이 여러 개의 작업(태스크)을 동시에 처리



다림질하면서 이어폰으로  
전화하는 주부

운전하면서  
화장하는 운전자

제품의 판독과 포장 작업의  
두 기능을 갖춘 기계

# 쓰레드와 멀티스레딩

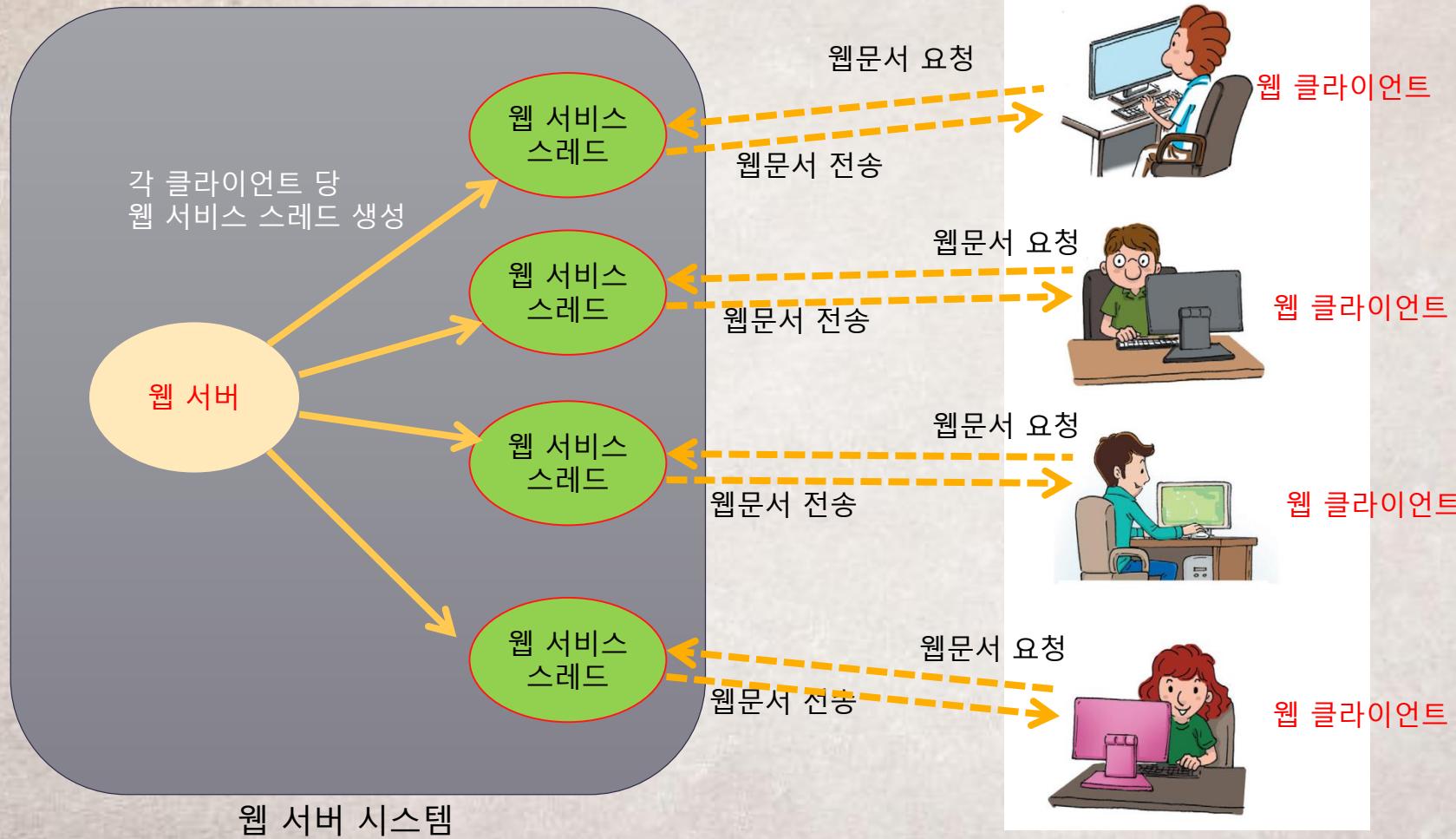
- 쓰레드

- 운영체제에 의해 관리되는 하나의 작업 혹은 태스크
- 다수의 쓰레드를 동시에 실행시키도록 응용프로그램을 작성하는 기법을 **멀티쓰레딩**이라 함.
- 쓰레드 실행코드와 운영체제에서 관리하는 스레드 정보로 이분화
  - 스레드 코드 – 작업을 실행하기 위해 사용자가 작성한 프로그램 코드
  - 스레드 정보 – 스레드 명, 스레드 ID, 스레드 소요시간, 스레드 우선 순위 등 운영체제가 관리하는 정보

- 자바의 멀티태스킹

- 멀티스레딩만 가능
  - 자바에 프로세스 개념은 존재하지 않고, 스레드 개념만 존재
  - 스레드는 실행 단위
  - 스레드는 스케줄링 단위
- 하나의 응용프로그램은 여러 개의 스레드로 구성 가능
  - 스레드 사이의 통신에 따른 오버헤드가 크지 않음

# 웹 서버의 멀티스레딩 사례



# 자바 스레드(Thread)란?

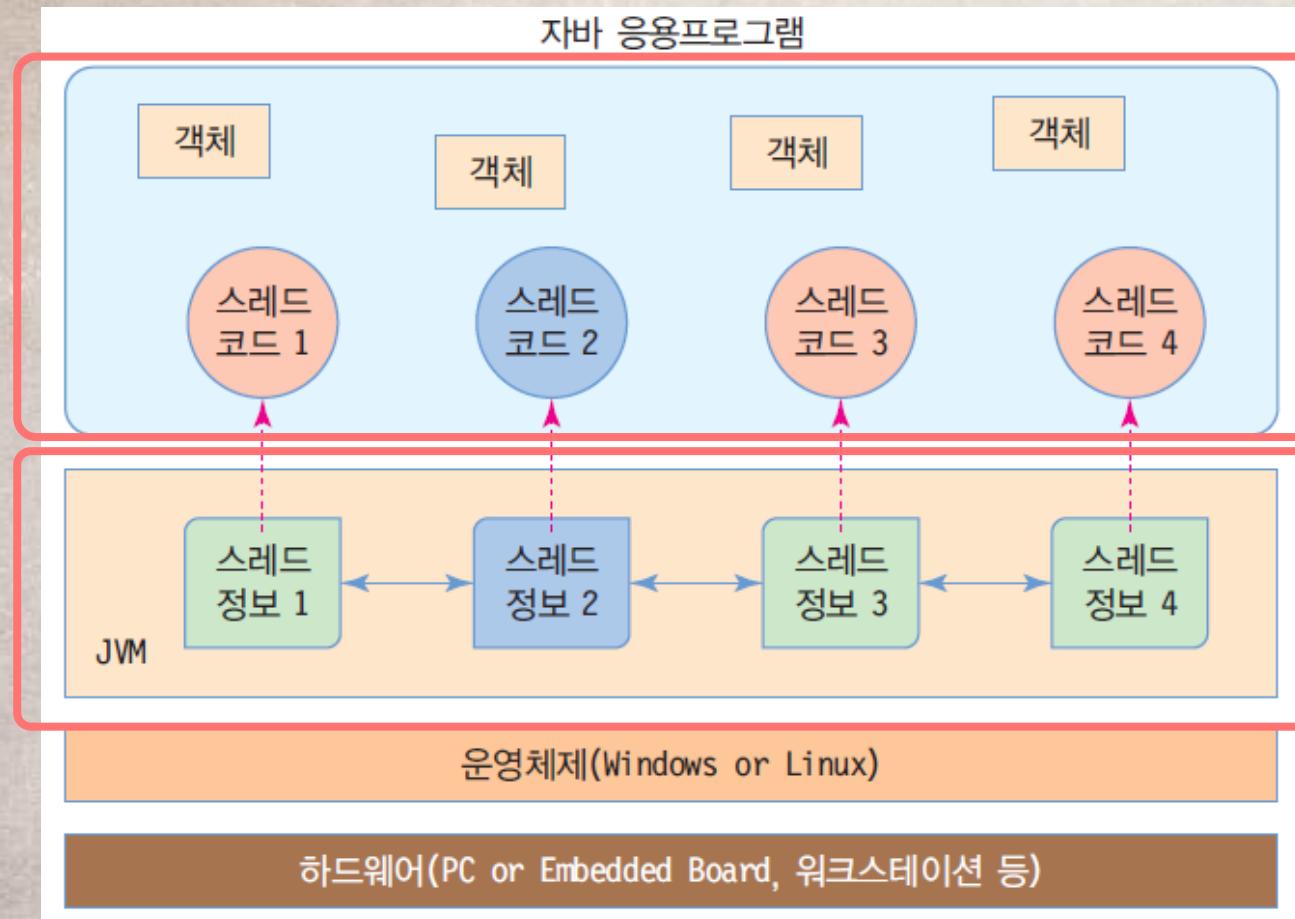
- 자바 스레드
  - 자바 가상 기계(JVM)에 의해 스케줄되는 실행 단위의 코드 블럭
  - 스레드의 생명 주기는 JVM에 의해 관리됨
    - JVM은 스레드 단위로 스케줄링
- JVM과 멀티스레드의 관계
  - 하나의 JVM은 하나의 자바 응용프로그램만 실행
    - 자바 응용프로그램이 시작될 때 JVM이 함께 실행됨
    - 자바 응용프로그램이 종료하면 JVM도 함께 종료함
  - 하나의 응용프로그램은 하나 이상의 스레드로 구성 가능

# JVM과 자바 응용프로그램, 스레드의 관계



두 개의 자바 응용프로그램을 동시에 실행시키고자 하면 **두 개의 JVM을 이용**하고 응용프로그램은 서로 **소켓** 등을 이용하여 통신

# 자바 스레드와 JVM



- 각 스레드의 스레드 코드는 응용프로그램 내에 존재함

JVM이 스레드를 관리함

- 스레드가 몇 개인지?
- 스레드 코드의 위치가 어디인지?
- 스레드의 우선순위는 얼마인지?  
등 ...

현재 하나의 JVM에 의해 4 개의 스레드가 실행 중이며  
그 중 스레드 2가 JVM에 의해 스케줄링되어 실행되고 있음

# 스레드 만들기

- 스레드 실행을 위해 개발자가 하는 작업
  - 스레드 코드 작성
  - JVM에게 스레드를 생성하고 스레드 코드를 실행하도록 요청
- 자바에서 스레드 만드는 2 가지 방법
  - java.lang.Thread 클래스를 이용하는 경우
  - java.lang.Runnable 인터페이스를 이용하는 경우

# Thread 클래스의 메소드

- 생성자
  - Thread()
  - Thread(Runnable target)
  - Thread(String name)
  - Thread(Runnable target, String name)
- 스레드 시작시키기
  - void start()
- 스레드 코드
  - void run()
- 스레드 잠자기
  - static void sleep(long mills)
- 다른 스레드 죽이기
  - void interrupt()
- 다른 스레드에게 양보
  - static void yield()
  - 현재 스레드의 실행을 중단하고 다른 스레드가 실행될 수 있도록 양보한다.
- 다른 스레드가 죽을 때까지 기다리기
  - void join()
- 현재 스레드 객체 알아내기
  - static Thread currentThread()
- 스레드 ID 알아내기
  - long getId()
- 스레드 이름 알아내기
  - String getName()
- 스레드 우선순위값 알아내기
  - int getPriority()
- 스레드의 상태 알아내기
  - Thread.State getState()
    - Blocked, Waiting, Runnable, ..

# Thread 클래스를 이용한 스레드 생성

- 스레드 클래스 작성
    - Thread 클래스 상속. 새 클래스 작성
  - 스레드 코드 작성
    - run() 메소드 오버라이딩
      - run() 메소드를 스레드 코드라고 부름
      - run() 메소드에서 스레드 실행 시작
  - 스레드 객체 생성
  - 스레드 시작
    - start() 메소드 호출
      - 스레드로 작동 시작
      - JVM에 의해 스케줄되기 시작함
- 
- ```
class TimerThread extends Thread {  
    .....  
}
```
- ```
TimerThread th = new TimerThread();
```
- ```
th.start();
```

## \* Thread를 상속받아 1초 단위로 초 시간을 출력하는 TimerThread 스레드 작성

스레드 클래스 정의

스레드 코드 작성

1초에 한 번씩 n을 증가 시켜 콘솔에 출력한다.

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

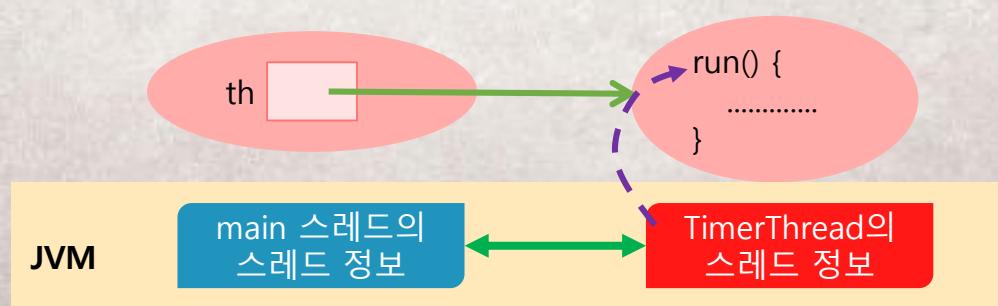
스레드 객체 생성  
스레드 시작

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

0  
1  
2  
3  
4

main() 스레드

TimerThread 스레드



# 예제 exam01 : Thread를 상속받아 1초 단위의 타이머 만들기

TimerThread.java TimerThreadEx.java

```
1 public class TimerThread extends Thread {  
2  
3  
4     @Override  
5     public void run() {  
6         int n = 0;  
7  
8         while(true) {  
9             System.out.println(n);  
10            n++;  
11            try {  
12                Thread.sleep(1000);  
13            }catch(InterruptedException e) {  
14                e.printStackTrace();  
15                return;  
16            }  
17        }  
18    }  
19}  
20}  
21}
```

TimerThread.java \*TimerThreadEx.java

```
1 public class TimerThreadEx {  
2  
3  
4     public TimerThreadEx() {  
5         TimerThread th = new TimerThread();  
6         th.start();  
7     }  
8  
9     public static void main(String[] args) {  
10        // TODO Auto-generated method stub  
11        new TimerThreadEx();  
12    }  
13  
14}  
15  
16|
```

Problems Javadoc Declaration Console

<terminated> RunnableTimerEx [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2015. 5. 14. 오후 8:27:32)

```
0  
1  
2  
3  
4  
5
```

# 스레드 주의 사항

- run() 메소드가 종료하면 스레드는 종료한다.
  - 스레드가 계속 존재하게 하려면 run() 메소드 내에 무한 루프가 실행되어야 한다.
- 한번 종료한 스레드는 다시 시작시킬 수 없다.
  - 스레드 객체를 생성하여 다시 스레드로 등록하여야 한다.
- 한 스레드에서 다른 스레드를 강제 종료할 수 있다.
  - 뒤에서 다룸

# Runnable 인터페이스로 스레드 만들기

- 스레드 클래스 작성
  - Runnable 인터페이스 구현하는 새 클래스 작성
- 스레드 코드 작성
  - run() 메소드 오버라이딩
    - run() 메소드를 스레드 코드라고 부름
    - run() 메소드에서 스레드 실행 시작
- 스레드 객체 생성
- 스레드 시작
  - start() 메소드 호출

```
class TimerRunnable implements Runnable {
```

```
}
```

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

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

## \*Runnable 인터페이스를 상속받아 1초 단위로 초 시간을 출력하는 스레드 작성

스레드 클래스 정의

스레드 코드 작성

1초에 한 번씩 n을 증가  
시켜 콘솔에 출력한다.

```
class TimerRunnable implements Runnable {  
    int n = 0;  
    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;}  
        }  
    }  
}
```

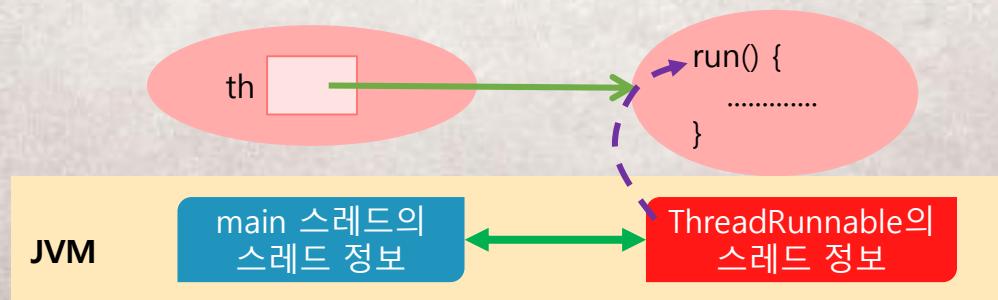
스레드 객체 생성  
스레드 시작

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

0  
1  
2  
3  
4

main() 스레드

ThreadRunnable 스레드



## 예제 exam02 : Runnable인터페이스를 구현하여 1초 단위 타이머 만들기

```
1  public class RunnableTimer implements Runnable {  
2  
3  
4    @Override  
5    public void run() {  
6      int n = 0;  
7  
8      while(true) {  
9        System.out.println(n);  
10       n++;  
11       try {  
12         Thread.sleep(1000);  
13       }catch(InterruptedException e) {  
14         e.printStackTrace();  
15         return;  
16       }  
17     }  
18   }  
19 }  
20 }  
21 }
```

```
1  public class RunnableTimerEx {  
2  
3  
4    public RunnableTimerEx() {  
5      RunnableTimer runnable = new RunnableTimer();  
6      Thread th = new Thread(runnable);  
7      th.start();  
8    }  
9  
10   public static void main(String[] args) {  
11     new RunnableTimerEx();  
12   }  
13 }  
14 }
```

Problems @ Javadoc Declaration Console

<terminated> RunnableTimerEx [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2015. 5. 14. 오후 8:27:32)

```
0  
1  
2  
3  
4  
5
```

# 스레드 정보

| 필드          | 타입     | 내용                                                                                                                                       |
|-------------|--------|------------------------------------------------------------------------------------------------------------------------------------------|
| 스레드 이름      | 스트링    | 스레드의 이름으로서 사용자가 지정                                                                                                                       |
| 스레드 ID      | 정수     | 스레드 고유의 식별자 번호                                                                                                                           |
| 스레드 상태      | 정수     | NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCK, TERMINATED 등 6개 상태 중 하나<br>- TIMED_WAITING은 시간 인자가 들어간 메소드(Thread.sleep())가 호출될 때, 진입되는 상태 |
| 스레드 우선순위    | 정수     | 스레드 스케줄링 시 사용되는 우선순위 값으로서 1~10 사이의 값이며 10이 최상위 우선순위                                                                                      |
| 스레드 그룹      | 정수     | 여러 개의 자바 스레드가 하나의 그룹을 형성할 수 있으며 이 경우 스레드가 속한 그룹                                                                                          |
| 스레드 레지스터 스택 | 메모리 블록 | 스레드가 실행되는 동안 레지스터들의 값                                                                                                                    |

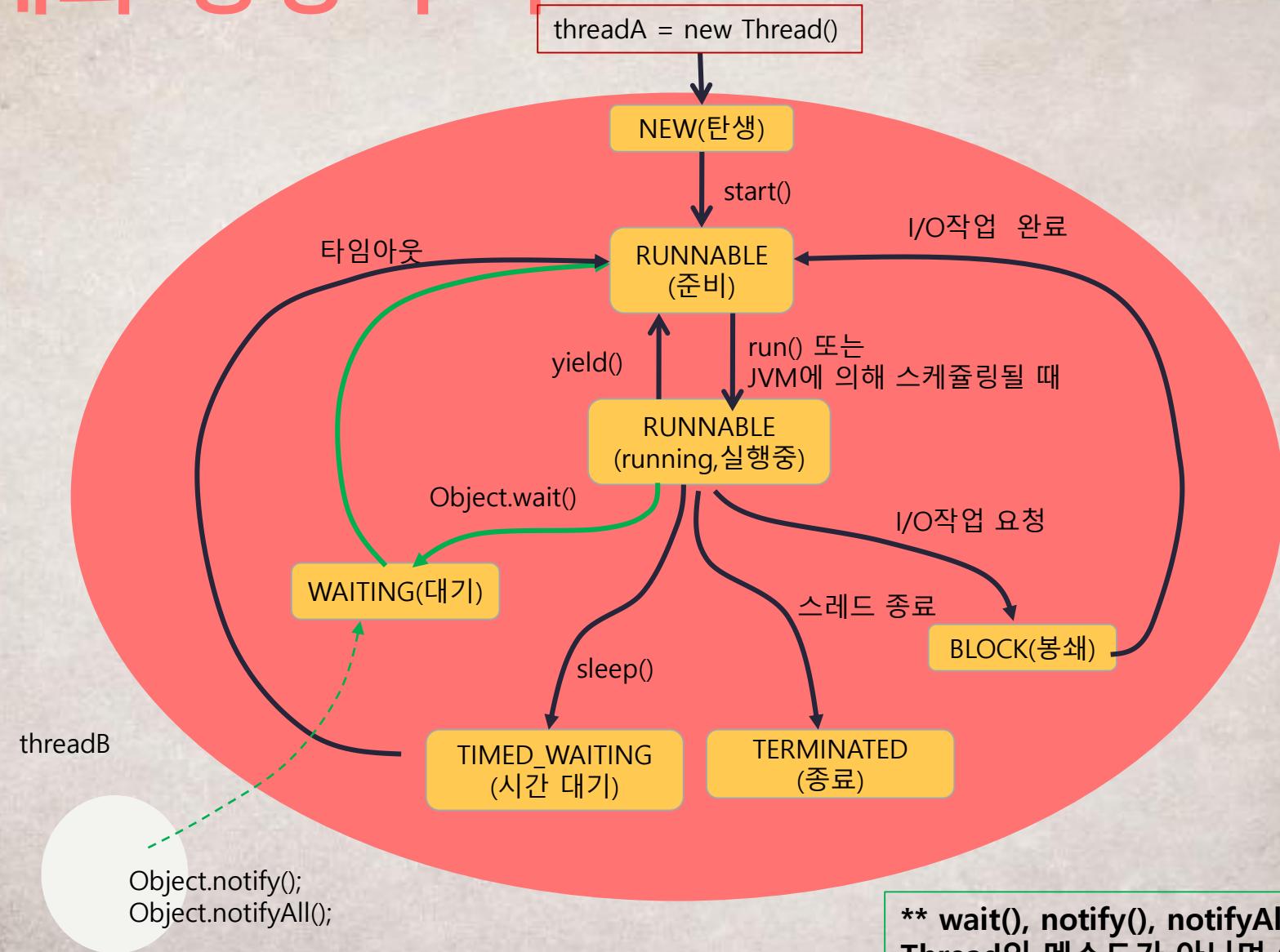
# 스레드 상태

- 스레드 상태 6 가지
  - NEW
    - 스레드가 생성되었지만 스레드가 아직 실행할 준비가 되지 않았음
  - RUNNABLE
    - 스레드가 JVM에 의해 실행되고 있거나 실행 준비되어 스케줄링을 기다리는 상태
  - WAITING
    - 어떤 Object 객체에서 다른 스레드가 notify(), notifyAll()을 불러주기를 기다리고 있는 상태.
    - 스레드 동기화를 위해 사용
  - TIMED\_WAITING
    - 스레드가 sleep(n) 호출로 인해 n 밀리초 동안 잠을 자고 있는 상태
  - BLOCK
    - 스레드가 I/O 작업을 요청하면 JVM이 자동으로 이 스레드를 BLOCK 상태로 만든다.
  - TERMINATED
    - 스레드가 종료한 상태
- 스레드 상태는 JVM에 의해 기록 관리됨

# 스레드 상태와 생명 주기

## 스레드 상태 6 가지

- NEW
- RUNNABLE
- WAITING
- TIMED\_WAITING
- BLOCK
- TERMINATED



\*\* wait(), notify(), notifyAll()은 Thread의 메소드가 아니며 Object의 메소드임

# 스레드 우선 순위와 스케줄링

- 스레드의 우선 순위
  - 최대값 = 10(MAX\_PRIORITY)
  - 최소값 = 1(MIN\_PRIORITY)
  - 보통값 = 5(NORMAL\_PRIORITY)
- 스레드 우선 순위는 응용프로그램에서 변경 가능
  - void setPriority(int priority)
  - int getPriority()
- main() 스레드의 우선 순위 값은 초기에 보통값 5로 지정.
- 스레드는 부모 스레드와 동일한 우선순위 값을 가지고 탄생
- JVM의 스케줄링 정책
  - 철저한 우선 순위 기반
    - 가장 높은 우선 순위의 스레드가 우선적으로 스케줄링
    - 동일한 우선 순위의 스레드는 돌아가면서 스케줄링(라운드 로빈).

# main()은 자바의 main 스레드

- main() 메소드
  - JVM에 의해 자동으로 스레드화
  - 자바 스레드 : main 스레드
  - main() 함수가 스레드 코드로 사용

The screenshot shows an IDE interface with two tabs at the top: 'TimerRunnable.java' and 'RunnableTimerEx.java'. The active tab is 'ThreadMainEx.java'. The code in 'ThreadMainEx.java' is as follows:

```
1 public class ThreadMainEx {  
2     public static void main(String[] args) {  
3         long id = Thread.currentThread().getId();  
4         String name = Thread.currentThread().getName();  
5         int priority = Thread.currentThread().getPriority();  
6         Thread.State state = Thread.currentThread().getState();  
7  
8         System.out.println("현재 스레드 이름 = " + name);  
9         System.out.println("현재 스레드 ID = " + id);  
10        System.out.println("현재 스레드 우선순위 값 = " + priority);  
11        System.out.println("현재 스레드 상태 = " + state);  
12    }  
13 }  
14 }
```

Below the code editor is a 'Console' tab. The console output shows the results of the program's execution:

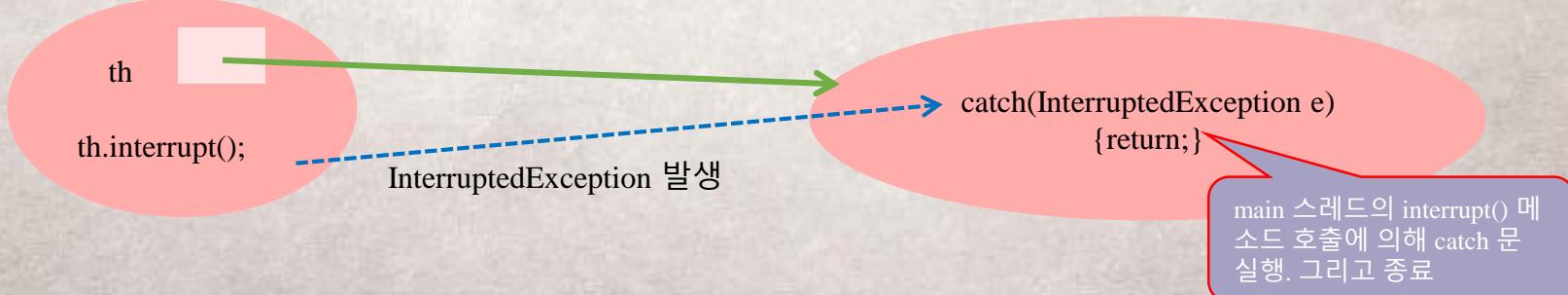
```
<terminated> ThreadMainEx [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2015. 5. 14. 오후 8:32:40)  
현재 스레드 이름 = main  
현재 스레드 ID = 1  
현재 스레드 우선순위 값 = 5  
현재 스레드 상태 = RUNNABLE
```

# 스레드 종료와 타 스레드 강제 종료

- 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  
}
```

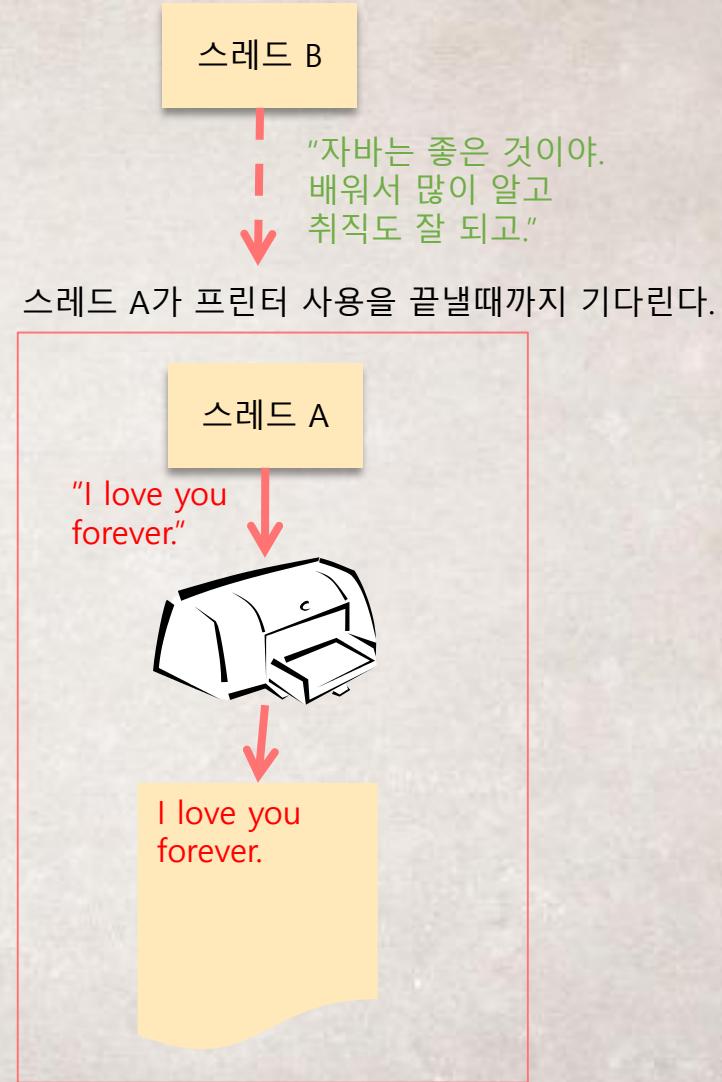
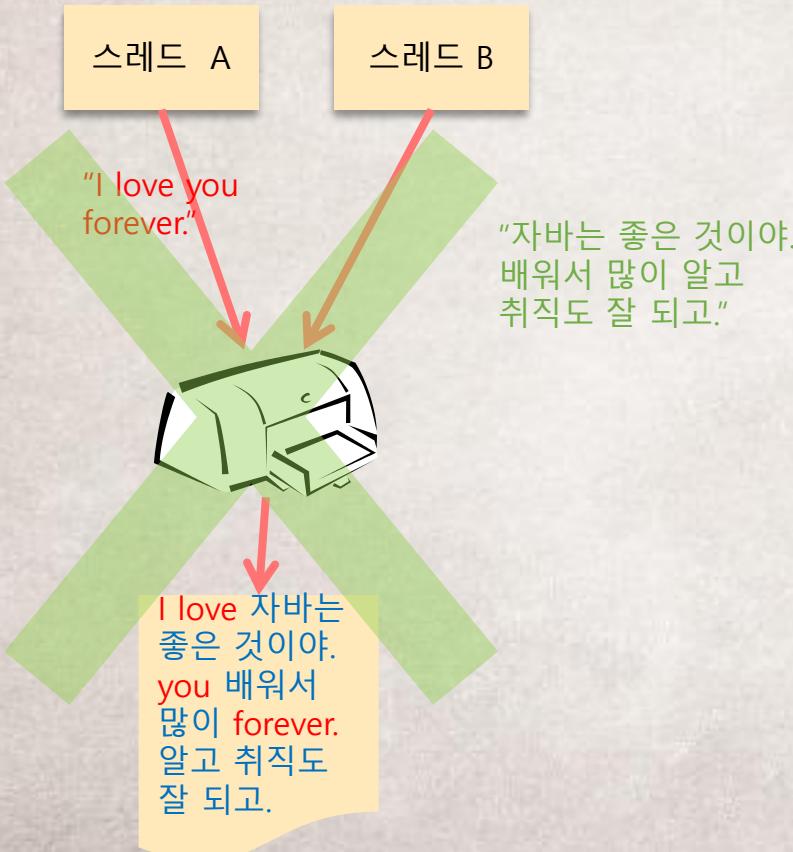
main() 스레드



# 스레드 동기화(Thread Synchronization)

- 멀티스레드 프로그램 작성시 주의점
  - 다수의 스레드가 공유 데이터에 동시에 접근하는 경우
    - 공유 데이터의 값에 예상치 못한 결과 발생 가능
- 스레드 동기화
  - 멀티스레드의 공유 데이터의 동시 접근 문제 해결책
    - 공유데이터를 접근하고자 하는 모든 스레드의 한 줄 세우기
    - 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 공유 데이터에 접근하지 못하도록 함

# 두 스레드가 프린터에 동시 쓰기 수행 시



# synchronized 키워드

- synchronized 키워드
  - 한 스레드만이 독점적으로 실행되어야 하는 부분(동기화 코드)을 표시하는 키워드
    - 임계 영역(critical section) 표기 키워드
- synchronized 키워드 사용 가능한 부분
  - 메소드 전체 혹은 코드 블럭
- synchronized 부분이 실행될 때,
  - 실행 스레드는 모니터 소유
    - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
  - 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드는 대기

```
synchronized void add() {  
    int n = getCurrentSum();  
    n+=10;  
    setCurrentSum(n);  
}
```

synchronized 메소드

```
void execute() {  
    // 다른 코드들  
    //  
    synchronized(this) {  
        int n = getCurrentSum();  
        n+=10;  
        setCurrentSum(n);  
    }  
    //  
    // 다른 코드들  
}
```

synchronized 코드블럭

# exam03. synchronized 사용 예 : 점수판 코딩

```
SyncObject.java WorkerThread.java SynchronizedEx.java
1 public class SynchronizedEx {
2
3     public static void main(String[] args) {
4         SyncObject obj = new SyncObject();
5
6         Thread th1 = new WorkerThread("kitae",obj);
7         Thread th2 = new WorkerThread("hyosoo",obj);
8         th1.start();
9         th2.start();
10    }
11
12 }
13
14 }
```

```
SyncObject.java WorkerThread.java SynchronizedEx.java
1 class SyncObject {
2     int sum = 0;
3
4     synchronized void add(){
5         int n = sum;
6
7         Thread.currentThread().yield();
8
9         n += 10;
10        sum = n;
11        System.out.println(Thread.currentThread().getName() + ":" + sum);
12    }
13
14 }
15
16 }
```

# 09.3 synchronized 사용 예 : 집계판 사례를 코딩

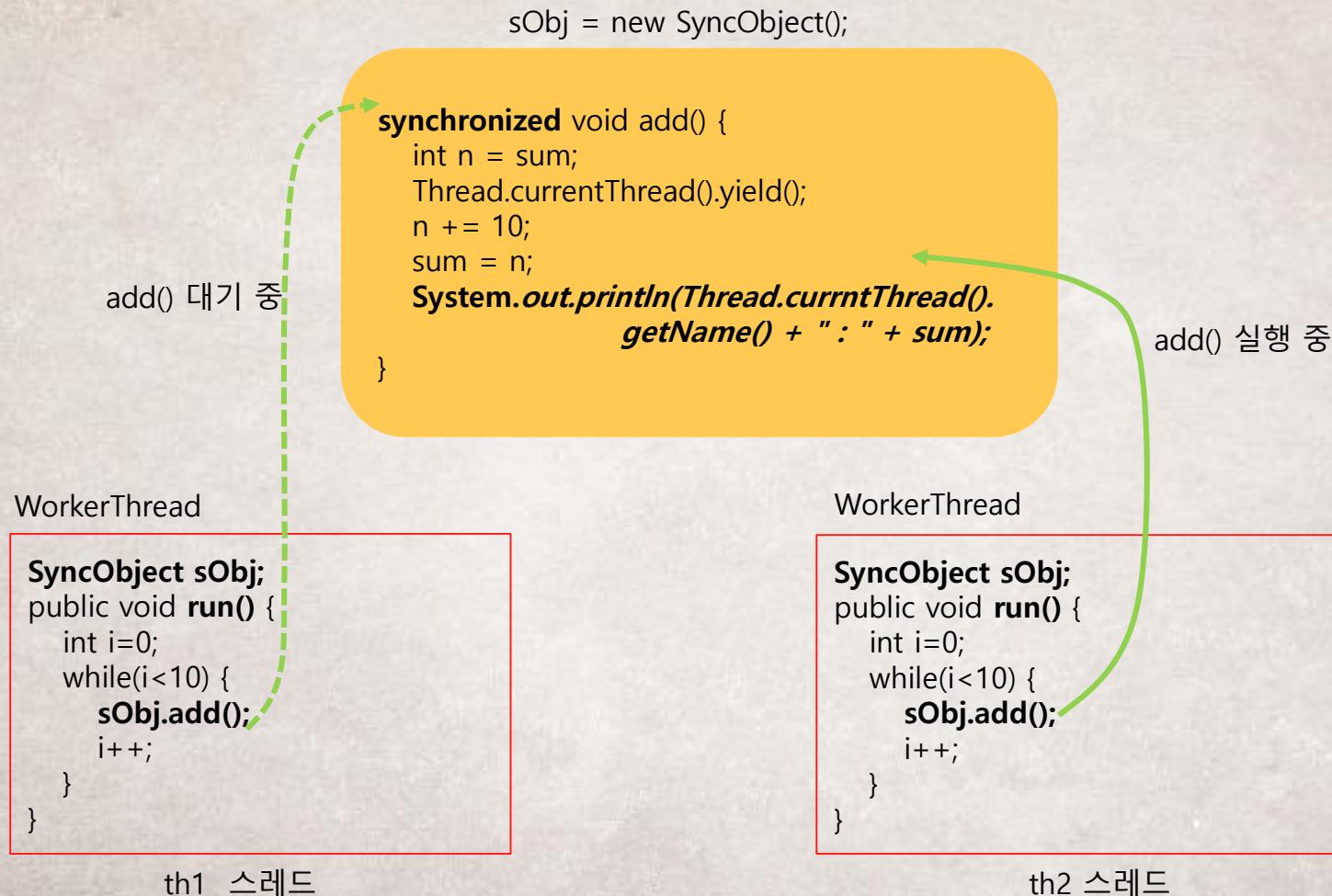
The screenshot shows an IDE interface with two tabs open: SyncObject.java and WorkerThread.java. The WorkerThread.java tab is active, displaying the following code:

```
1 public class WorkerThread extends Thread {  
2     SyncObject sObj;  
3     WorkerThread(String name, SyncObject sObj){  
4         super(name);  
5         this.sObj = sObj;  
6     }  
7     @Override  
8     public void run() {  
9         int i = 0;  
10        while(i < 10){  
11            sObj.add();  
12            i++;  
13        }  
14    }  
15 }
```

Below the code editor is a console window showing the output of the application. The output consists of two sets of interleaved numbers, representing the concurrent execution of two threads. The first set of numbers (kitae) starts at 10 and goes up to 100. The second set (hyosoo) starts at 110 and goes up to 200. Both sets are printed in pairs, indicating that each thread is printing every other number in the sequence.

```
kitae:10  
kitae:20  
kitae:30  
kitae:40  
kitae:50  
kitae:60  
kitae:70  
kitae:80  
kitae:90  
kitae:100  
hyosoo:110  
hyosoo:120  
hyosoo:130  
hyosoo:140  
hyosoo:150  
hyosoo:160  
hyosoo:170  
hyosoo:180  
hyosoo:190  
hyosoo:200
```

# SyncObject 객체에 대한 스레드의 동시 접근



## 09.4 점수판 예에서 synchronized 사용하지 않을 경우

The screenshot shows an IDE interface with three tabs at the top: SyncObject.java, WorkerThread.java, and SynchronizedEx.java. The SyncObject.java tab contains the following code:

```
1 class SyncObject {  
2     int sum = 0;  
3  
4     void add(){  
5         int n = sum;  
6  
7         Thread.currentThread();  
8         n += 10;  
9         sum = n;  
10        System.out.println(  
11    }  
12 }  
13 }  
14 }  
15 }  
16 }
```

The SynchronizedEx.java tab shows the output of the application in the Console tab:

```
<terminated> SynchronizedEx [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2015. 5. 14. 오후 8:56:59)  
hyosoo:10  
kitae:10  
hyosoo:20  
kitae:30  
hyosoo:40  
kitae:50  
hyosoo:60  
kitae:70  
hyosoo:80  
kitae:90  
hyosoo:100  
kitae:110  
hyosoo:120  
kitae:130  
hyosoo:140  
kitae:150  
hyosoo:160  
kitae:170  
hyosoo:180  
kitae:190
```

A red callout points to the first two lines of the output with the text "add() 충돌". A red arrow points from the end of the "kitae:190" line to a explanatory text block.

kitae와 hyosoo가 각각 10번씩 add()를 호출하였지만 동기화가 이루어지지 않아 공유 변수 sum에 대한 접근에 충돌이 있었고, 수를 많이 잃어버리게 되어 누적 점수가 190 밖에 되지 못함

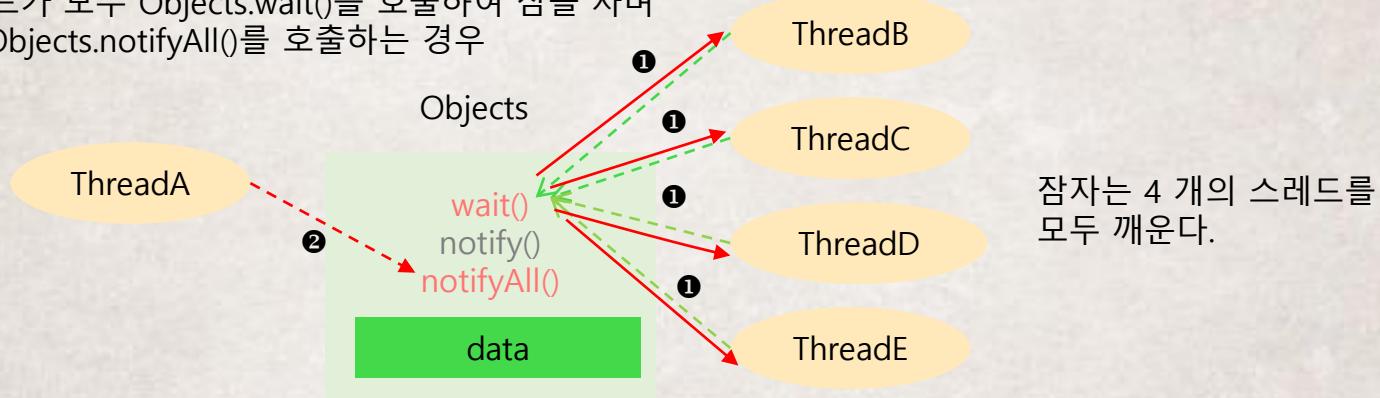
# 동기화 메소드: wait(), notify(), notifyAll()

- 동기화 객체
  - 두 개 이상의 스레드 사이에 동기화 작업에 사용되는 객체
- 동기화 메소드
  - synchronized 블럭 내에서만 사용되어야 함
  - wait()
    - 다른 스레드가 notify()를 불러줄 때까지 기다린다.
  - notify()
    - wait() 호출로 인해 대기중인 스레드를 깨우고 RUNNABLE 상태로 한다.
    - 2개 이상의 스레드가 대기중이라도 오직 한 개의 스레드만 깨워 RUNNABLE 상태로 한다.
  - notifyAll()
    - wait() 호출로 인해 대기중인 모든 스레드를 깨우고 이들을 모두 RUNNABLE 상태로 한다.
- 동기화 메소드는 Object 클래스의 메소드이다.
  - 모든 객체가 동기화 객체가 될 수 있다.
  - Thread 객체도 동기화 객체로 사용될 수 있다.

하나의 Thread가 Objects.wait()를 호출하여 잠을 자는 경우



4 개의 스레드가 모두 Objects.wait()를 호출하여 잠을 자며  
ThreadA는 Objects.notifyAll()를 호출하는 경우



4 개의 스레드가 모두 Objects.wait()를 호출하여 잠을 자며  
ThreadA는 Objects.notify()를 호출하는 경우

