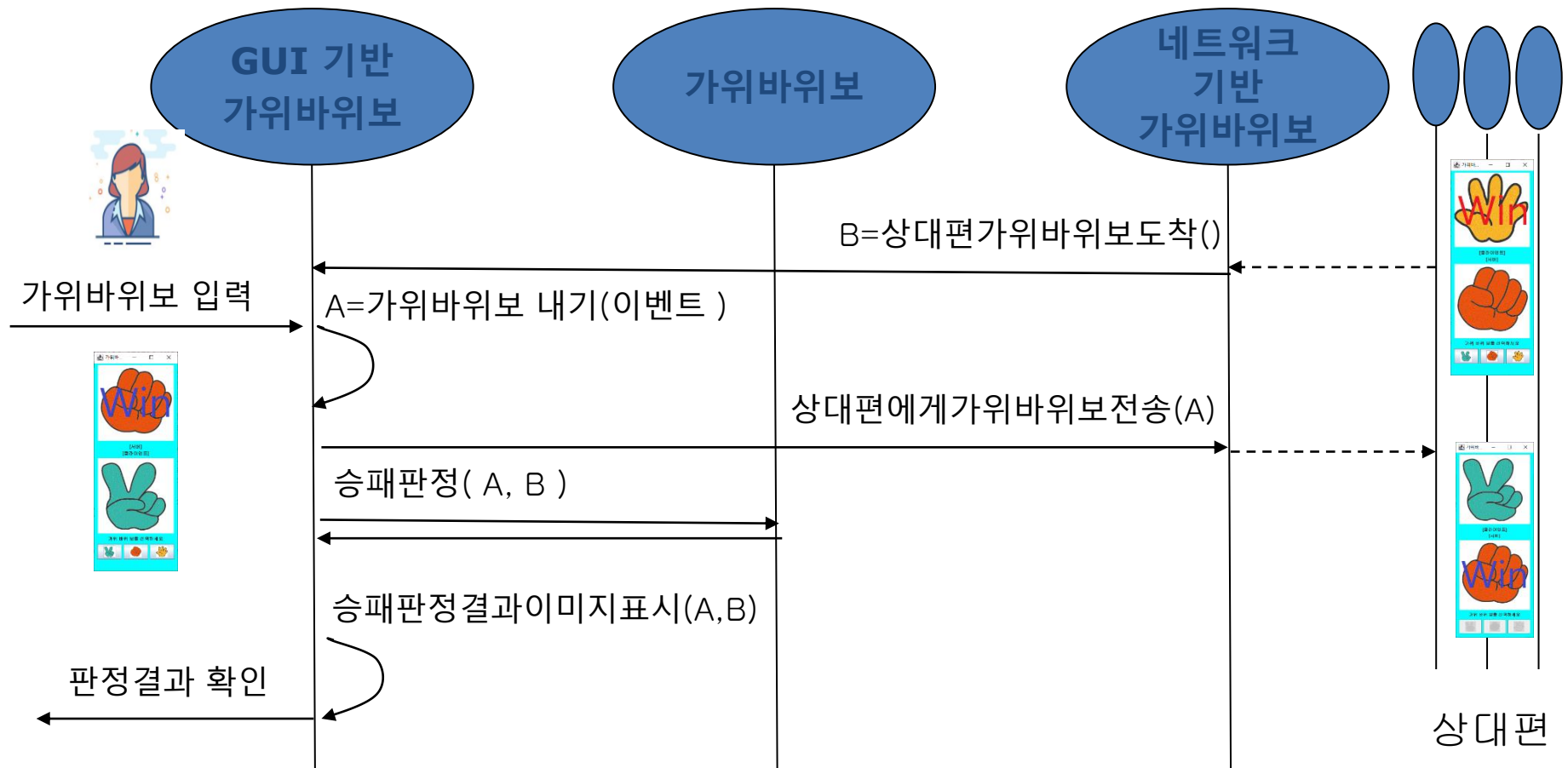


생각하기

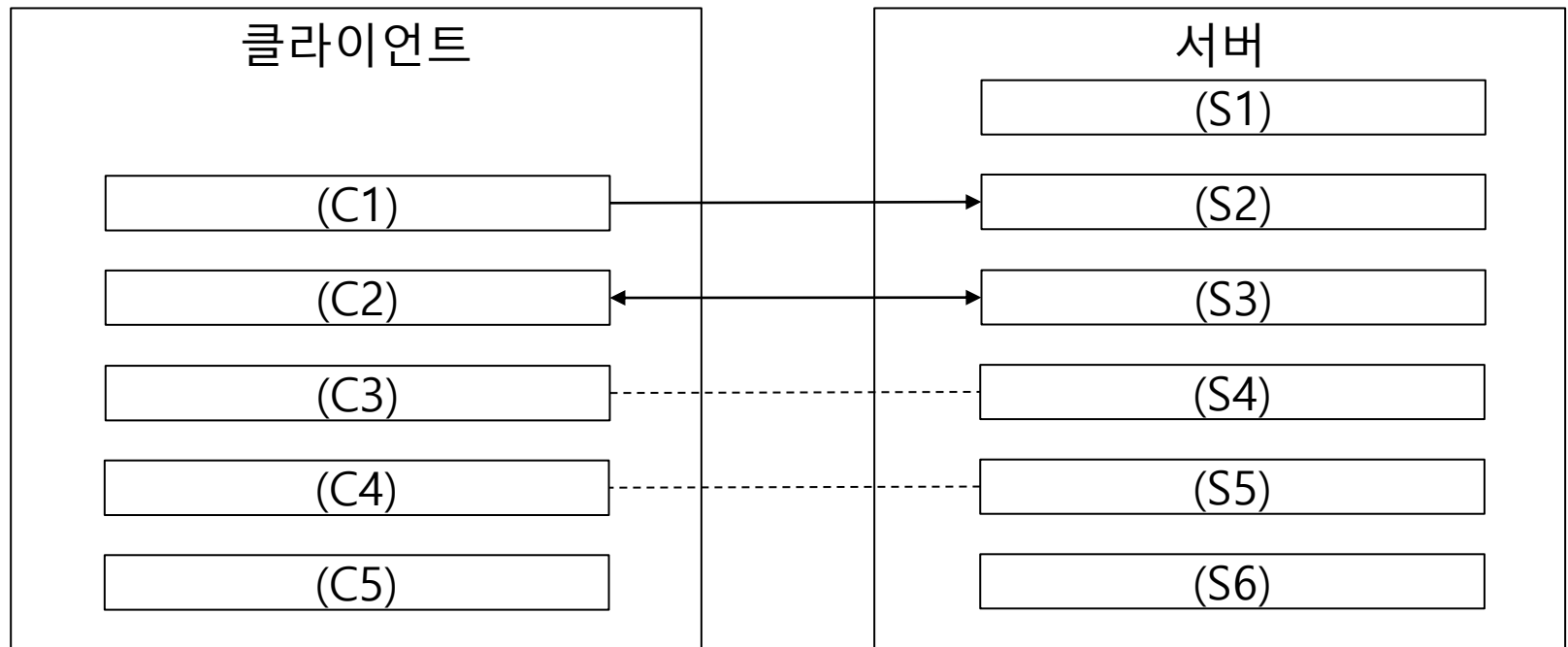
■ 가위바위보 네트워크 프로그램 시퀀스 다이어그램



생각하기

■ 네트워크 채팅 프로그램

- 네트워크의 실행 순서를 고려하여 빈칸을 채우고 점선을 실선 화살표로 표기하시오



생각하기

■ 네트워크 채팅 프로그램

- 빈칸을 채울 때 참고하세요
 - 상대방에게 데이터 보내기
 - 상대방에서 보낸 데이터가 도착할 때까지 대기
 - 클라이언트에서 서버로 연결 요청
 - 클라이언트에서 서버로 연결 요청시 연결 활성화
 - 네트워크 입출력 초기화
 - 서버 소켓 연결
 - 네트워크 종료

생각하기

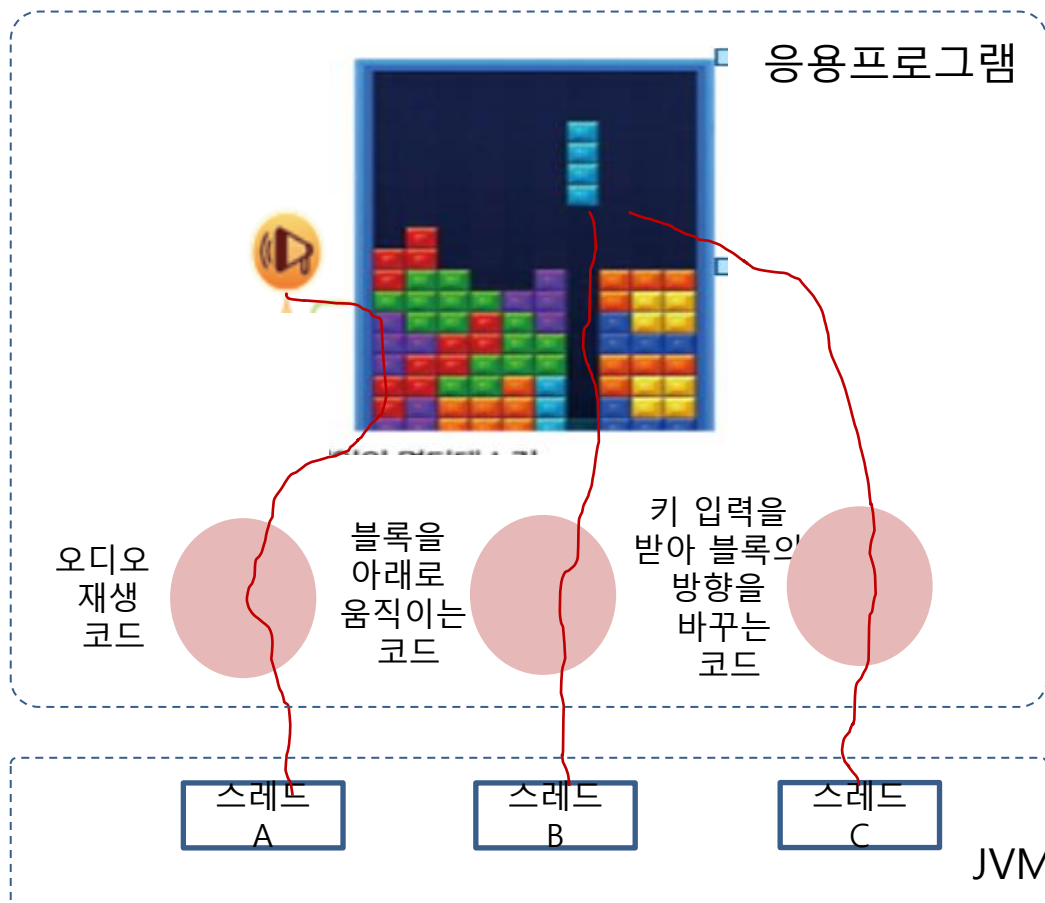
■ 멀티 스레드 프로그램

- 채팅 프로그램에서 대화를 아무 때나 주고받는다고 가정
 - 사용자가 데이터를 입력 및 전송할 때까지 프로그램은 계속 대기
 - 상대방에서 보낸 데이터가 도착할 때까지 프로그램은 계속 대기
- 네트워크 전송 및 수신 of 실행 순서는 어떻게 관리하면 좋을까?
 - 사용자와 상대방 중에서 누가 먼저 데이터를 전송할까?
 - 사용자가 일방적으로 여러 번 데이터를 전송하는 경우 어떻게?
 - 상대방이 일방적으로 여러 번 데이터를 전송하는 경우 어떻게?

생각하기

■ 멀티 스레드 프로그램

- 테트리스 프로그램을 참고하세요



* JVM은 3 개의 스레드 중 하나를 선택하여 실행시킨다. 예를 들어 스레드 A를 선택하면 스레드의 A의 코드(사용자가 작성한 코드)를 호출한다. 스레드 A를 일시 중단하고, JVM이 스레드 B를 실행시켜려면 다시 스레드 B의 코드(사용자가 작성한 코드)를 호출한다.



TCP/IP 소개

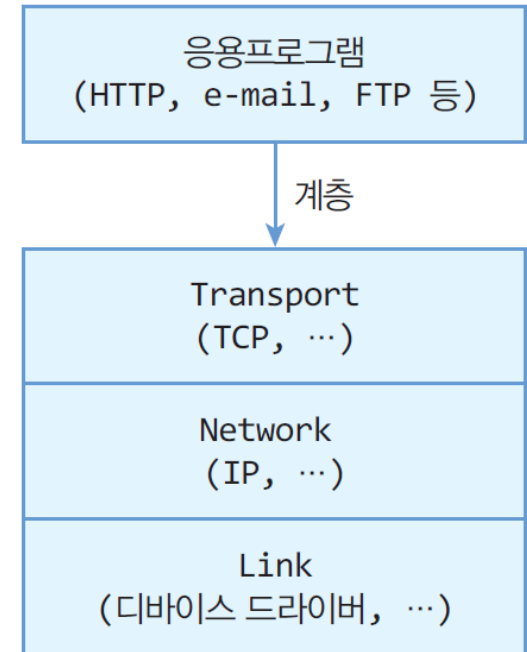
7

□ TCP/IP 프로토콜

- TCP는 Transmission Control Protocol
- 두 시스템 간에 신뢰성 있는 데이터의 전송을 관장하는 프로토콜
- TCP에서 동작하는 응용프로그램 사례
 - e-mail, FTP, 웹(HTTP) 등

□ IP

- Internet Protocol
- 패킷 교환 네트워크에서 송신 호스트와 수신 호스트가 데이터를 주고 받는 것을 관장하는 프로토콜
- TCP보다 하위 레벨 프로토콜



IP 주소

8

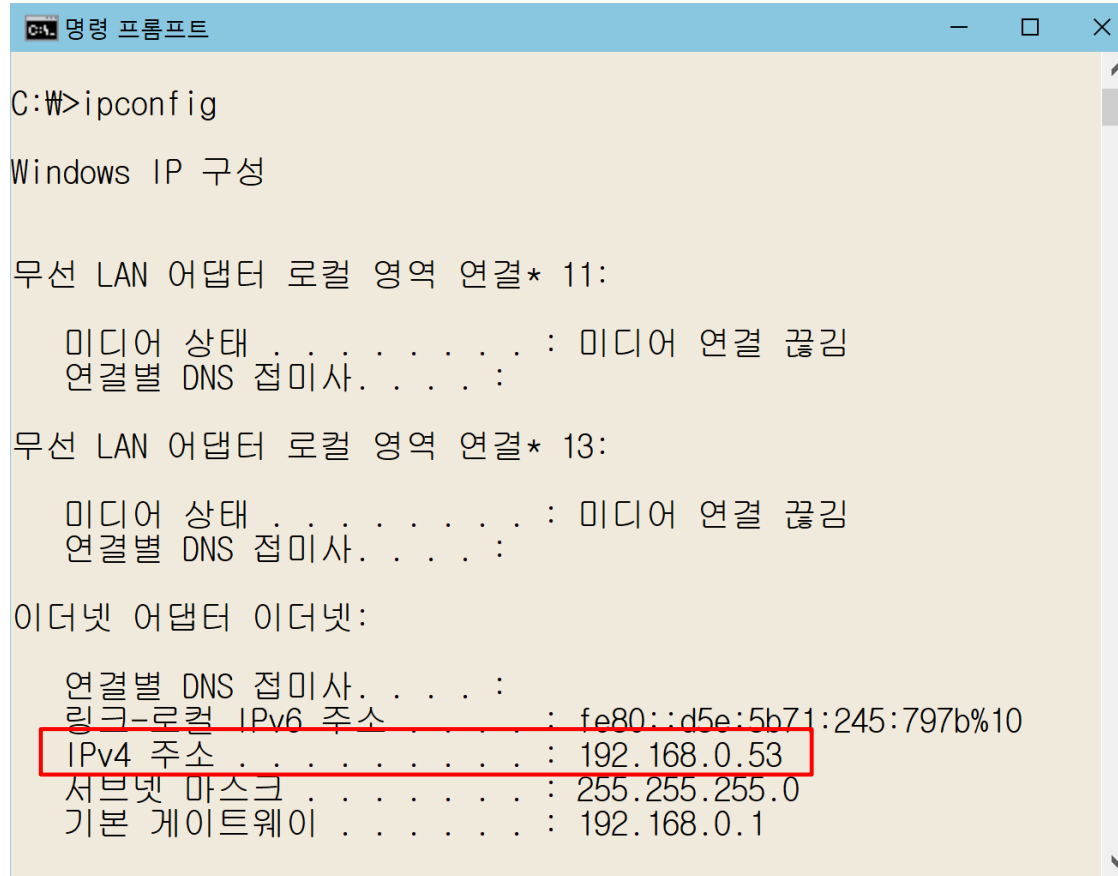
□ IP 주소

- ▣ 네트워크 상에서 유일하게 식별될 수 있는 컴퓨터 주소
 - 숫자로 구성된 주소
 - 4개의 숫자가 '.'으로 연결
 - 예) 192.156.11.15
- ▣ 숫자로 된 주소는 기억하기 어려우므로 www.naver.com과 같은 문자열로 구성된 도메인 이름으로 바꿔 사용
 - DNS(Domain Name System)
 - 문자열로 구성된 도메인 이름을 숫자로 구성된 IP 주소로 자동 변환
- ▣ 현재는 32비트의 IP 버전 4(IPv4)가 사용되고 있음
 - IP 주소 고갈로 인해 128비트의 IP 버전 6(IPv6)이 점점 사용되는 추세

내 컴퓨터의 IP 주소 확인하기

9

- 내 컴퓨터의 윈도우에서 명령창을 열어 ipconfig 명령 수행



```
C:\>ipconfig

Windows IP 구성

무선 LAN 어댑터 로컬 영역 연결* 11:

    미디어 상태 . . . . . : 미디어 연결 끊김
    연결별 DNS 접미사 . . . . :

무선 LAN 어댑터 로컬 영역 연결* 13:

    미디어 상태 . . . . . : 미디어 연결 끊김
    연결별 DNS 접미사 . . . . :

이더넷 어댑터 이더넷:

    연결별 DNS 접미사 . . . . :
    링크-로컬 IPv6 주소 . . . . : fe80::d5e:5b71:245:797b%10
    IPv4 주소 . . . . . : 192.168.0.53
    서브넷 마스크 . . . . . : 255.255.255.0
    기본 게이트웨이 . . . . . : 192.168.0.1
```

□ 포트

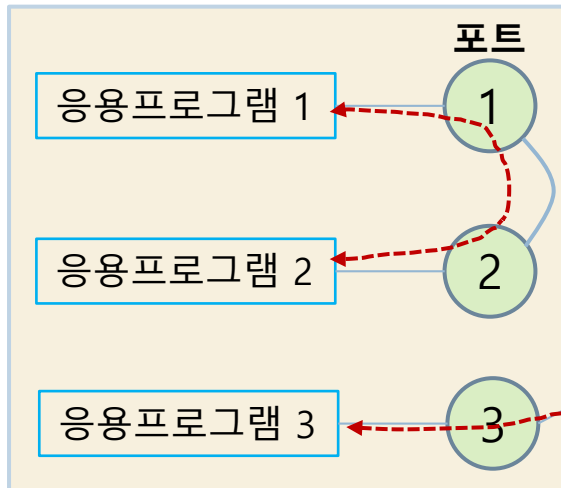
- 통신하는 프로그램 간에 가상의 연결단 포트 생성
 - IP 주소는 네트워크 상의 컴퓨터 또는 시스템을 식별하는 주소
 - 포트 번호를 이용하여 통신할 응용프로그램 식별
- 모든 응용프로그램은 하나 이상의 포트 생성 가능
 - 포트를 이용하여 상대방 응용프로그램과 데이터 교환
- 잘 알려진 포트(well-known ports)
 - 시스템이 사용하는 포트 번호
 - 잘 알려진 응용프로그램에서 사용하는 포트 번호
 - 0부터 1023 사이의 포트 번호
 - ex) SSH 23, HTTP 80, FTP 21
 - 잘 알려진 포트 번호는 개발자가 사용하지 않는 것이 좋음
 - 충돌 가능성 있음



포트를 이용한 통신

11

컴퓨터1(IP: 203.1.1.110)



컴퓨터1(IP: 113.67.23.120)

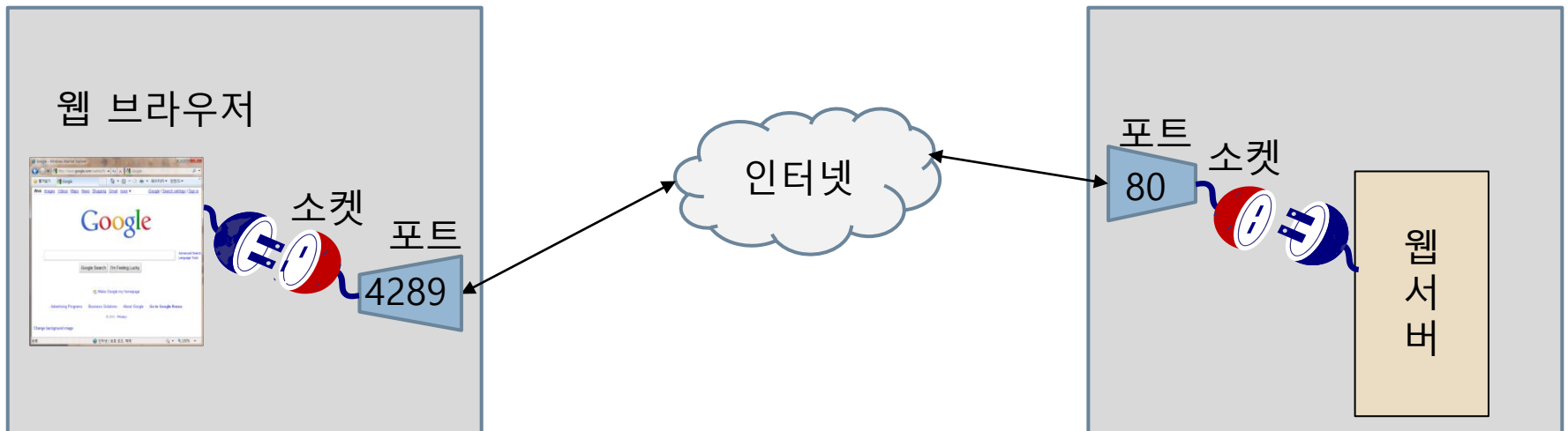


소켓 프로그래밍

12

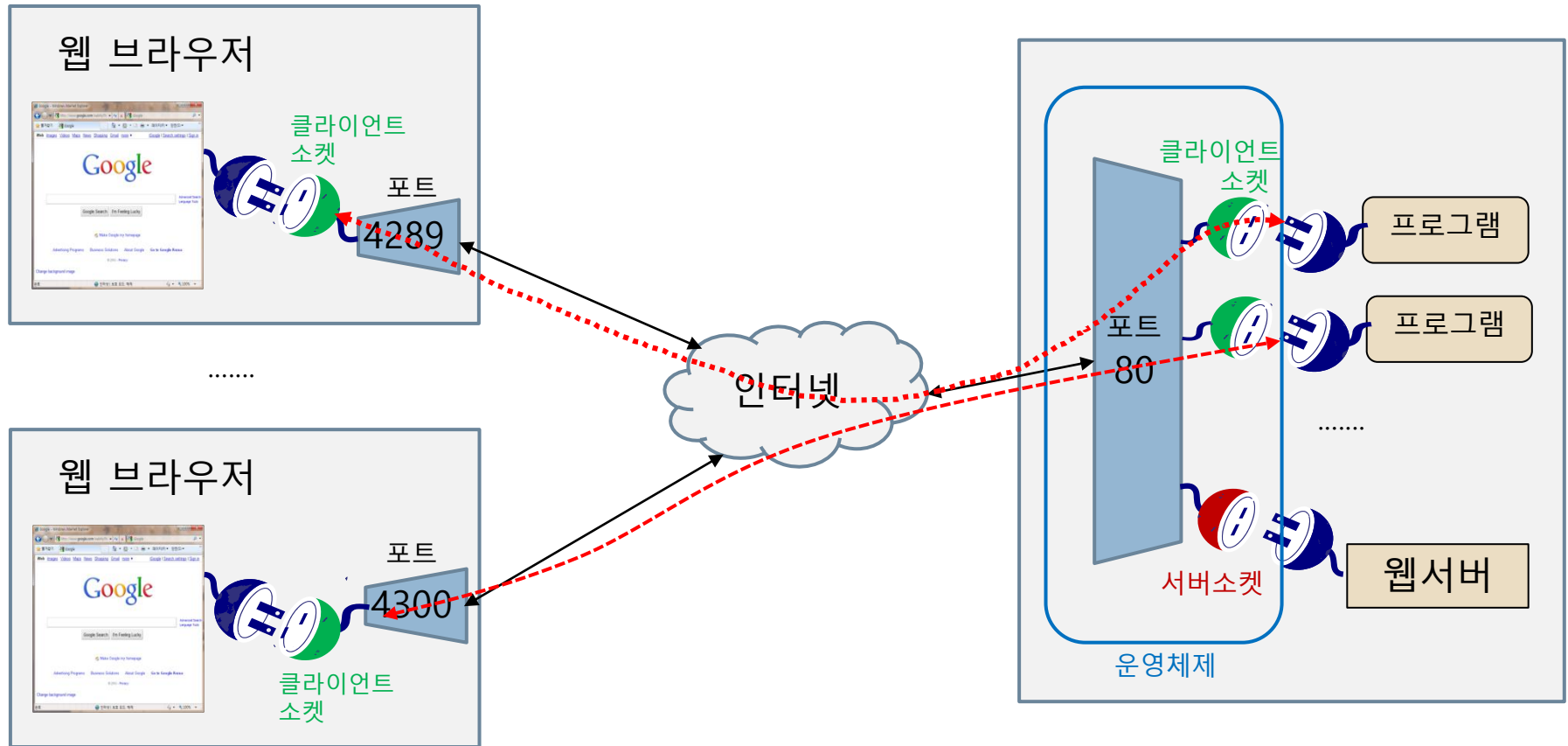
□ 소켓 (socket)

- TCP/IP 네트워크를 이용하여 쉽게 통신 프로그램을 작성하도록 지원하는 기반 기술
- 소켓
 - 두 응용프로그램 간의 양방향 통신 링크의 한쪽 끝 단
 - 소켓끼리 데이터를 주고받음
 - 소켓은 특정 IP 포트 번호와 결합
- 자바로 소켓 통신할 수 있는 라이브러리 지원
- 소켓 종류 : 서버 소켓과 클라이언트 소켓



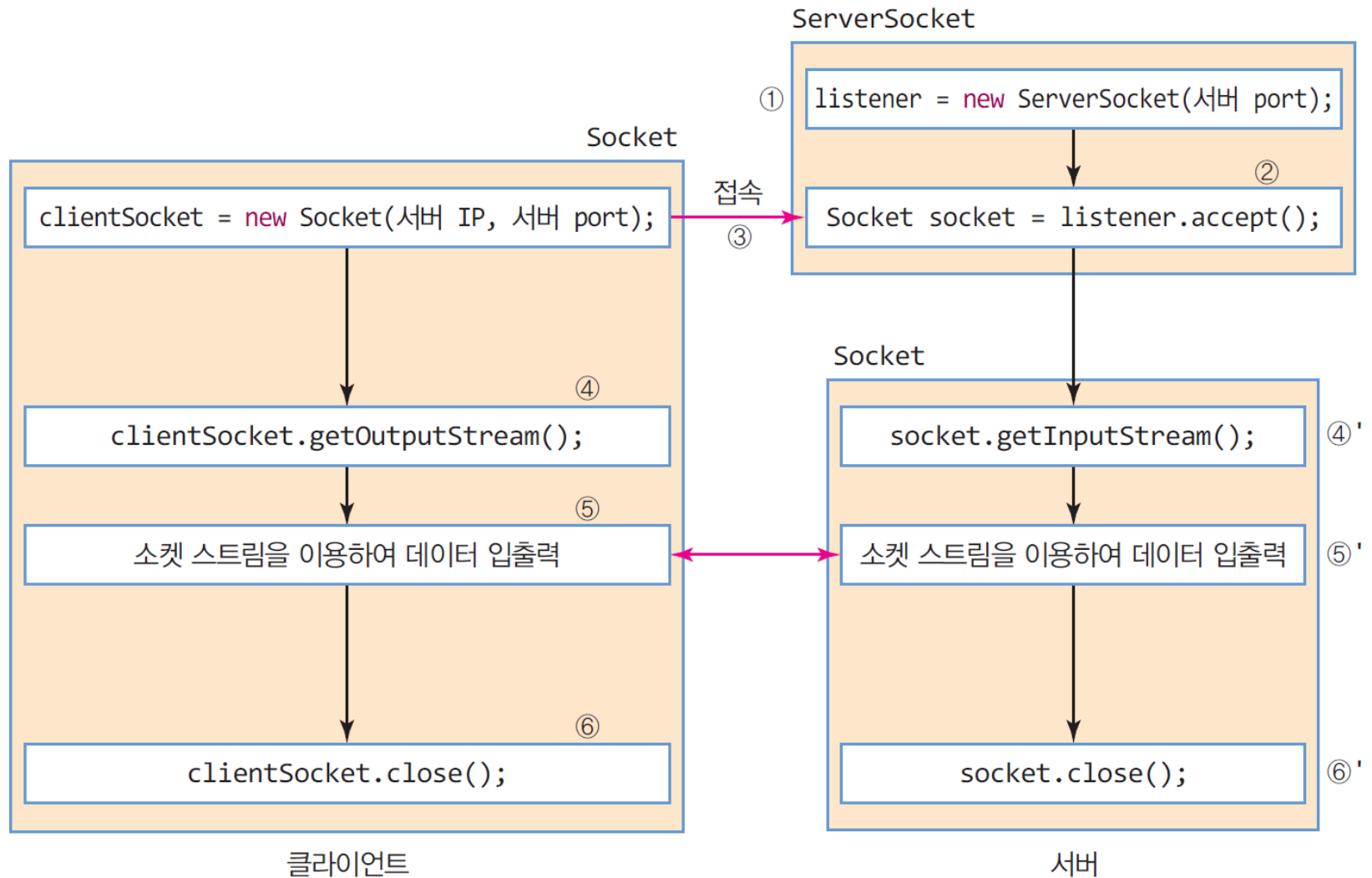
소켓을 이용한 웹 서버와 클라이언트 사이의 통신 사례

13



소켓을 이용한 서버 클라이언트 통신 프로그램의 전형적인 구조

14



Socket 클래스, 클라이언트 소켓

15

- Socket 클래스
 - ▣ 클라이언트 소켓에 사용되는 클래스
 - ▣ java.net 패키지에 포함
 - ▣ 생성자

생성자	설명
Socket	연결되지 않은 상태의 소켓을 생성
Socket(InetAddress address, int port)	소켓을 생성하고, 지정된 IP 주소(addresss)와 포트 번호(port)에서 대기하는 원격 응용프로그램의 소켓에 연결
Socket(String host, int port)	소켓을 생성하여 지정된 호스트(host)와 포트 번호(port)에 연결한다. 호스트 이름이 null인 경우는 루프백(loopback) 주소로 가정

Socket 클래스의 메소드

16

메소드	설명
<code>void bind(SocketAddress bindpoint)</code>	소켓에 로컬 IP 주소와 로컬 포트 지정
<code>void close()</code>	소켓을 닫는다.
<code>void connect(SocketAddress endpoint)</code>	소켓을 서버에 연결
<code>InetAddress getInetAddress()</code>	소켓에 연결된 서버 IP 주소 반환
<code>InputStream getInputStream()</code>	소켓에 연결된 입력 스트림 반환. 이 스트림을 이용하여 소켓이 상대방으로부터 받은 데이터를 읽을 수 있음
<code>InetAddress getLocalAddress()</code>	소켓이 연결된 로컬 주소 반환
<code>int getLocalPort()</code>	소켓의 로컬 포트 번호 반환
<code>int getPort()</code>	소켓에 연결된 서버의 포트 번호 반환
<code>OutputStream getOutputStream()</code>	소켓에 연결된 출력 스트림 반환. 이 스트림에 출력하면 소켓이 서버로 데이터 전송
<code>boolean isBound()</code>	소켓이 로컬 주소에 연결되어있으면 true 반환
<code>boolean isConnected()</code>	소켓이 서버에 연결되어 있으면 true 반환
<code>boolean isClosed()</code>	소켓이 닫혀있으면 true 반환
<code>void setSoTimeout(int timeout)</code>	데이터 읽기 타임아웃 시간 지정. 0이면 타임아웃 해제

클라이언트에서 소켓으로 서버에 접속하는 코드

17

- 클라이언트 소켓 생성 및 서버에 접속

```
Socket clientSocket = new Socket("128.12.1.1", 5550);
```

- Socket 객체의 생성되면 곧 바로 128.12.1.1의 주소의 5550포트에 자동 접속

- 소켓으로부터 데이터를 전송할 입출력 스트림 생성

```
BufferedReader in = new BufferedReader(
    new InputStreamReader(clientSocket.getInputStream()));
BufferedWriter out = new BufferedWriter(
    new OutputStreamWriter(clientSocket.getOutputStream()));
```

- 서버로 데이터 전송

- flush()를 호출하면 스트림 속에 데이터를 남기지 않고 모두 전송

```
out.write("hello" + "\n");
out.flush();
```

- 서버로부터 데이터 수신

```
String line = in.readLine();
//서버로부터 한 행의 문자열 수신
```

- 네트워크 접속 종료

```
clientSocket.close();
```

ServerSocket 클래스, 서버 소켓

18

□ ServerSocket 클래스

- ▣ 서버 소켓에 사용되는 클래스, java.net 패키지에 포함
- ▣ 생성자

생성자	설명
ServerSocket(int port)	지정된 포트 번호(port)와 결합된 소켓 생성

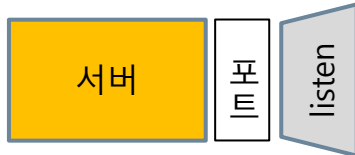
▣ 메소드

메소드	설명
Socket accept()	클라이언트로부터 연결 요청을 기다리다 요청이 들어오면 수락하고 클라이언트와 데이터를 주고받을 새 Socket 객체를 반환
void close()	서버 소켓을 닫는다.
InetAddress getInetAddress()	서버 소켓의 로컬 IP 주소 반환
int getLocalPort()	서버 소켓의 로컬 포트 번호 반환
boolean isBound()	서버 소켓이 로컬 주소에 연결되어있으면 true 반환
boolean isClosed()	서버 소켓이 닫혀있으면 true 반환
void setSoTimeout(int timeout)	accept()가 대기하는 타임아웃 시간 지정. 0이면 무한정 대기

서버에 클라이언트가 연결되는 과정

19

- 서버는 서버 소켓으로 들어오는 연결 요청을 기다림(listen)

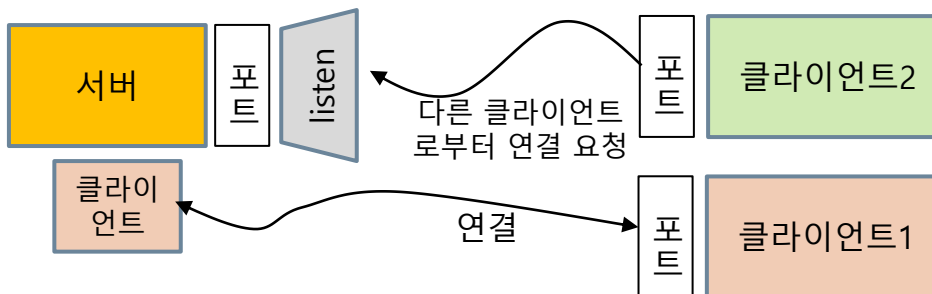


- 클라이언트가 서버에게 연결 요청



- 서버가 연결 요청 수락(accept)

- 새로운 클라이언트 소켓을 만들어 클라이언트와 통신하게 함
- 그리고 다시 다른 클라이언트의 연결을 기다림



서버가 클라이언트와 통신하는 과정

20

▣ 서버 소켓 생성

```
ServerSocket serverSocket = new ServerSocket(5550);
```

- 서버는 접속을 기다리는 포트로 5550 선택

▣ 클라이언트로부터 접속 기다림

```
Socket socket = serverSocket.accept();
```

- accept() 메소드는 연결 요청이 오면 새로운 Socket 객체 반환
- 접속 후 새로 만들어진 Socket 객체를 통해 클라이언트와 통신

▣ 네트워크 입출력 스트림 생성

```
BufferedReader in = new BufferedReader(  
    new InputStreamReader(socket.getInputStream()));  
BufferedWriter out = new BufferedWriter(  
    new OutputStreamWriter(socket.getOutputStream()));
```

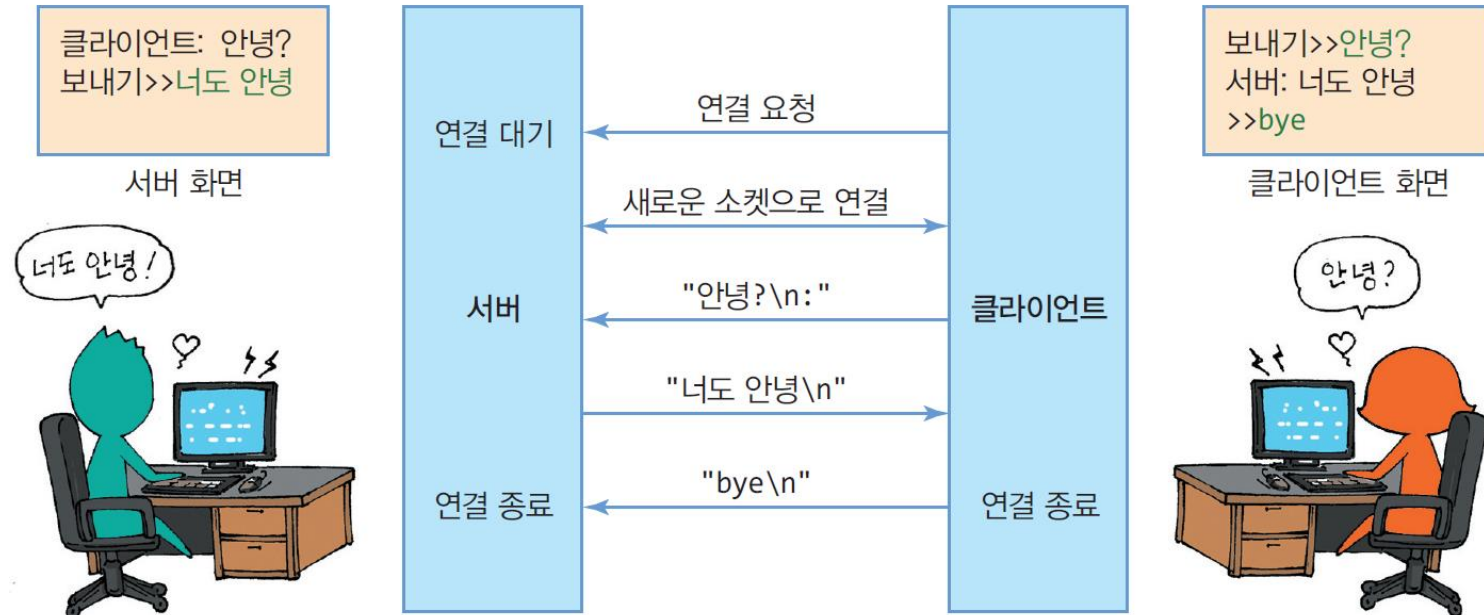
- Socket 객체의 getInputStream()과 getOutputStream() 메소드를 이용하여 입출력 데이터 스트림 생성

소켓을 이용한 서버/클라이언트 채팅 예제

21

□ 간단한 채팅 프로그램

- 서버와 클라이언트가 1:1로 채팅
- 클라이언트와 서버가 서로 한번씩 번갈아 가면서 문자열 전송
 - 문자열 끝에 "\n"을 덧붙여 보내고 라인 단위로 수신
- 클라이언트가 bye를 보내면 프로그램 종료



서버 프로그램

ServerEx.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ServerEx {
    public static void main(String[] args) {
        BufferedReader in = null;
        BufferedWriter out = null;
        ServerSocket listener = null;
        Socket socket = null;
        Scanner scanner = new Scanner(System.in); // 키보드에서 읽을 scanner 객체 생성
        try {
            listener = new ServerSocket(9999); // 서버 소켓 생성
            System.out.println("연결을 기다리고 있습니다.....");
            socket = listener.accept(); // 클라이언트로부터 연결 요청 대기
            System.out.println("연결되었습니다.");
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            while (true) {
                String inputMessage = in.readLine(); // 클라이언트로부터 한 행 읽기
                if (inputMessage.equalsIgnoreCase("bye")) {
                    System.out.println("클라이언트에서 bye로 연결을 종료하였음");
                    break; // "bye"를 받으면 연결 종료
                }
                System.out.println("클라이언트: " + inputMessage);
                System.out.print("보내기>>"); // 프롬프트
                String outputMessage = scanner.nextLine(); // 키보드에서 한 행 읽기
                out.write(outputMessage + "\n"); // 키보드에서 읽은 문자열 전송
                out.flush(); // out의 스트림 버퍼에 있는 모든 문자열 전송
            }
        } catch (IOException e) { System.out.println(e.getMessage());
        } finally {
            try {
                scanner.close(); // scanner 닫기
                socket.close(); // 통신용 소켓 닫기
                listener.close(); // 서버 소켓 닫기
            } catch (IOException e) { System.out.println("클라이언트와 채팅 중 오류가 발생했습니다."); }
        }
    }
}
```

클라이언트 프로그램 ClientEx.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class ClientEx {
    public static void main(String[] args) {
        BufferedReader in = null;
        BufferedWriter out = null;
        Socket socket = null;
        Scanner scanner = new Scanner(System.in); // 키보드에서 읽을 scanner 객체 생성
        try {
            socket = new Socket("localhost", 9999); // 클라이언트 소켓 생성. 서버에 연결
            in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
            while (true) {
                System.out.print("보내기>>"); // 프롬프트
                String outputMessage = scanner.nextLine(); // 키보드에서 한 행 읽기
                if (outputMessage.equalsIgnoreCase("bye")) {
                    out.write(outputMessage + "\n"); // "bye" 문자열 전송
                    out.flush();
                    break; // 사용자가 "bye"를 입력한 경우 서버로 전송 후 실행 종료
                }
                out.write(outputMessage + "\n"); // 키보드에서 읽은 문자열 전송
                out.flush(); // out의 스트림 버퍼에 있는 모든 문자열 전송
                String inputMessage = in.readLine(); // 서버로부터 한 행 수신
                System.out.println("서버: " + inputMessage);
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                scanner.close();
                if(socket != null) socket.close(); // 클라이언트 소켓 닫기
            } catch (IOException e) {
                System.out.println("서버와 채팅 중 오류가 발생했습니다.");
            }
        }
    }
}
```




스레드와 멀티태스킹

멀티태스킹(multi-tasking) 개념

25

□ 멀티태스킹

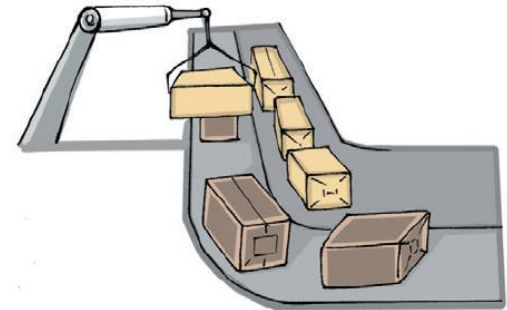
- ▣ 하나의 응용프로그램이 여러 개의 작업(태스크)을 동시에 처리



다림질하면서
전화 받는 주부



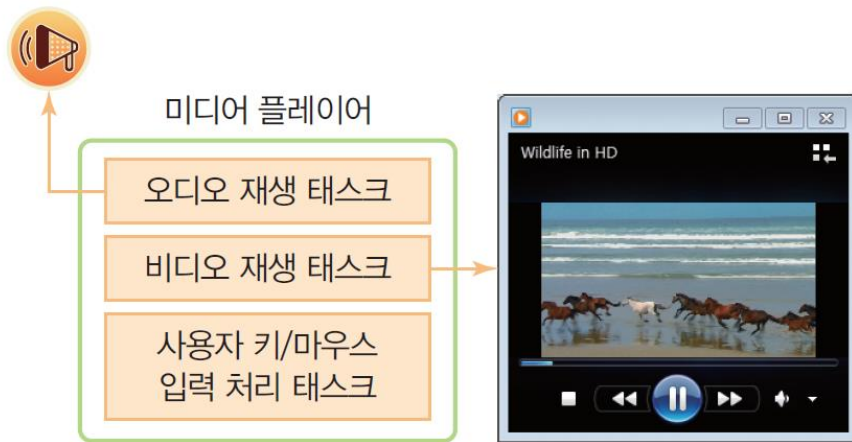
운전하면서 음악을
듣는 연수 양



판독과 포장을
동시에 하는 기계

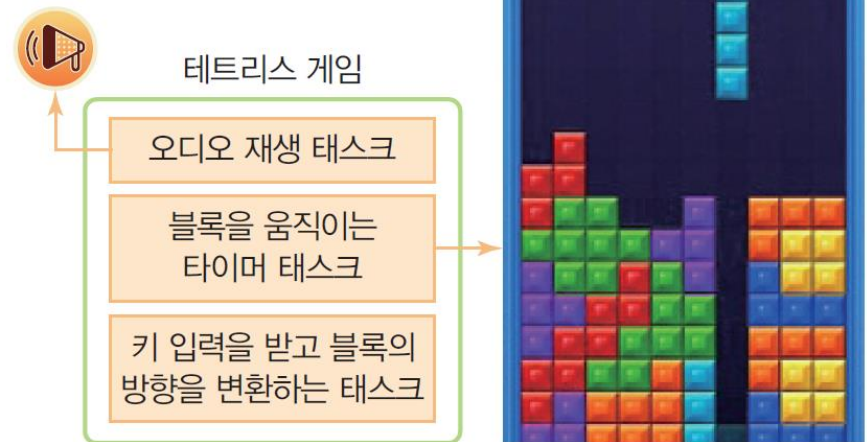
멀티태스킹 응용프로그램 사례

26



(a) 미디어 플레이어의 멀티태스킹

* 3개의 태스크 동시 실행

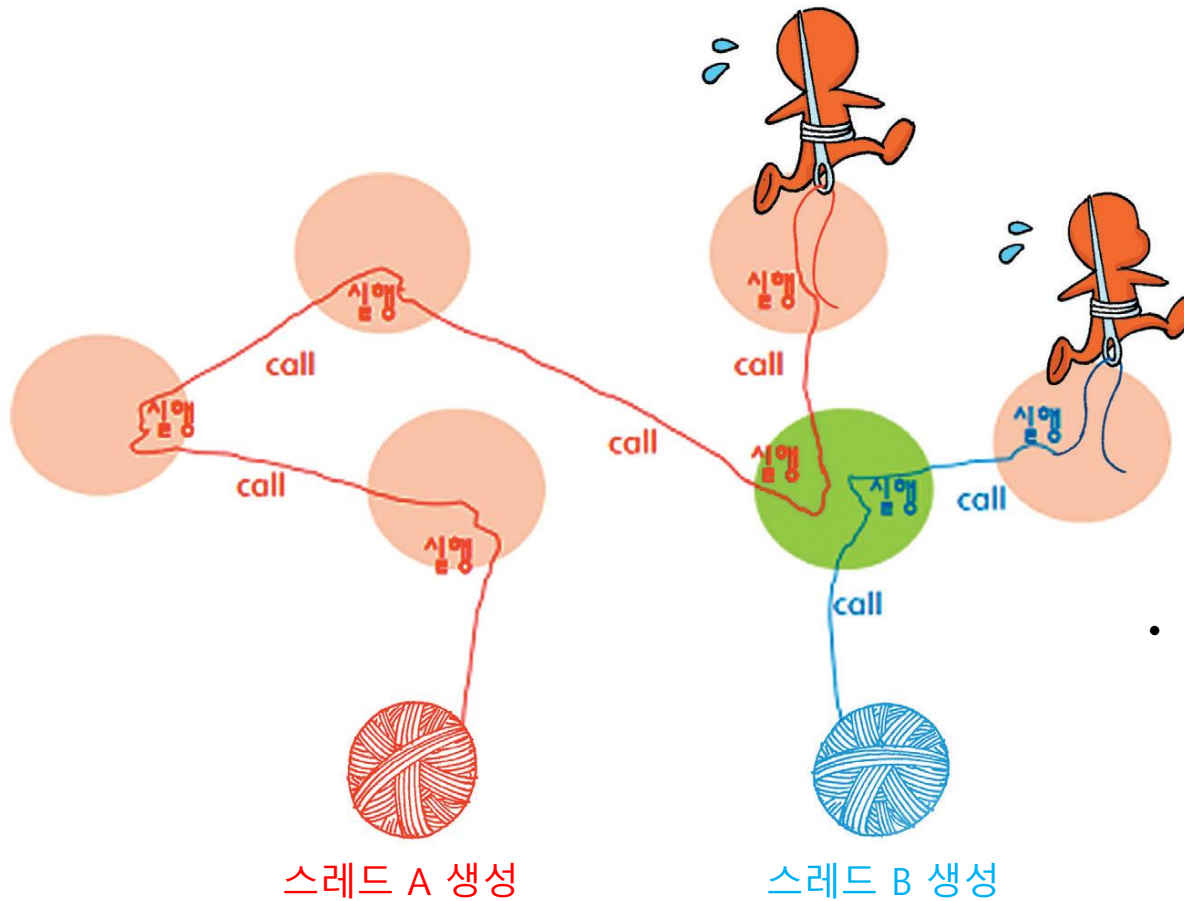


(b) 테트리스 게임의 멀티태스킹

* 3개의 태스크 동시 실행

스레드(thread) 개념과 실(thread)

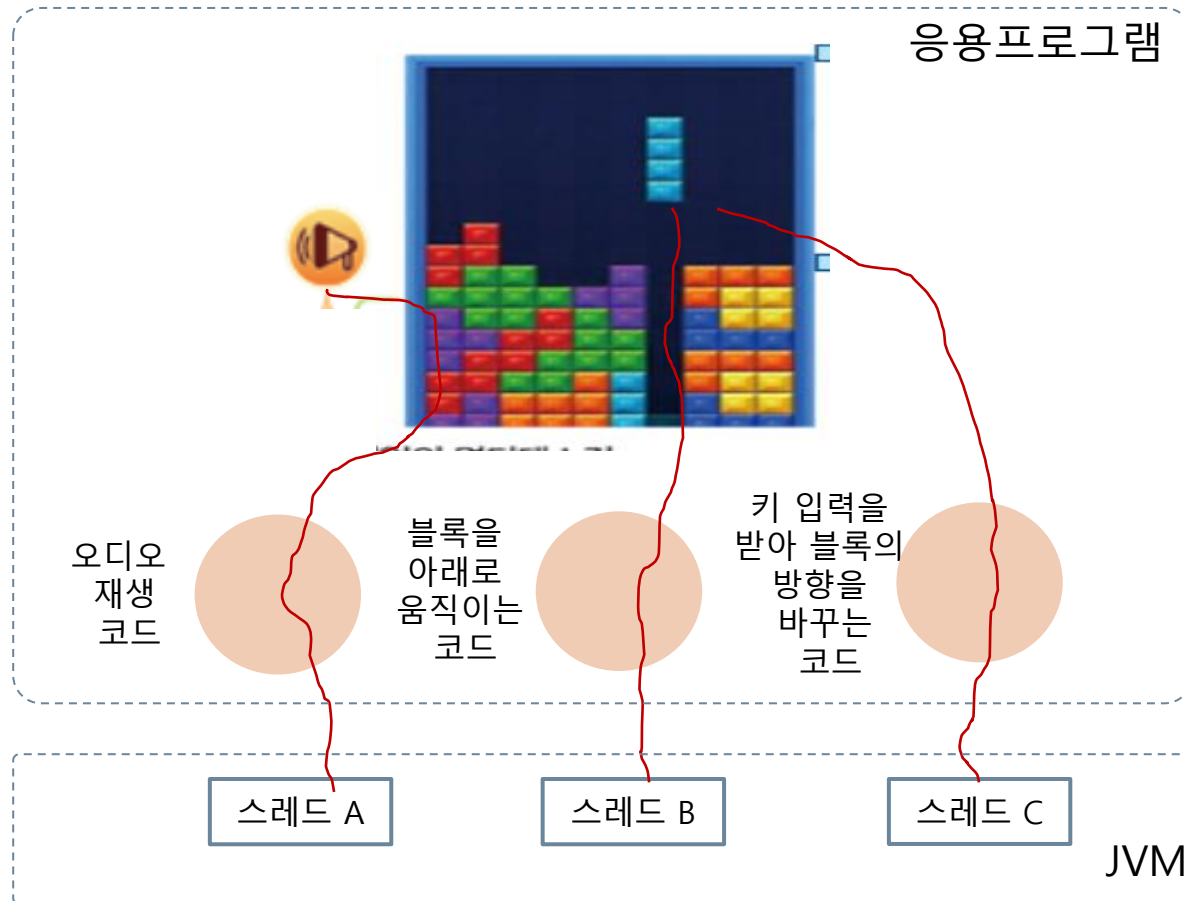
27



- 마치 바늘이 하나의 실(thread)을 가지고 바느질하는 것과 자바의 스레드는 일맥 상통함

테트리스 프로그램을 구성하는 멀티스레드 분석

28



* JVM은 3 개의 스레드 중 하나를 선택하여 실행시킨다. 예를 들어 스레드 A를 선택하면 스레드의 A의 코드(사용자가 작성한 코드)를 호출한다. 스레드 A를 일시 중단하고, JVM이 스레드 B를 실행시켜려면 다시 스레드 B의 코드(사용자가 작성한 코드)를 호출한다.

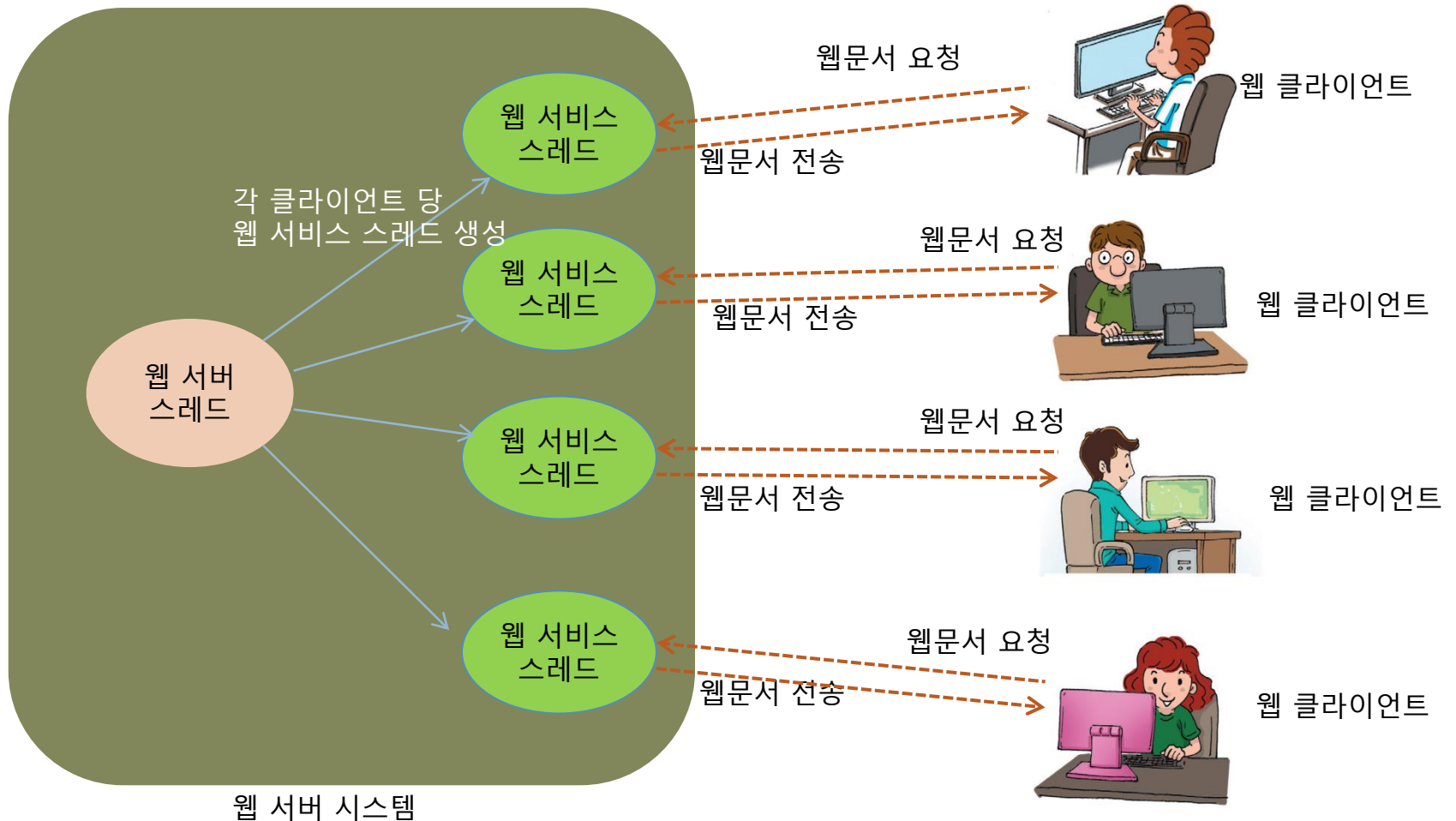
스레드와 멀티스레딩

29

- 스레드
 - ▣ 사용자가 작성한 코드로서, JVM에 의해 스케줄링되어 실행되는 단위
- 자바의 멀티태스킹
 - ▣ 하나의 응용프로그램은 여러 개의 스레드로 구성 가능
- 멀티스레딩의 효과
 - ▣ 한 스레드가 대기하는 동안 다른 스레드 실행
 - ▣ 프로그램 전체적으로 시간 지연을 줄임

웹 서버의 멀티스레딩 사례

30



자바 스레드(Thread)란?

31

□ 자바 스레드

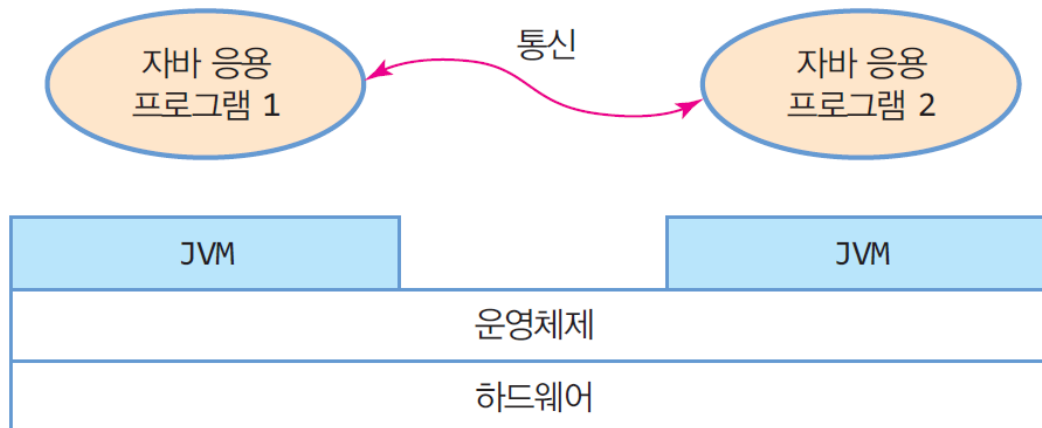
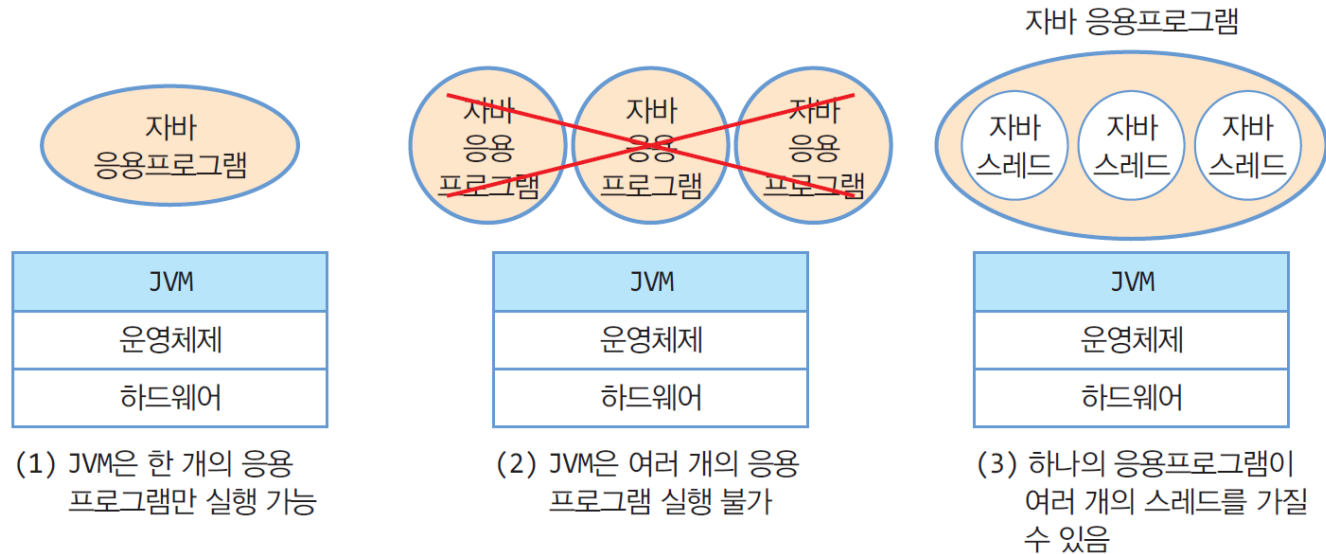
- ▣ 자바 가상 기계(JVM)에 의해 스케줄되는 실행 단위의 코드 블록
- ▣ 스레드의 생명 주기는 JVM에 의해 관리됨
 - JVM은 스레드 단위로 스케줄링

□ 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

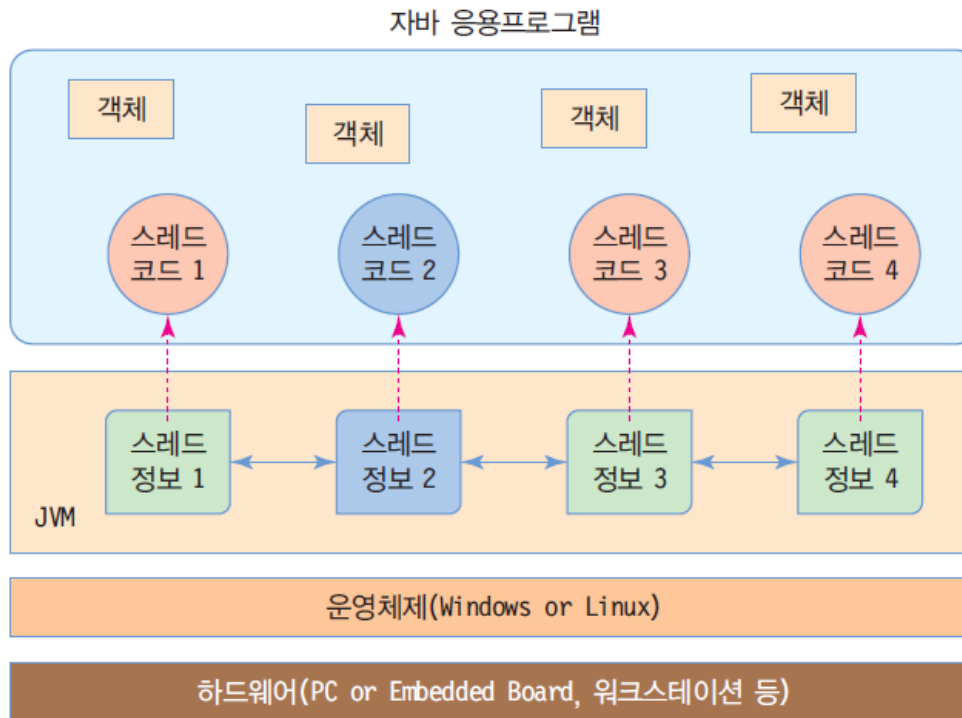
32



두 개의 자바 응용프로그램을 동시에 실행시키고자 하면 두 개의 JVM을 이용하고 응용프로그램은 서로 소켓 등을 이용하여 통신

자바 스레드와 JVM

33



- 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

자바에서 스레드 만들기

34

- 스레드 실행을 위해 개발자가 하는 작업
 - ▣ 스레드 코드 작성
 - ▣ 스레드를 생성하고 스레드 코드를 실행하도록 JVM에게 요청

- 스레드 만드는 2 가지 방법
 - ▣ `java.lang.Thread` 클래스를 이용하는 경우
 - ▣ `java.lang.Runnable` 인터페이스를 이용하는 경우

Thread 클래스를 이용한 스레드 생성

35

□ 스레드 클래스 작성

- Thread 클래스 상속. 새 클래스 작성

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

□ 스레드 코드 작성

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

□ 스레드 객체 생성

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

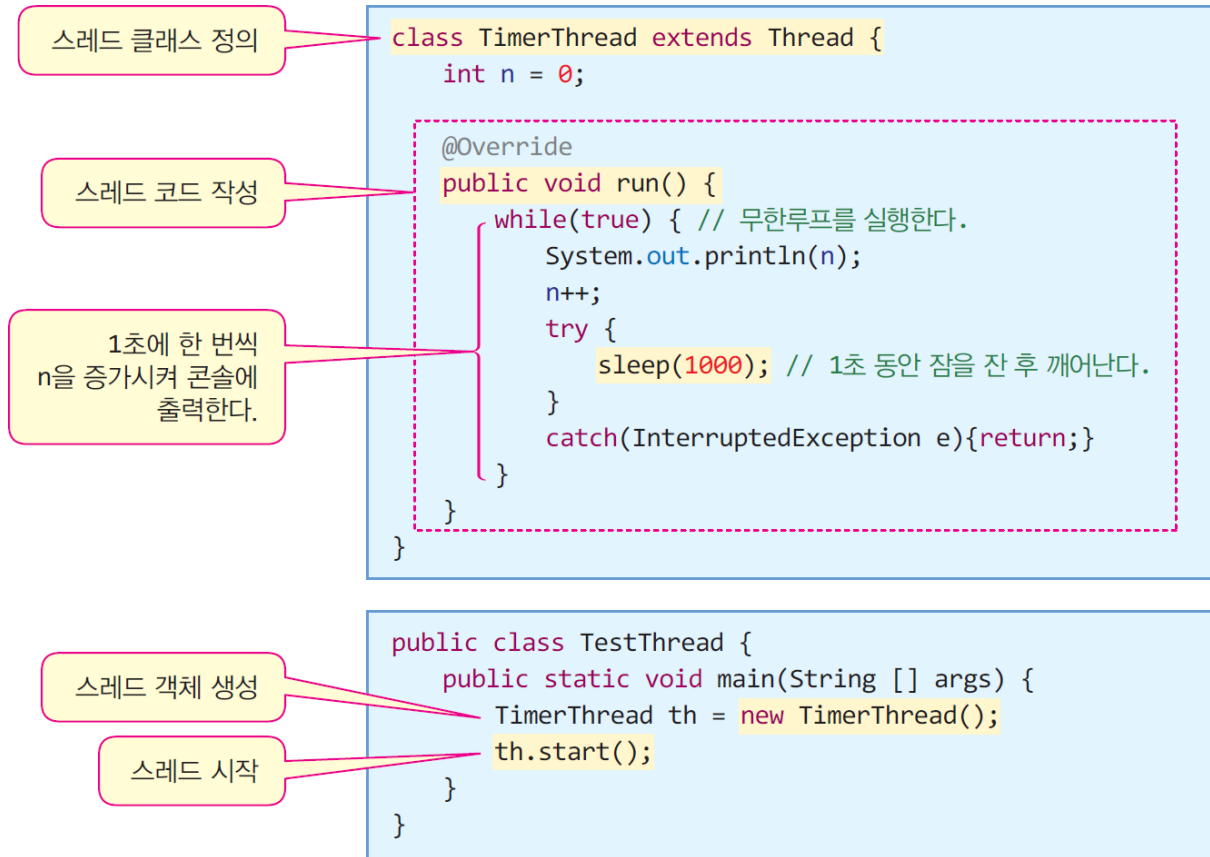
□ 스레드 시작

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

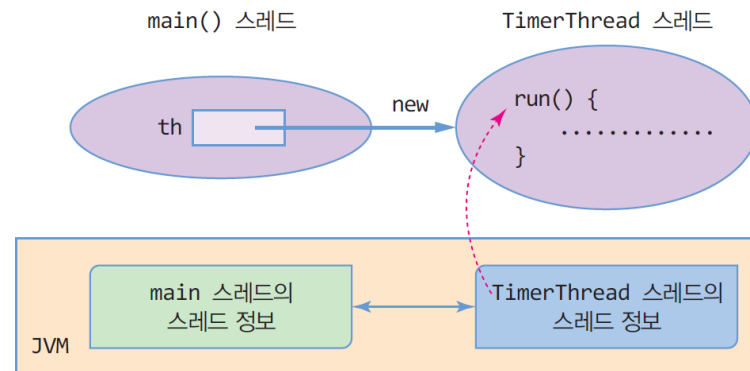
- start() 메소드 호출

- 스레드로 작동 시작, JVM에 의해 스케줄되기 시작함

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



0
1
2
3
4
.
.
.
.



스레드 만들 때 주의 사항

37

- `run()` 메소드가 종료하면 스레드는 종료한다.
 - ▣ 스레드가 계속 살아있게 하려면 `run()` 메소드 내 무한루프 작성
- 한번 종료한 스레드는 다시 시작시킬 수 없다.
 - ▣ 다시 스레드 객체를 생성하고 `start()`를 호출해야 함
- 한 스레드에서 다른 스레드를 강제 종료할 수 있다.
 - ▣ 뒤에서 다룸

Runnable 인터페이스로 스레드 만들기

38

- 스레드 클래스 작성
 - ▣ Runnable 인터페이스 구현하는 새 클래스 작성

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

- 스레드 코드 작성
 - ▣ run() 메소드 구현
 - run() 메소드를 스레드 코드라고 부름
 - run() 메소드에서 스레드 실행 시작

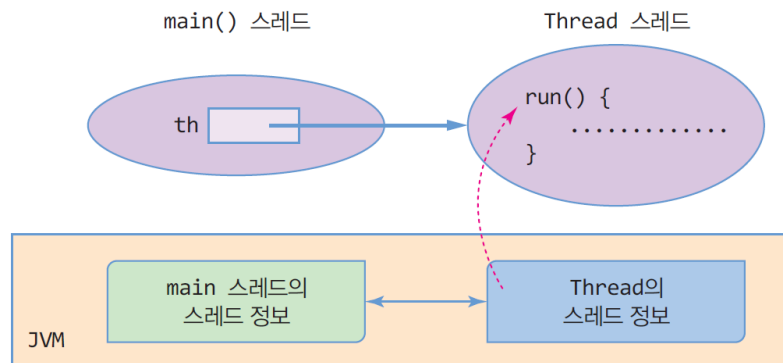
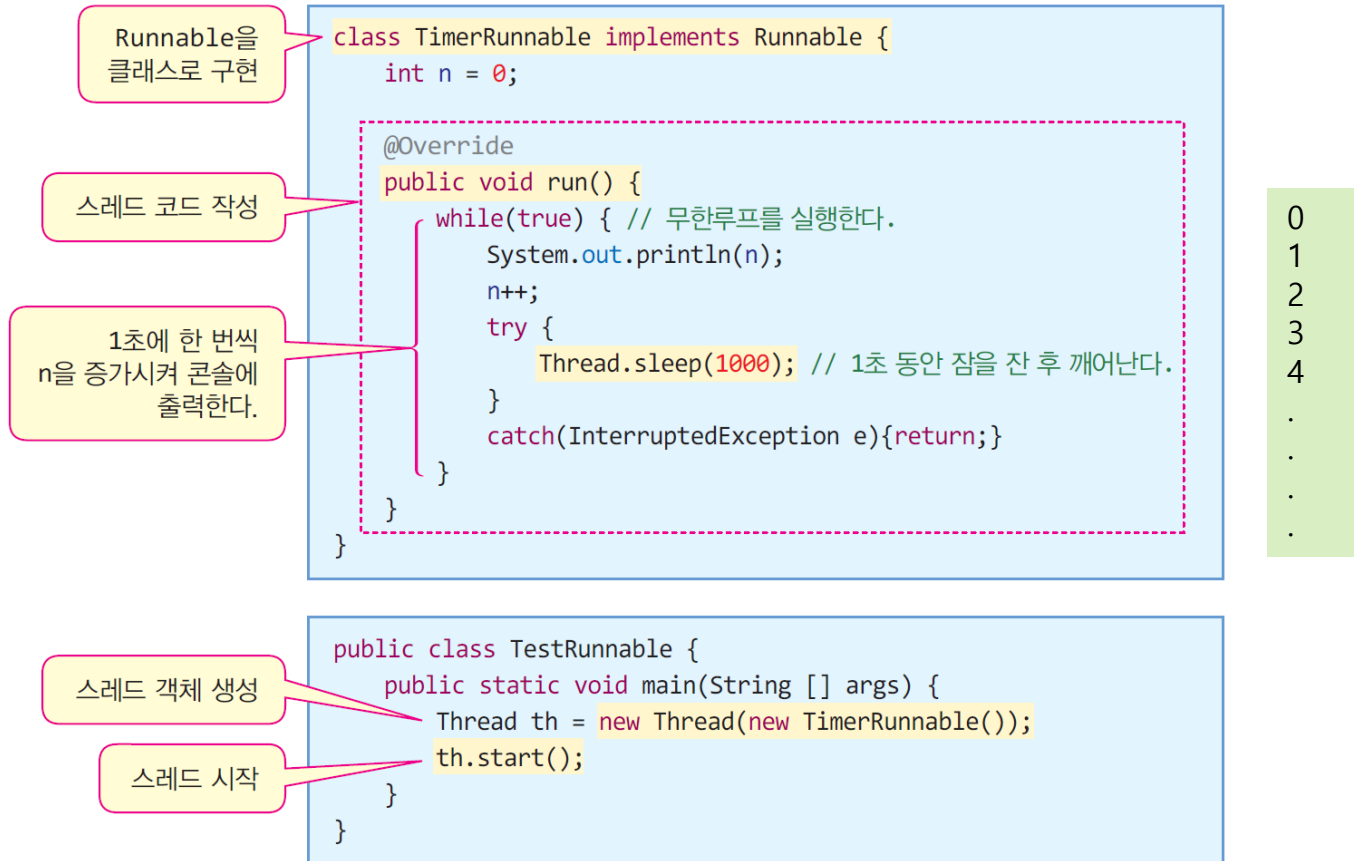
- 스레드 객체 생성

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

- 스레드 시작
 - ▣ start() 메소드 호출

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

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

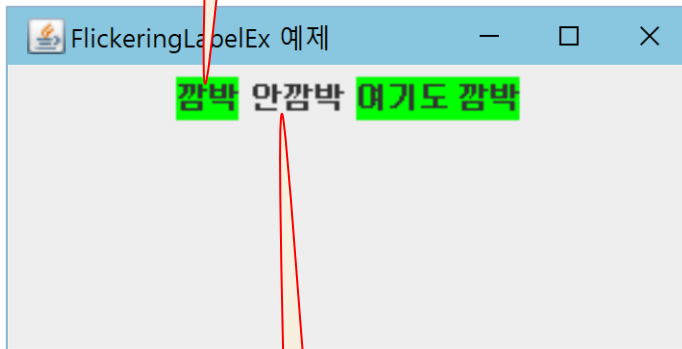


예제 13-3 : 깜박이는 문자열을 가진 레이블 만들기

40

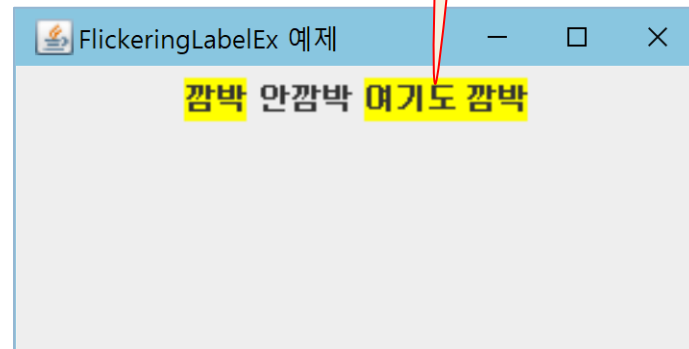
JLabel을 상속받아 문자열을 깜박이는 FlickeringLabel 컴포넌트를 작성하라.

500 밀리 주기로 초록색과 노란색으로 번갈아 깜박이는 레이블



깜박이지 않는 레이블

300 밀리 주기로 초록색과 노란색으로 번갈아 깜박이는 레이블



예제 13-3 : 정답 코드

41

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

```
class FlickeringLabel extends JLabel implements Runnable {
    private long delay;
    public FlickeringLabel(String text, long delay) {
        super(text);
        this.delay = delay;
        setOpaque(true);
        Thread th = new Thread(this);
        th.start();
    }
    @Override
    public void run() {
        int n=0;
        while(true) {
            if(n == 0)
                setBackground(Color.YELLOW);
            else
                setBackground(Color.GREEN);
            if(n == 0) n = 1;
            else n = 0;
            try {
                Thread.sleep(delay);
            }
            catch(InterruptedException e) {
                return;
            }
        }
    }
}
```

```
public class FlickeringLabelEx extends JFrame {
    public FlickeringLabelEx() {
        setTitle("FlickeringLabelEx 예제");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());

        // 깜박이는 레이블 생성
        FlickeringLabel fLabel = new FlickeringLabel("깜박",500);

        // 깜박이지 않는 레이블 생성
        JLabel label = new JLabel("안깜박");

        // 깜박이는 레이블 생성
        FlickeringLabel fLabel2 = new FlickeringLabel("여기도 깜박",300);

        c.add(fLabel);
        c.add(label);
        c.add(fLabel2);

        setSize(300,150);
        setVisible(true);
    }

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

스레드 정보

42

필드	타입	내용
스레드 이름	STRING	스레드의 이름으로서 사용자가 지정
스레드 ID	정수	스레드 고유의 식별자 번호
스레드의 PC(Program Count)	정수	현재 실행 중인 스레드 코드의 주소
스레드 상태	정수	NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCK, TERMINATED 등 6개 상태 중 하나
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

43

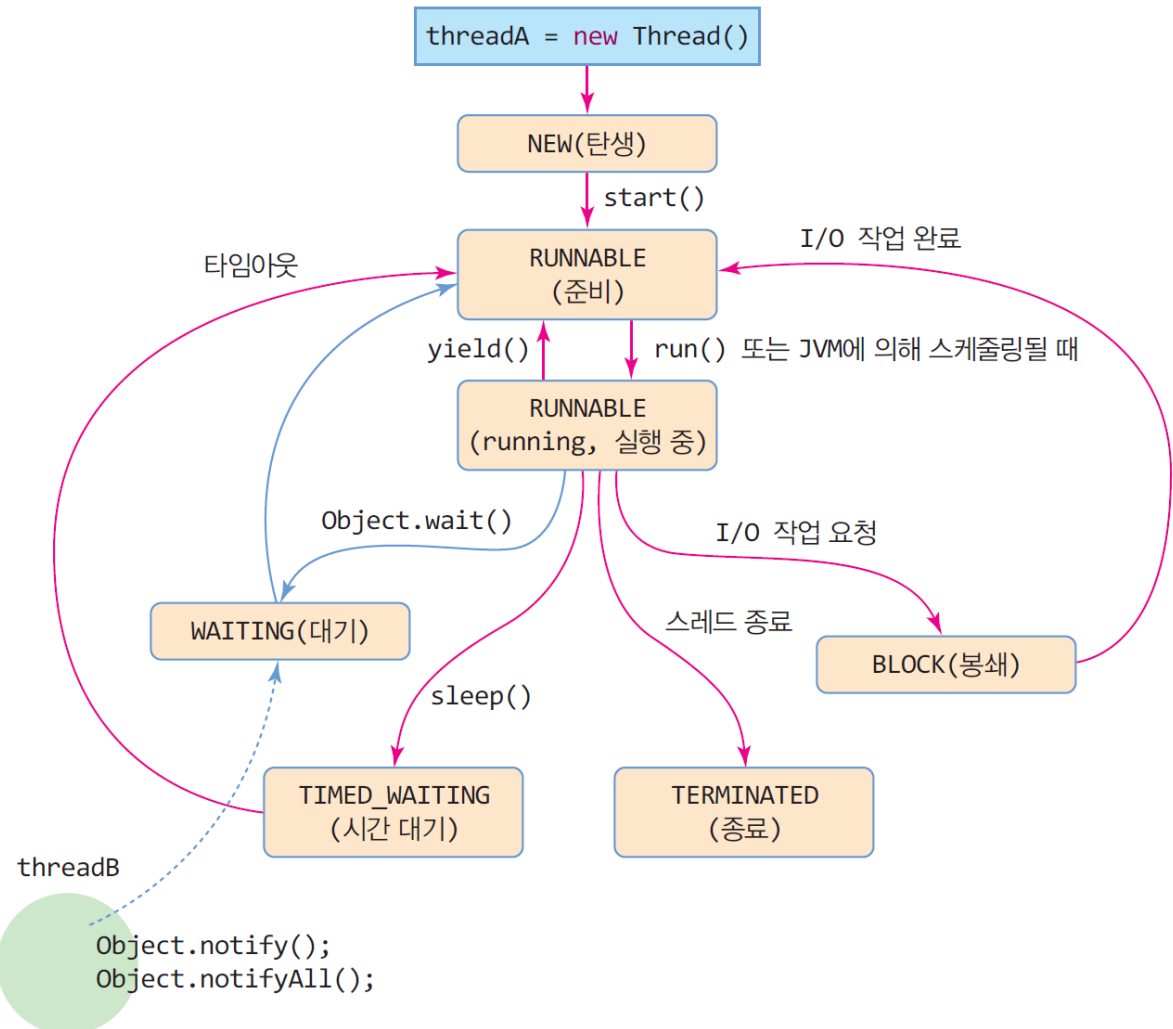
- 스레드 상태 6 가지
 - ▣ NEW
 - 스레드가 생성되었지만 스레드가 아직 실행할 준비가 되지 않았음
 - ▣ RUNNABLE
 - 스레드가 현재 실행되고 있거나
 - 실행 준비되어 스케줄링을 기다리는 상태
 - ▣ WAITING
 - wait()를 호출한 상태
 - 다른 스레드가 notify()나 notifyAll()을 불러주기를 기다리고 있는 상태
 - 스레드 동기화를 위해 사용
 - ▣ TIMED_WAITING
 - sleep(n)을 호출하여 n 밀리초 동안 잠을 자고 있는 상태
 - ▣ BLOCK
 - 스레드가 I/O 작업을 요청하면 JVM이 자동으로 BLOCK 상태로 만듦
 - ▣ TERMINATED
 - 스레드가 종료한 상태
- 스레드 상태는 JVM에 의해 기록 관리됨

스레드 상태와 생명 주기

44

스레드 상태 6 가지

- NEW
- RUNNABLE
- WAITING
- TIMED_WAITING
- BLOCK
- TERMINATED



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

스레드 우선순위와 스케줄링

45

- 스레드의 우선순위
 - ▣ 최대값 = 10(MAX_PRIORITY)
 - ▣ 최소값 = 1(MIN_PRIORITY)
 - ▣ 보통값 = 5(NORMAL_PRIORITY)
- 스레드 우선순위는 응용프로그램에서 변경 가능
 - ▣ `void setPriority(int priority)`
 - ▣ `int getPriority()`
- `main()` 스레드의 우선순위 값은 초기에 5
- 스레드는 부모 스레드와 동일한 우선순위 값을 가지고 탄생
- JVM의 스케줄링 정책
 - ▣ 철저한 우선순위 기반
 - 가장 높은 우선순위의 스레드가 우선적으로 스케줄링
 - 동일한 우선순위의 스레드는 돌아가면서 스케줄링(라운드 로빈)

main()을 실행하는 main 스레드

46

- main 스레드와 main() 메소드
 - ▣ JVM은 응용프로그램을 실행을 시작할 때 스레드 생성 : main 스레드
 - ▣ JVM은 main 스레드에게 main() 메소드 실행하도록 함
 - main() 메소드가 종료하면 main 스레드 종료

예제 13-4 : main 스레드의 정보 출력

47

main() 메소드에서 현재 실행 중인 스레드 정보를 출력하라

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

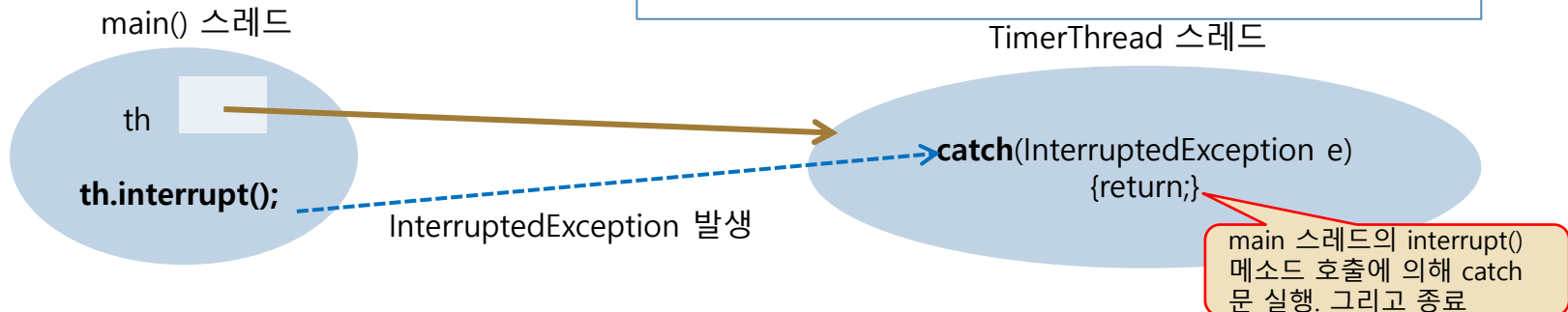
48

- 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 {  
    private int n = 0;  
    @Override  
    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  
            }  
        }  
    }  
}
```

만일 return 하지 않으면
스레드는 종료하지 않음



스레드 동기화(Thread Synchronization)

49

- 멀티스레드 프로그램 작성시 주의점
 - ▣ 다수 스레드가 공유 데이터에 동시에 접근하는 경우
 - 공유 데이터의 값에 예상치 못한 결과 발생 가능
- 스레드 동기화
 - ▣ 공유 데이터에 대한 멀티스레드의 동시 접근 문제 해결책
 - 공유 데이터를 접근하는 모든 스레드 한 줄 세우기
 - 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 대기하도록 함

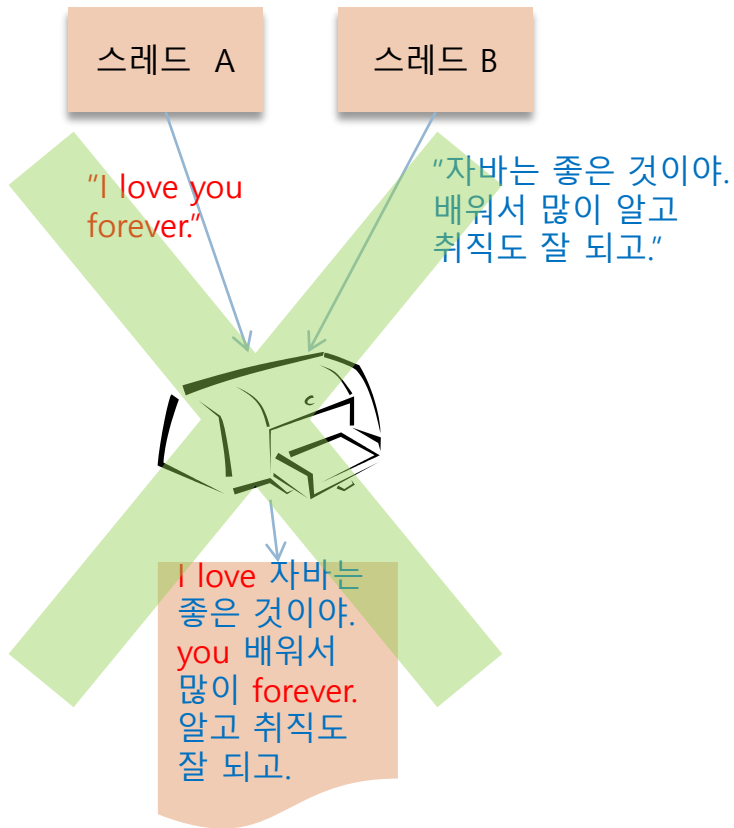
두 스레드의 프린터 동시 쓰기로 충돌하는 사례

50

스레드 B

"자바는 좋은 것이야.
배워서 많이 알고
취직도 잘 되고."

스레드 A가 프린터 사용을 끝낼때까지 기다린다.



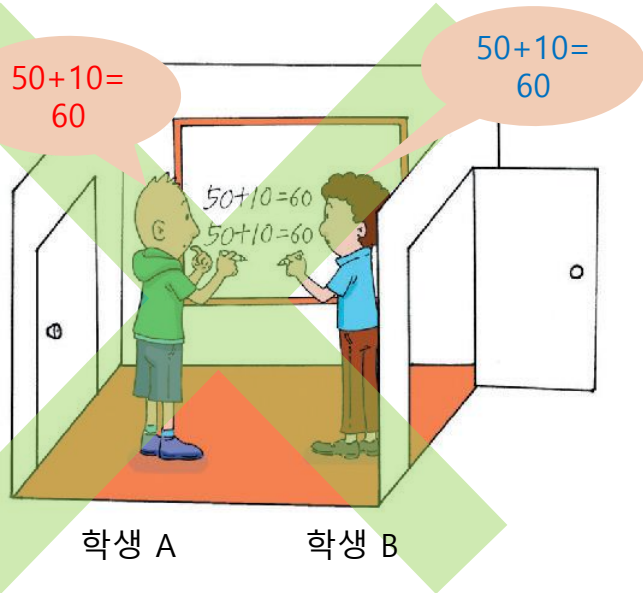
두 스레드가 동시에 프린터에 쓰는 경우
문제 발생



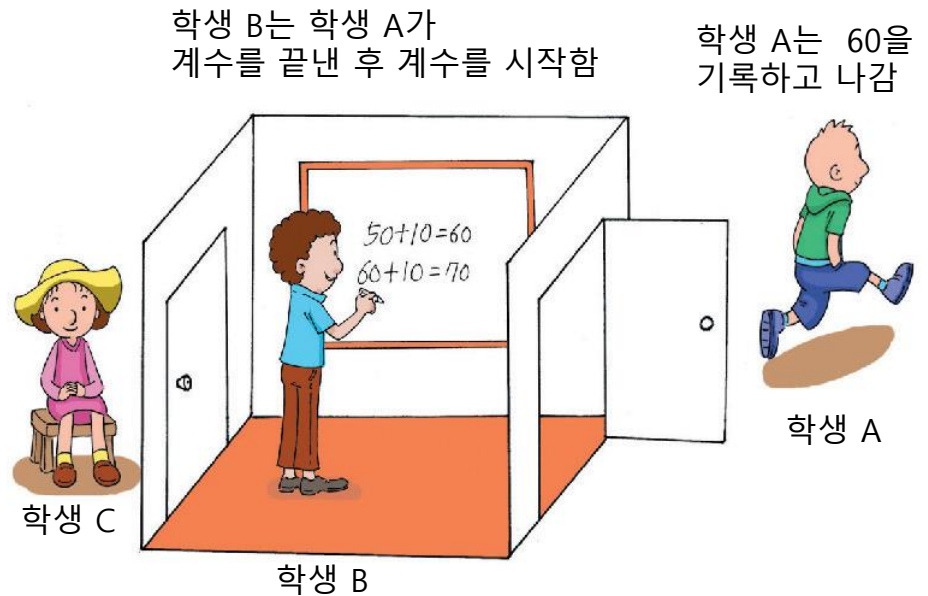
한 스레드의 출력이 끝날 때까지
대기함으로써 정상 출력

공유 집계판에 동시 접근하는 사례

51



두 학생이 동시에 방에 들어와서
집계판을 수정하는 경우
집계판의 **결과가 잘못됨**



방에 먼저 들어간 학생이
집계를 끝내기를 기다리면
정상 처리

스레드 동기화 기법

52

□ 스레드 동기화

- ▣ 공유 데이터에 동시에 접근하는 다수의 스레드가 공유 데이터를 배타적으로 접근하기 위해 상호 협력(coordination)하는 것
- ▣ 동기화의 핵심
 - 스레드의 공유 데이터에 대한 배타적 독점 접근 보장

□ 자바에서 스레드 동기화 방법

- ▣ synchronized로 동기화 블록 지정
- ▣ wait()-notify() 메소드로 스레드 실행 순서 제어

synchronized 블록 지정

53

- synchronized 키워드
 - ▣ 한 스레드가 독점 실행해야 하는 부분(동기화 코드)을 표시하는 키워드
 - 임계 영역(critical section) 표기 키워드
 - ▣ 메소드 전체 혹은 코드 블록
- synchronized 블록에 대한 컴파일러의 처리
 - ▣ 먼저 실행한 스레드가 모니터 소유
 - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
 - ▣ 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드 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 코드 블록

synchronized 사용 예 : 공유 집계판 사례를 코딩

54

```
public class SynchronizedEx {
    public static void main(String [] args) {
        SharedBoard board = new SharedBoard();
        Thread th1 = new StudentThread("kitae", board);
        Thread th2 = new StudentThread("hyosoo", board);
        th1.start();
        th2.start();
    }
}

class SharedBoard {
    private int sum = 0; // 집계판의 합
    synchronized public void add() {
        int n = sum;
        Thread.yield(); // 현재 실행 중인 스레드 양보
        n += 10; // 10 증가
        sum = n; // 증가한 값을 집계합에 기록
        System.out.println(Thread.currentThread().getName() + " : " + sum);
    }
    public int getSum() { return sum; }
}

class StudentThread extends Thread {
    private SharedBoard board; // 집계판의 주소
    public StudentThread(String name, SharedBoard board) {
        super(name);
        this.board = board;
    }

    @Override
    public void run() {
        for(int i=0; i<10; i++)
            board.add();
    }
}
```

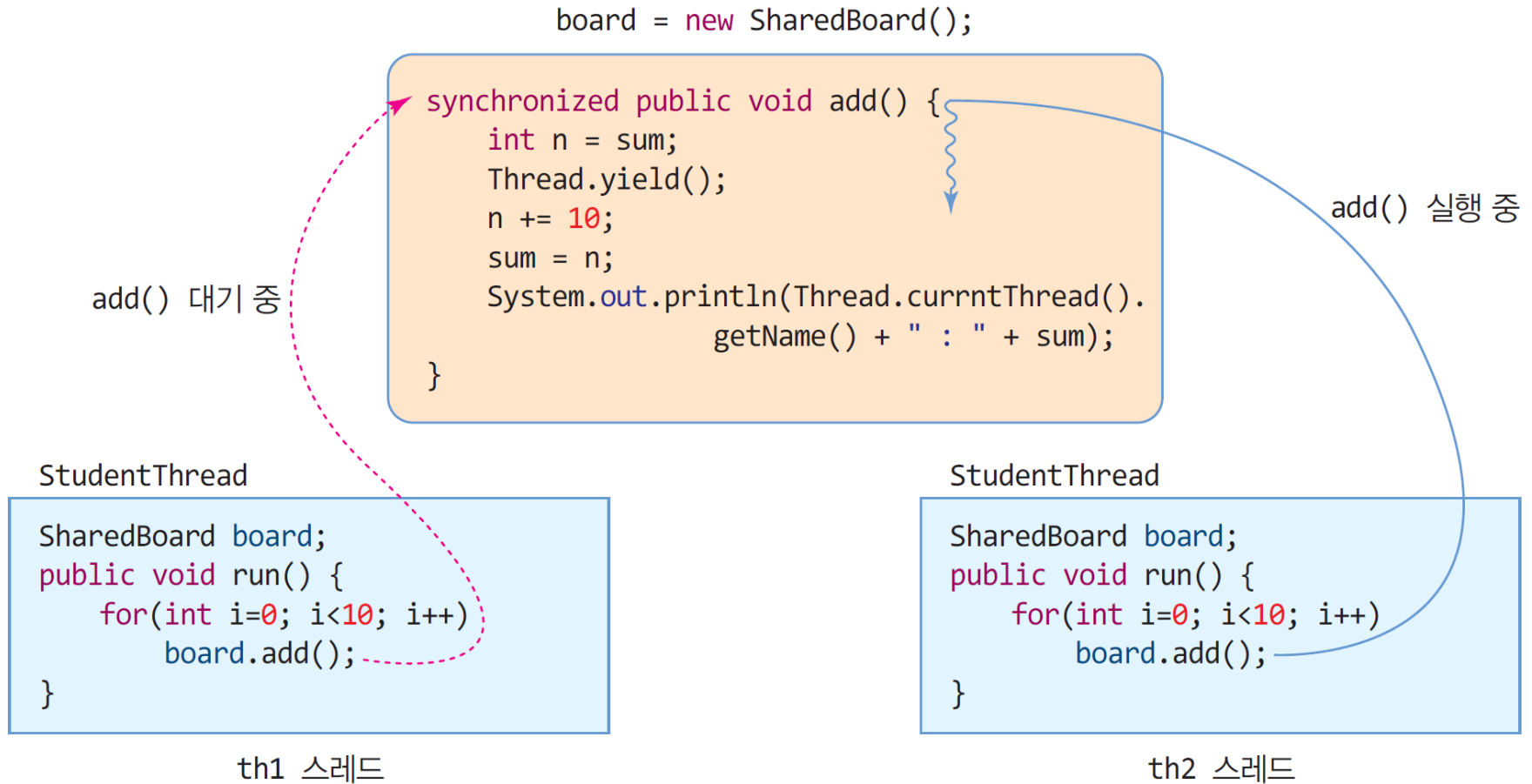
- 집계판 : class SharedBoard
- 각 학생 : class StudentThread
(각 학생은 하나의 스레드)

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

kitae와 hyosoo가 각각
10번씩 add()를 호출,
동기화가 잘 이루어져서
최종 누적 점수 sum이 200이 됨

스레드2가 SharedBoard의 add()를 실행하고 있는 동안, 스레드1이 호출할 때 대기

55



공유집계판 사례에서 synchronized 사용하지 않아 충돌로 인해 데이터에 오류가 발생한 경우

56

```
public class SynchronizedEx {
    public static void main(String [] args) {
        SharedBoard board = new SharedBoard();
        Thread th1 = new StudentThread("kitae", board);
        Thread th2 = new StudentThread("hyosoo", board);
        th1.start();
        th2.start();
    }
}

class SharedBoard {
    private int sum = 0; // 집계판의 합
    synchronized public void add() {
        int n = sum;
        Thread.yield(); // 현재 실행 중인 스레드 양보
        n += 10; // 10 증가
        sum = n; // 증가한 값을 집계합에 기록
        System.out.println(Thread.currentThread().getName() + " : " + sum);
    }
    public int getSum() { return sum; }
}

class StudentThread extends Thread {
    private SharedBoard board; // 집계판의 주소
    public StudentThread(String name, SharedBoard board) {
        super(name);
        this.board = board;
    }

    @Override
    public void run() {
        for(int i=0; i<10; i++)
            board.add();
    }
}
```

kitae : 10
hyosoo : 20
kitae : 30
hyosoo : 30
kitae : 40
hyosoo : 50
kitae : 50
hyosoo : 60
kitae : 60
hyosoo : 60
kitae : 70
hyosoo : 70
kitae : 80
hyosoo : 90
kitae : 100
hyosoo : 100
kitae : 110
hyosoo : 120
kitae : 130
hyosoo : 140
hyosoo : 150

kitae와 hyosoo가 각각 10번씩 add()를 호출하였지만 add()에 대한 동기화가 이루어지지 않아 공유 변수 sum을 kitae와 hyosoo가 각각 사용하여 누적 점수가 150 밖에 되지 못함

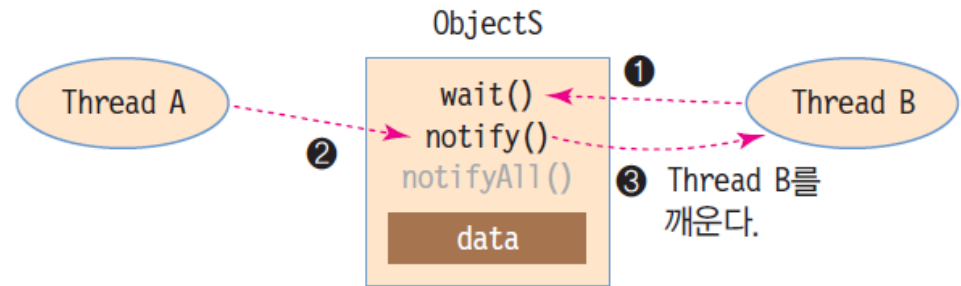
*컴퓨터가 빨라서 위와 같은 충돌 상황이 쉽게 벌어지지 않는 경우, StudentThread의 run()에서의 for문 반복 수를 100혹은 그 이상으로 시도

wait(), notify(), notifyAll()를 이용한 동기화

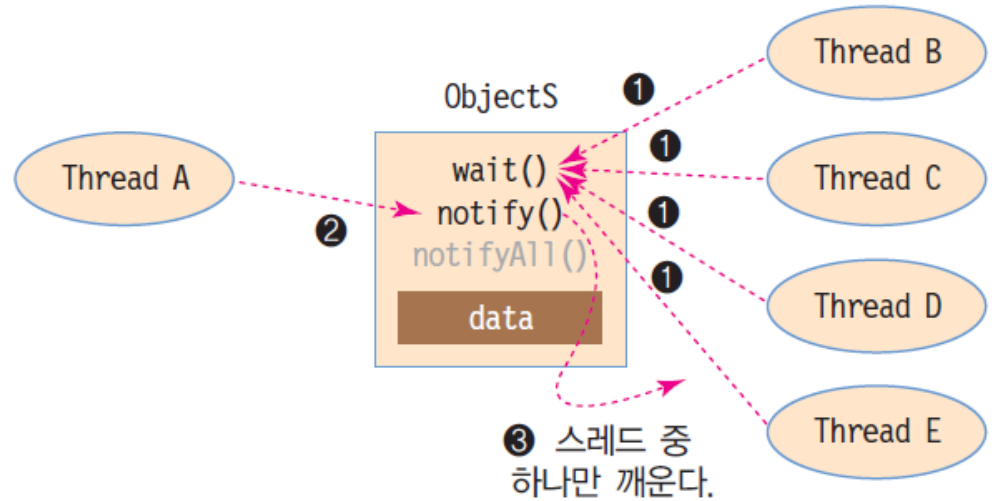
57

- 동기화 객체
 - ▣ 두 개 이상의 스레드 동기화에 사용되는 객체
- 동기화 메소드
 - ▣ wait()
 - 다른 스레드가 notify()를 불러줄 때까지 기다린다.
 - ▣ notify()
 - wait()를 호출하여 대기중인 스레드를 깨우고 RUNNABLE 상태로 만든다.
 - 2개 이상의 스레드가 대기중이라도 오직 한 스레드만 깨운다.
 - ▣ notifyAll()
 - wait()를 호출하여 대기중인 모든 스레드를 깨우고 모두 RUNNABLE 상태로 만든다.
 - ▣ synchronized 블록 내에서만 사용되어야 함
- wait(), notify(), notifyAll()은 Object의 메소드
 - ▣ 모든 객체가 동기화 객체가 될 수 있다.
 - ▣ Thread 객체도 동기화 객체로 사용될 수 있다.

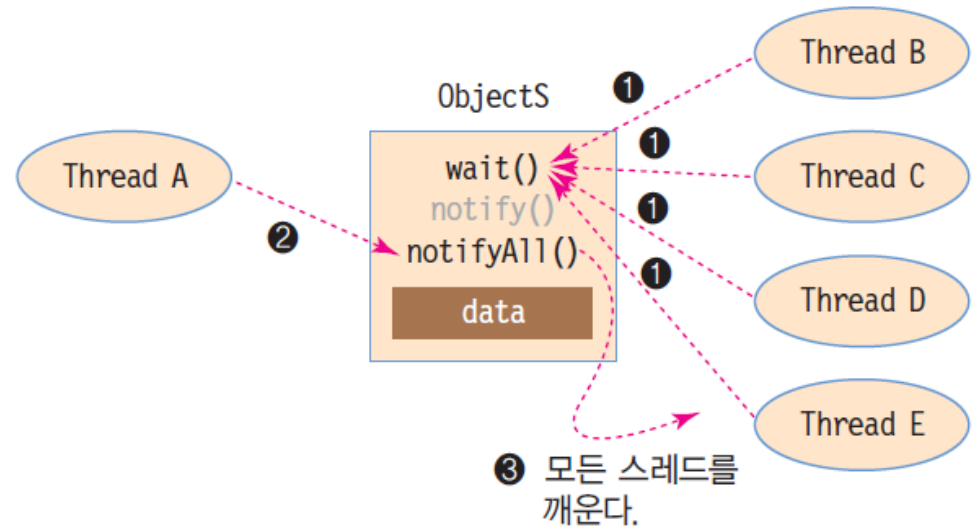
Thread A가 ObjectS.wait()를 호출하여 무한 대기하고, Thread B가 ObjectS.notify()를 호출하여 ObjectS에 대기하고 있는 Thread A를 깨운다.



4 개의 스레드가 모두 ObjectS.wait()를 호출하여 대기하고, ThreadA는 ObjectS.notify()를 호출하여 대기 중인 스레드 중 하나만 깨우는 경우



4 개의 스레드가 모두 ObjectS.wait()를 호출하여 대기하고, ThreadA는 ObjectS.notifyAll()를 호출하여 대기중인 4개의 스레드를 모두 깨우는 경우



예제 13-6 : wait(), notify()를 이용한 바 채우기

59

```
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);

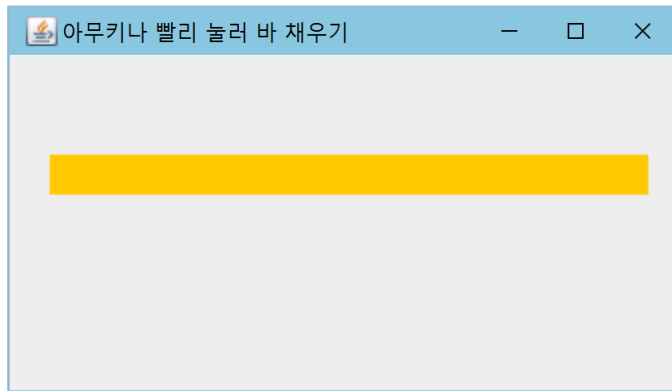
        c.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e)
            {
                bar.fill();
            }
        });
        setSize(350,200);
        setVisible(true);

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

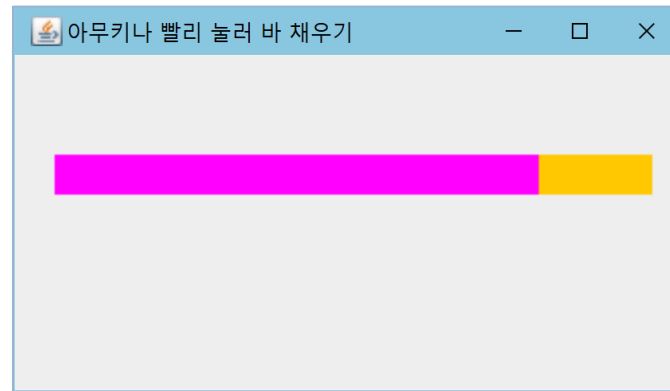
    public static void main(String[] args) {
        new TabAndThreadEx(
            "아무키나 빨리 눌러 바 채우기");
    }
}
```

실행 결과

60



초기 화면



키를 반복하여 빨리 누른 화면