

Networking and Threads: Make a Connection [2]

Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hankyong.ac.kr>

sleep을 사용하여 프로그램을 더 예측 가능하게!

```
public class MyRunnable implements Runnable {  
  
    public void run() {  
        go();  
    }  
  
    public void go() {  
  
        try {  
            Thread.sleep(2000);  
        } catch (InterruptedException ex) {  
            ex.printStackTrace();  
        }  
  
        doMore();  
    }  
  
    public void doMore() {  
        System.out.println("top o' the stack");  
    }  
}  
  
class ThreadTestDrive {  
    public static void main (String[] args) {  
        Runnable theJob = new MyRunnable();  
        Thread t = new Thread(theJob);  
        t.start();  
        System.out.println("back in main");  
    }  
}
```

← Calling sleep here will force the new thread to leave the currently-running state!

The main thread will become the currently-running thread again, and print out "back in main". Then there will be a pause (for about two seconds) before we get to this line, which calls doMore() and prints out "top o' the stack"

```
File Edit Window Help SnoozeButton  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack  
% java ThreadTestDrive  
back in main  
top o' the stack
```

2개의 쓰레드를 만들고 시작시키는 예제

```
public class RunThreads implements Runnable {
```

```
    public static void main(String[] args) {  
        RunThreads runner = new RunThreads();  
        Thread alpha = new Thread(runner);  
        Thread beta = new Thread(runner);  
        alpha.setName("Alpha thread");  
        beta.setName("Beta thread");  
        alpha.start();  
        beta.start();  
    }
```

← Make one Runnable instance.

← Make two threads, with the same Runnable (the same job—we'll talk more about the "two threads and one Runnable" in a few pages).

← Name the threads.

← Start the threads.

```
    public void run() {  
        for (int i = 0; i < 25; i++) {  
            String threadName = Thread.currentThread().getName();  
            System.out.println(threadName + " is running");  
        }  
    }  
}
```

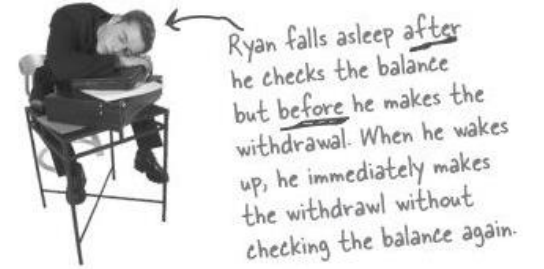
Each thread will run through this loop, printing its name each time.

```
File Edit Window Help Centauri  
Alpha thread is running  
Alpha thread is running  
Alpha thread is running  
Beta thread is running  
Alpha thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Beta thread is running  
Alpha thread is running
```

Wow! Threads are
the greatest thing since the
MINI Cooper! I can't think
of a single downside to using
threads, can you?



Um, yes. There IS a dark side



Threads can lead to concurrency 'issues'

- ✓ 병행성 문제는 **경쟁 상태**(race conditions)가 될 수 있다.
 - 경쟁 상태 => 데이터 손상(data corruption) => ...
- ✓ 죽음의 시나리오:
 - 두 개 이상의 스레드가 한 개의 객체 데이터에 접근하는 경우

각 스레드는 이 세상에 오로지 자기 자신만이 존재하는 걸로 착각하고서 동작한다!!

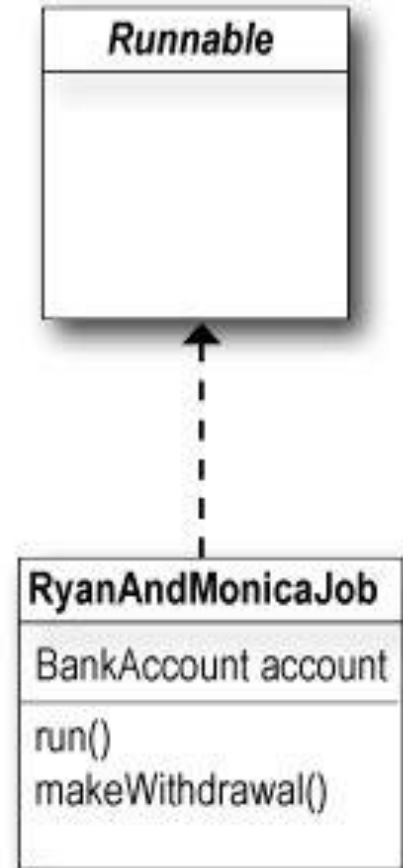
- ✓ 바로 이것이 문제를 야기시킨다.
 - 한 스레드가 '**Runnable**' 상태일 때는 **무의식 상태**라고 할 수 있다
 - 스레드가 다시 '**Running**' 상태로 되었을 때 자신이 전에 중단되었던 사실을 전혀 알지 못한다

Ryan and Monica example

두 개의 스레드(라이언과 모니카)가 하나의 객체(은행 계좌)를 공유할 때 생길 수 있는 문제

- ✓ 코드에는 BankAccount와 MonicaAndRyanJob 이라는 두 개의 클래스가 있다.
- ✓ MonicaAndRyanJob 클래스에서는 Runnable을 구현한다.
- ✓ Runnable 객체 (라이언과 모니카 둘 다 수행해야 할 작업):
 “잔액을 확인한 후 인출한다.”
- ✓ 제한조건:
 “각 스레드는 잔액을 확인하는 일과 인출하는 일 사이에 반드시 한 번 잠이 들어야 한다.”
- ✓ MonicaAndRyanJob 클래스는 **BankAccount** 인스턴스 변수(공동 계좌)를 가진다.

1. RyanAndMonicaJob의 인스턴스 한 개를 만든다
`RyanAndMonicaJob theJob = new RyanAndMonicaJob();`
2. 같은 Runnable 객체를 가지는 두 개의 쓰레드를 만든다
`Thread one = new Thread(theJob);`
`Thread two = new Thread(theJob);`
3. 쓰레드에 이름을 부여하고 시작시킨다
`one.setName("Ryan");`
`two.setName("Monica");`
`one.start();`
`two.start();`
4. 두 개의 쓰레드가 run() 메소드를 실행시키는걸 지켜본다
(두 쓰레드 모두 계좌 잔액을 확인한 후 인출)



run() 메소드에서 라이언과 모니카가 정확히 해야 할 일:

잔액 확인 후 잔액이 충분히 남아있다면 돈을 인출한다.

⇒ 이렇게 하면 계좌에서 초과 인출하는 경우를 막아줄 것이다

제한조건:

라이언과 모니카는 각자 잔액을 확인한 후 돈을 인출하기 전에 항상 잠이 들어야 한다.

```
if (account.getBalance() >= amount) {  
    try {  
        Thread.sleep(500);  
    } catch (InterruptedException ex) {ex.printStackTrace();}  
}
```


실습과제 16-1 Ryan and Monica example

```
class BankAccount {  
    private int balance = 100;   
  
    public int getBalance() {  
        return balance;  
    }  
  
    public void withdraw(int amount) {  
        balance = balance - amount;  
    }  
}
```

← The account starts with a balance of \$100.

```
public class RyanAndMonicaJob implements Runnable {
```

```
    private BankAccount account = new BankAccount();
```

← There will be only ONE instance of the RyanAndMonicaJob. That means only ONE instance of the bank account. Both threads will access this one account.

```
    public static void main (String [] args) {
```

```
        RyanAndMonicaJob theJob = new RyanAndMonicaJob();
```

```
        Thread one = new Thread(theJob);
```

```
        Thread two = new Thread(theJob);
```

```
        one.setName("Ryan");
```

```
        two.setName("Monica");
```

```
        one.start();
```

```
        two.start();
```

```
    }
```

← Instantiate the Runnable (job)
← Make two threads, giving each thread the same Runnable job. That means both threads will be accessing the one account instance variable in the Runnable class.

```

public void run() {
    for (int x = 0; x < 10; x++) {
        makeWithdrawal(10);
        if (account.getBalance() < 0) {
            System.out.println("Overdrawn!");
        }
    }
}

```

In the run() method, a thread loops through and tries to make a withdrawal with each iteration. After the withdrawal, it checks the balance once again to see if the account is overdrawn.

```

private void makeWithdrawal(int amount) {
    if (account.getBalance() >= amount) {
        System.out.println(Thread.currentThread().getName() + " is about to withdraw");
        try {
            System.out.println(Thread.currentThread().getName() + " is going to sleep");
            Thread.sleep(500);
        } catch (InterruptedException ex) {ex.printStackTrace(); }
        System.out.println(Thread.currentThread().getName() + " woke up.");
        account.withdraw(amount);
        System.out.println(Thread.currentThread().getName() + " completes the withdrawal");
    }
    else {
        System.out.println("Sorry, not enough for " + Thread.currentThread().getName());
    }
}

```

잔액 확인 => 잔액이 충분하지 않으면 그냥 메시
지를 프린트한다 => 잔액이 충분하면 잠이 든다 =>
깨어나서 인출한다

We put in a bunch of print statements so we can see what's happening as it runs.

실행 화면

How did this happen?



```
Run: RyanAndMonicaJob x
Ryan woke up
Ryan completes the withdrawal
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up
Monica completes the withdrawal
Sorry, not enough for Monica
Sorry, not enough for Monica
Sorry, not enough for Monica
Sorry, not enough for Monica
Sorry, not enough for Monica
Ryan woke up
Ryan completes the withdrawal
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Process finished with exit code 0
```

Ryan and Monica example : 가능한 시나리오

makeWithdrawal() 메소드에서 항상 잔액을 확인하고 인출을 수행하지만 여전히 **초과 인출**하는 상황이 발생할 수 있다.

가능한 시나리오:

Ryan이 잔액을 확인하고 잔액이 남아있는걸 보고 잠이 든다.

Ryan이 잠들어 있는 동안 Monica가 들어와서 잔액을 확인한다. 그녀 또한 돈이 충분히 남아있음을 확인한다. 이 시점에서 그녀는 Ryan이 잠에서 깨어나서 돈을 인출하리라는 것을 전혀 예상하지 못한다.

Monica가 잠이 든다.

Ryan이 깨어나서 돈을 인출한다.

Monica도 깨어나서 돈을 인출한다. 큰 문제가 발생했다!! Monica가 잔액을 확인하고 인출하는 사이에 Ryan이 깨어나서 계좌에서 돈을 인출해버린 것이다.

여기에서 Monica가 잔액을 확인했던 일은 사실 무의미하다. 왜냐하면 Ryan이 잔액을 이미 확인했고 아직 돈을 인출하는 과정 중에 있기 때문에.

따라서 **Ryan이 깨어나서 그의 거래가 완료될 때까지 Monica가 공동계좌에 접근할 수 없게 만들어야 한다.**

그 역도 마찬가지!

They need a lock for account access!

락(lock)은 다음처럼 동작한다:

1. 은행 계좌 트랜잭션(잔액 확인 후 인출)과 관련된 락이 있다.

키는 하나밖에 없다.
2. Ryan이 계좌에 접근할 때 락을 잠그고 키를 가져간다.
3. Ryan은 트랜잭션을 완료할 때까지 키를 호주머니에 넣어둔다.

Ryan이 유일한 키를 가지고 있다. 따라서 Ryan이 계좌의 락을 열고 키를 반납할 때까지 Monica는 계좌에 접근할 수가 없다!!



은행 계좌 트랜잭션은 아무도 계좌를 사용하고 있지 않을 때는 해제되어 있다.




Ryan이 계좌에 접근하고 싶을 때 락을 잠그고 키를 가져간다.



Ryan이 일을 마치면 락을 열고 키를 반납한다. 이제 모니카가 키를 이용할 수 있다.

We need the makeWithdrawal() method to run as one atomic thing

어떤 메소드를 한 번에 한 스레드만 접근할 수 있게 만들고 싶으면 **synchronized**라는 키워드를 사용하면 된다!



```
private synchronized void makeWithdrawal(int amount) {  
    if (account.getBalance() >= amount) {  
        System.out.println(Thread.currentThread().getName() + " is about to withdraw");  
        try {  
            System.out.println(Thread.currentThread().getName() + " is going to sleep");  
            Thread.sleep(500);  
        } catch (InterruptedException ex) {ex.printStackTrace(); }  
        System.out.println(Thread.currentThread().getName() + " woke up.");  
        account.withdraw(amount);  
        System.out.println(Thread.currentThread().getName() + " completes the withdrawl");  
    } else {  
        System.out.println("Sorry, not enough for " + Thread.currentThread().getName());  
    }  
}
```

Using an object's lock

모든 자바 객체는 하나의 락(lock)을 가지고 있고 대부분 시간 동안 락은 열려 있다.

객체 락은 **synchronized** 메소드가 있을 때만 동작한다.

어떤 객체가 하나 이상의 **synchronized** 메소드를 가질 때 그 객체 락에 대한 키를 얻을 수 있는 스레드만이 **synchronized** 메소드에 들어갈 수가 있다!

락은 메소드당 하나씩 있는 것이 아니라 객체당 하나씩 있다.

⇒ 만일 한 객체가 두 개의 **synchronized** 메소드를 가지고 있다면 어떤 **synchronized** 메소드에도 두 스레드가 동시에 들어갈 수 없다.



The dreaded “Lost Update” problem

데이터베이스 분야에서 나온 고전적인 “Lost Update” 문제를 살펴보자.

“**Lost Update**” 문제는 하나의 프로세스를 중심으로 발생한다:

Step 1. 계좌에서 잔액을 가져온다

int i = balance;

Step 2. 잔액에 1을 더한다

balance = i + 1;

우리가 **balance++**라는 구문을 사용하더라도 이 문장이 “**원자적인 프로세스**”로 된다는 것을 보장해 주지는 않는다.

balance를 증가시키려는 두 개의 스레드가 있다고 하자

```
class TestSync implements Runnable {
```

```
    private int balance;
```

```
    public void run() {
```

```
        for(int i = 0; i < 50; i++) {
```

```
            increment();
```

```
            System.out.println("balance is " + balance);
```

```
        }
```

```
    }
```

```
    public void increment() {
```

```
        int i = balance;
```

```
        balance = i + 1;
```

```
    }
```

```
}
```

각 스레드가 50회 실행된다.
반복될 때마다 잔액이 1씩 증
가된다

balance의 값이 그 값을 읽은 시점에서 어떤 값이든
무조건 balance에 1을 증가시킨다. (balance의 현재
값에 1을 증가시키는 것이 아니라)

```
public class TestSyncTest {
```

```
    public static void main (String[] args) {
```

```
        TestSync job = new TestSync();
```

```
        Thread a = new Thread(job);
```

```
        Thread b = new Thread(job);
```

```
        a.start();
```

```
        b.start();
```

```
    }
```

```
}
```

Let's run this code...

① Thread A runs for awhile



Put the value of balance into variable i.
Balance is 0, so i is now 0.
Set the value of balance to the result of $i + 1$.
Now balance is 1.
Put the value of balance into variable i.
Balance is 1, so i is now 1.
Set the value of balance to the result of $i + 1$.
Now balance is 2.

② Thread B runs for awhile



Put the value of balance into variable i.
Balance is 2, so i is now 2.
Set the value of balance to the result of $i + 1$.
Now balance is 3.
Put the value of balance into variable i.
Balance is 3, so i is now 3.

*[now thread B is sent back to runnable,
before it sets the value of balance to 4]*

③ Thread A runs again, picking up where it left off



Put the value of balance into variable i.
Balance is 3, so i is now 3.
Set the value of balance to the result of $i + 1$.
Now balance is 4.
Put the value of balance into variable i.
Balance is 4, so i is now 4.
Set the value of balance to the result of $i + 1$.
Now balance is 5.

④ Thread B runs again, and picks up exactly where it left off!



Set the value of balance to the result of $i + 1$.
Now balance is 4.

쓰레드 A가 수행했던 마지막
업데이트 내용을 잃었다!

Make the increment() method atomic. Synchronize it!

increment() 메소드를 동기화 (synchronized)시키면 "Lost Update" 문제를 해결할 수 있다.

⇒ 메소드에 있는 두 단계를 더 이상 쪼갤 수 없는 원자 단위 (트랜잭션)로 묶어버리기 때문에.

```
public synchronized void increment() {  
    int i = balance;  
    balance = i + 1;  
}
```

NOTE

Once a thread enters the method, we have to make sure that all the steps in the method complete (as one atomic process) before any other thread can enter the method.

The deadly side of synchronization

Synchronized 코드를 사용할 때는 스레드 **교착상태** (thread deadlock)에 빠질 우려가 있으므로 특별히 주의해야 한다.

스레드 교착상태는 두 개의 스레드가 서로 상대방이 원하는 키를 가지고 있을 때 발생하게 된다.

스레드가 이런 상태에 빠지게 되면 이를 벗어날 방법이 없다.

⇒ 두 스레드는 그냥 앉아서 기다리는 수밖에 없다! 그리고 기다리고 또...

대부분 데이터베이스 관리 시스템은 동기화와 유사한 잠금장치를 가지고 있다.

⇒ 실제 트랜잭션 관리 시스템은 대부분의 경우 **교착상태를 처리하는 메커니즘**을 가지고 있다

⇒ 예를 들어, 애플리케이션 서버에서 두 트랜잭션이 너무 오랫동안 완료되지 않으면 이들이 교착상태에 빠진 것으로 간주할 수 있다

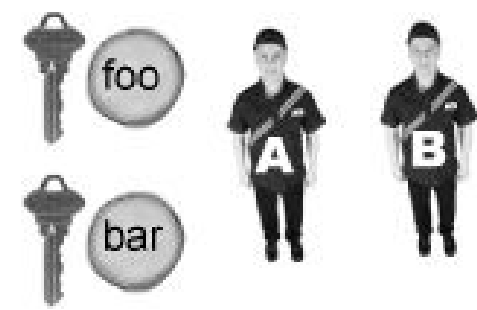
⇒ 그리하여 트랜잭션이 시작되기 전 상태로 "**트랜잭션 롤백**"을 수행할 수 있다

Java에서는 교착상태를 조절할 어떠한 메커니즘도 없다!

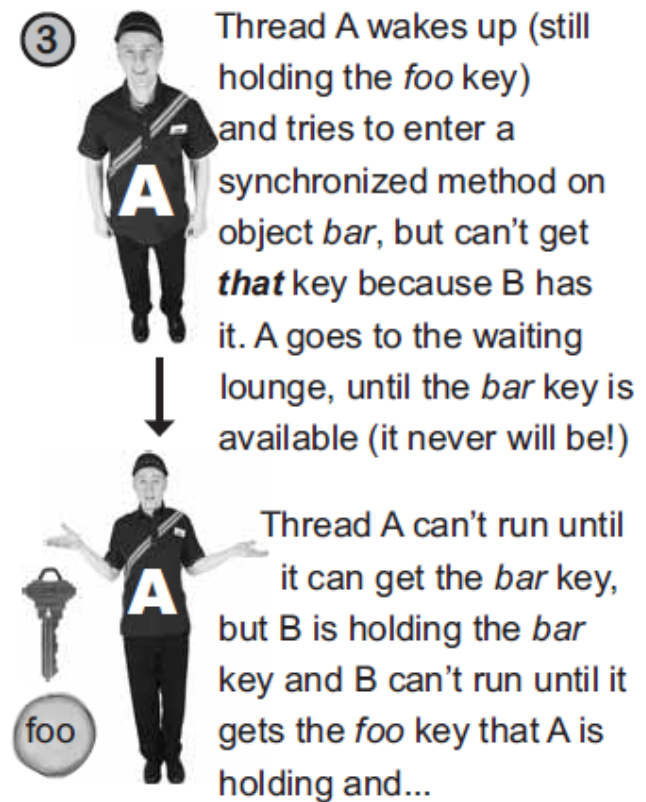
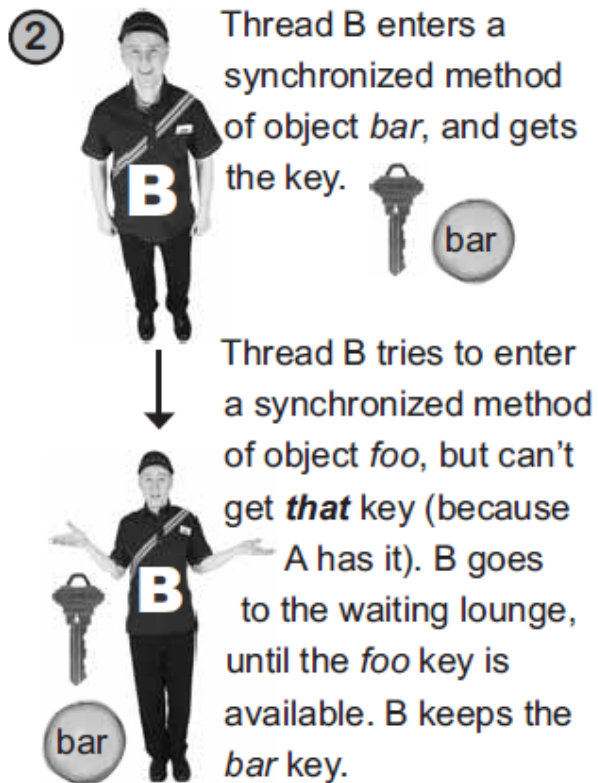
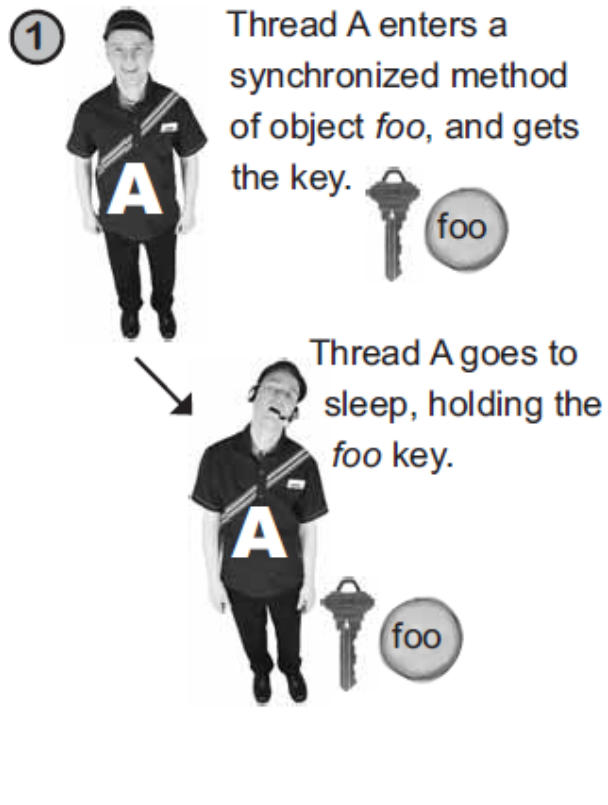
심지어 교착상태가 발생했는지조차도 모른다!

그래서 프로그래머가 주의깊게 설계하는 수밖에 없다.

A simple deadlock scenario



두 개의 Synchronized 객체와 두 개의 쓰레드가 있으면 교착상태에 빠질 수 있다.



실습과제 16-2

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleChatClient {

    JTextArea incoming;
    JTextField outgoing;
    BufferedReader reader;
    PrintWriter writer;
    Socket sock;

    public static void main(String[] args) {
        SimpleChatClient client = new SimpleChatClient();
        client.go();
    }

    public void go() {

        JFrame frame = new JFrame("Ludicrously Simple Chat Client");
        JPanel mainPanel = new JPanel();
        incoming = new JTextArea(15, 50);
        incoming.setLineWrap(true);
        incoming.setWrapStyleWord(true);
        incoming.setEditable(false);
        JScrollPane qScroller = new JScrollPane(incoming);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        outgoing = new JTextField(20);
        JButton sendButton = new JButton("Send");
        sendButton.addActionListener(new SendButtonListener());
        mainPanel.add(qScroller);
        mainPanel.add(outgoing);
        mainPanel.add(sendButton);
        setUpNetworking();

        Thread readerThread = new Thread(new IncomingReader());
        readerThread.start();

        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(400, 500);
        frame.setVisible(true);

    } // close go
}
```

This is mostly GUI code you've seen before. Nothing special except the highlighted part where we start the new 'reader' thread.

We're starting a new thread, using a new inner class as the Runnable (job) for the thread. The thread's job is to read from the server's socket stream, displaying any incoming messages in the scrolling text area.


```
private void setUpNetworking() {
```

```
    try {  
        sock = new Socket("127.0.0.1", 5000);  
        InputStreamReader streamReader = new InputStreamReader(sock.getInputStream());  
        reader = new BufferedReader(streamReader);  
        writer = new PrintWriter(sock.getOutputStream());  
        System.out.println("networking established");  
    } catch(IOException ex) {  
        ex.printStackTrace();  
    }  
} // close setUpNetworking
```

We're using the socket to get the input and output streams. We were already using the output stream to send to the server, but now we're using the input stream so that the new 'reader' thread can get messages from the server.

```
public class SendButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent ev) {  
        try {  
            writer.println(outgoing.getText());  
            writer.flush();  
  
        } catch(Exception ex) {  
            ex.printStackTrace();  
        }  
        outgoing.setText("");  
        outgoing.requestFocus();  
    }  
} // close inner class
```

Nothing new here. When the user clicks the send button, this method sends the contents of the text field to the server.

```
public class IncomingReader implements Runnable {  
    public void run() {  
        String message;  
        try {  
  
            while ((message = reader.readLine()) != null) {  
                System.out.println("read " + message);  
                incoming.append(message + "\n");  
  
            } // close while  
        } catch(Exception ex) {ex.printStackTrace();}  
    } // close run  
} // close inner class
```

This is what the thread does!

In the run() method, it stays in a loop (as long as what it gets from the server is not null), reading a line at a time and adding each line to the scrolling text area (along with a new line character).

```
} // close outer class
```


VerySimpleChatServer.java

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

public class VerySimpleChatServer {

    ArrayList clientOutputStreams;

    public class ClientHandler implements Runnable {
        BufferedReader reader;
        Socket sock;

        public ClientHandler(Socket clientSocket) {
            try {
                sock = clientSocket;
                InputStreamReader isReader = new InputStreamReader(sock.getInputStream());
                reader = new BufferedReader(isReader);

                } catch (Exception ex) {ex.printStackTrace();}
            } // close constructor

        public void run() {
            String message;
            try {
                while ((message = reader.readLine()) != null) {
                    System.out.println("read " + message);
                    tellEveryone(message);

                } // close while
            } catch (Exception ex) {ex.printStackTrace();}
        } // close run
    } // close inner class
}
```

To run the chat client, you need two terminals. First, launch this server from one terminal, then launch the client from another terminal

```

public static void main (String[] args) {
    new VerySimpleChatServer().go();
}

public void go() {
    clientOutputStreams = new ArrayList();
    try {
        ServerSocket serverSock = new ServerSocket(5000);

        while(true) {
            Socket clientSocket = serverSock.accept();
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
            clientOutputStreams.add(writer);

            Thread t = new Thread(new ClientHandler(clientSocket));
            t.start();
            System.out.println("got a connection");
        }

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    } // close go

public void tellEveryone(String message) {

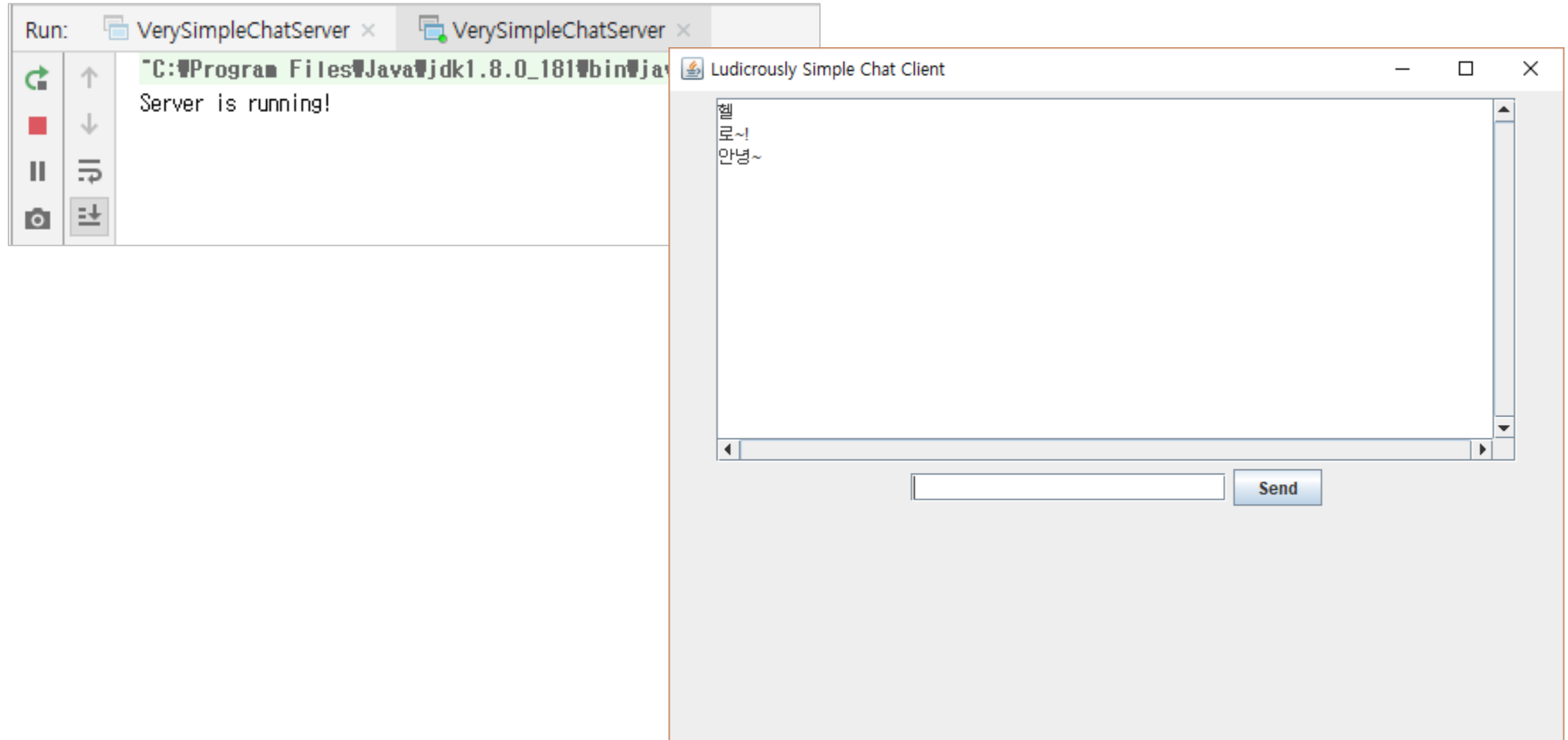
    Iterator it = clientOutputStreams.iterator();
    while(it.hasNext()) {
        try {
            PrintWriter writer = (PrintWriter) it.next();
            writer.println(message);
            writer.flush();
        } catch (Exception ex) {
            ex.printStackTrace();
        }

    } // end while

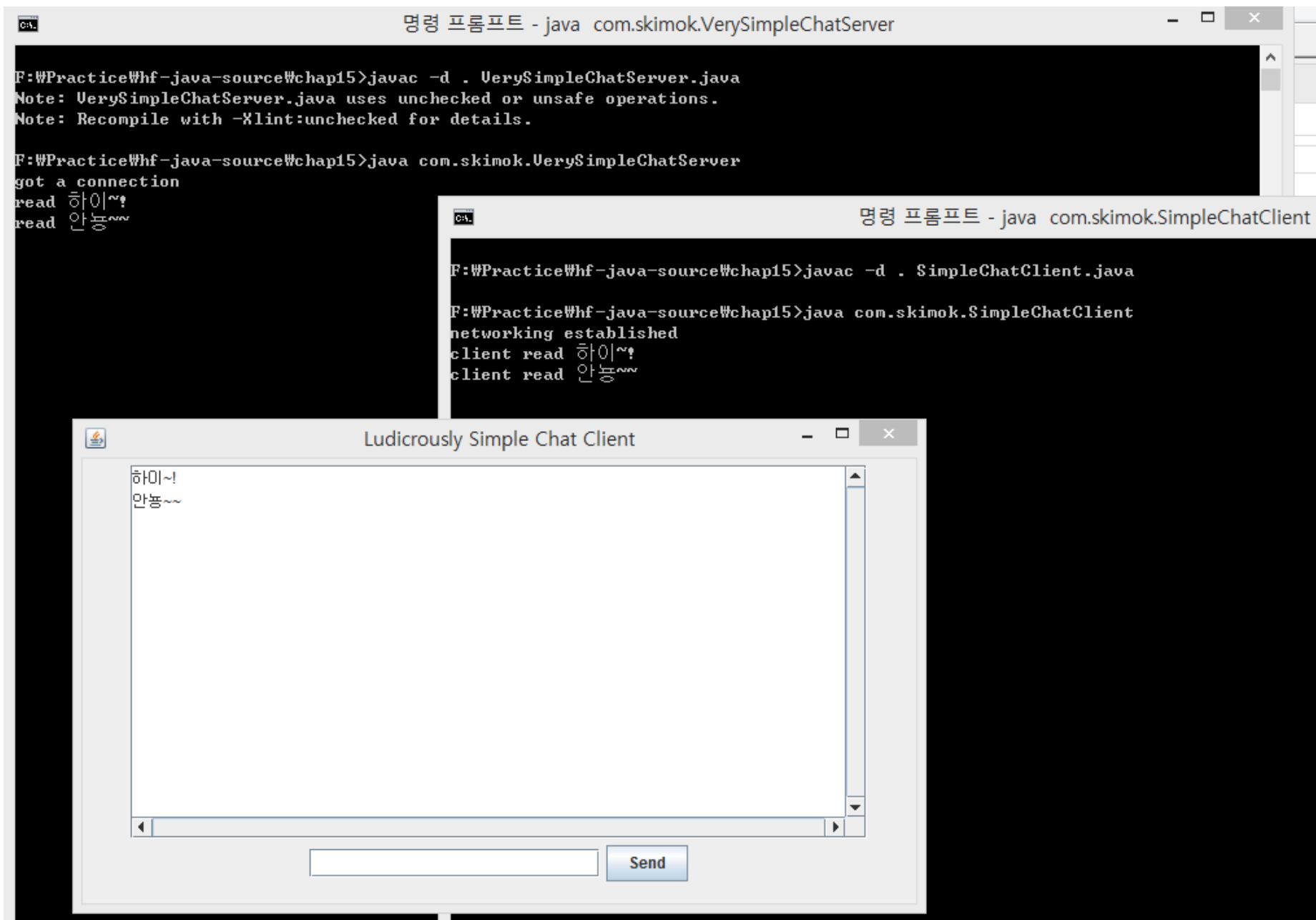
    } // close tellEveryone
} // close class

```

실행 화면



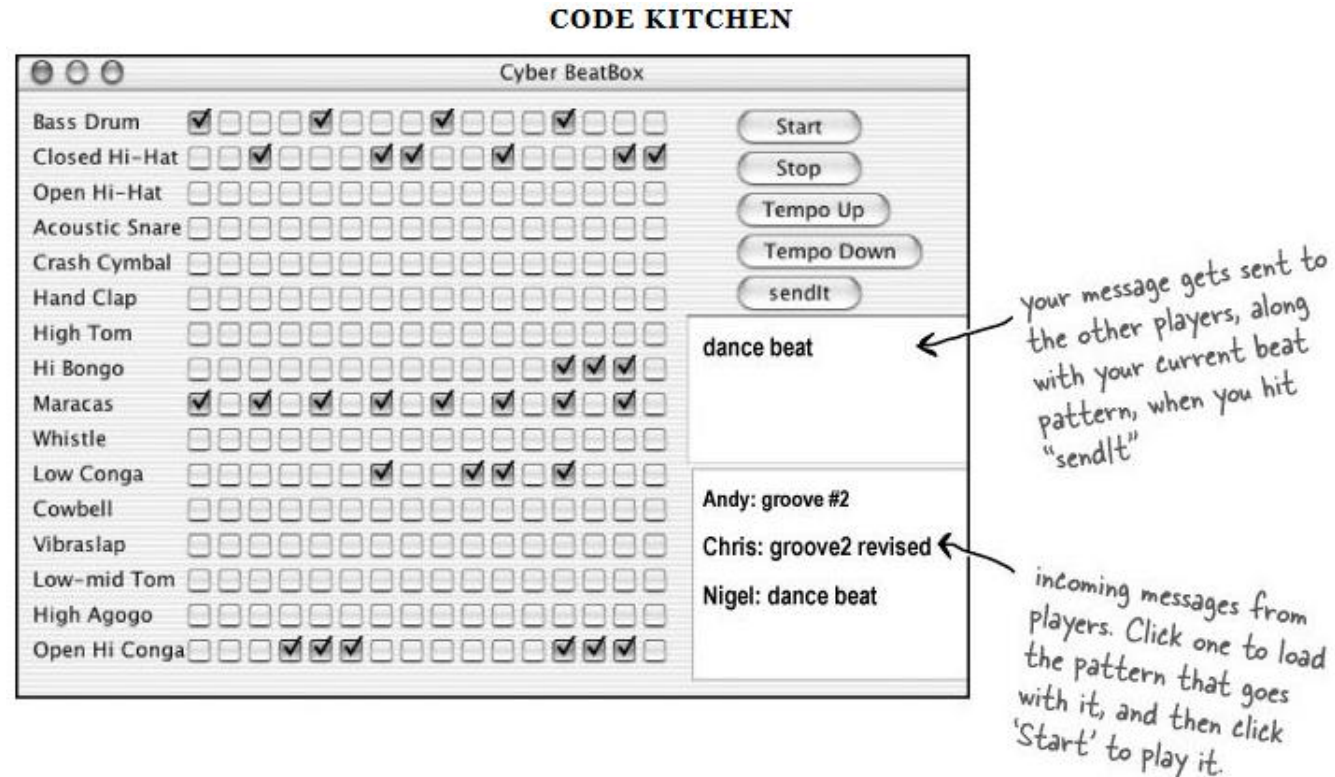
실행 화면



Code Kitchen

실습과제 16-3 BeatBoxFinal

Connecting the BeatBox to a simple MusicServer



This is the last version of the BeatBox! It connects to a simple MusicServer so that you can send and receive beat patterns with other clients.

BeatBoxFinal.java

```
1  package com.skimok;
2
3  import java.awt.*;
4  import javax.swing.*;
5  import javax.swing.event.ListSelectionEvent;
6  import javax.swing.event.ListSelectionListener;
7
8  import java.io.*;
9  import javax.sound.midi.*;
10 import java.util.*;
11 import java.awt.event.*;
12 import java.net.*;
13
14
15 public class BeatBoxFinal { // implements MetaEventListener
16
17     JPanel mainPanel;
18     JList incomingList;
19     JTextField userMessage;
20     ArrayList<JCheckBox> checkboxList;
21     int nextNum;
22     ObjectInputStream in;
23     ObjectOutputStream out;
24     Vector<String> listVector = new Vector<String>();
25     String userName ;
26     HashMap<String, boolean[]> otherSeqsMap = new HashMap<String, boolean[]>();
27     Sequencer sequencer;
28     Sequence sequence;
29     Sequence mySequence = null;
30     Track track;
31     JFrame theFrame;
32
33     String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat",
34                                "Open Hi-Hat", "Acoustic Snare", "Crash Cymbal", "Hand Clap",
```

```

35     "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga",
36     "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo",
37     "Open Hi Conga");
38     int[] instruments = {35,42,46,38,49,39,50,60,70,72,64,56,58,47,67,63};
39
40
41     public static void main (String[] args) {
42         new BeatBoxFinal().startUp(args[0]);
43     }
44
45     public void startUp(String name) {
46         userName = name;
47         try {
48             Socket sock = new Socket("220.68.240.100", 4242);
49             out = new ObjectOutputStream(sock.getOutputStream());
50             in = new ObjectInputStream(sock.getInputStream());
51             Thread remote = new Thread(new RemoteReader());
52             remote.start();
53         }
54         catch (Exception ex) {
55             System.out.println("couldn't connect - you'll have to play alone.");
56         }
57         setUpMidi();
58         buildGUI();
59     }
60
61     public void buildGUI() {
62         theFrame = new JFrame("Cyber BeatBox");
63         theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64         BorderLayout layout = new BorderLayout();
65         JPanel background = new JPanel(layout);
66         background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
67
68         checkboxList = new ArrayList<JCheckBox>();
69         Box buttonBox = new Box(BoxLayout.Y_AXIS);
70
71         JButton start = new JButton("Start");
72         start.addActionListener(new MyStartListener());
73         buttonBox.add(start);
74
75
76

```



```

77 JButton upTempo = new JButton("Tempo Up");
78 upTempo.addActionListener(new MyUpTempoListener());
79 buttonBox.add(upTempo);
80
81 JButton downTempo = new JButton("Tempo Down");
82 downTempo.addActionListener(new MyDownTempoListener());
83 buttonBox.add(downTempo);
84
85 JButton sendIt = new JButton("sendIt");
86 sendIt.addActionListener(new MySendListener());
87 buttonBox.add(sendIt);
88
89 JButton saveIt = new JButton("Serialize It"); // new button
90 saveIt.addActionListener(new MySendListener());
91 buttonBox.add(saveIt);
92
93 JTextField userMessage = new JTextField();
94 buttonBox.add(userMessage);
95
96 JList incomingList = new JList();
97 incomingList.addListSelectionListener(new MyListSelectionListener());
98 incomingList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
99 JScrollPane theList = new JScrollPane(incomingList);
100 buttonBox.add(theList);
101 incomingList.setListData(listVector);
102
103 Box nameBox = new Box(BoxLayout.Y_AXIS);
104 for (int i = 0; i < 16; i++) {
105     nameBox.add(new Label(instrumentNames[i]));
106 }
107
108 background.add(BorderLayout.EAST, buttonBox);
109 background.add(BorderLayout.WEST, nameBox);
110
111 theFrame.getContentPane().add(background);
112
113 GridLayout grid = new GridLayout(16,16);
114 grid.setVgap(1);

```

```

115 grid.setHgap(2);
116 mainPanel = new JPanel(grid);
117 background.add(BorderLayout.CENTER, mainPanel);
118
119 for (int i = 0; i < 256; i++) {
120     JCheckBox c = new JCheckBox();
121     c.setSelected(false);
122     checkboxList.add(c);
123     mainPanel.add(c);
124 } // end loop
125
126 theFrame.setBounds(50,50,300,300);
127 theFrame.pack();
128 theFrame.setVisible(true);
129 } // close method
130
131
132 public void setUpMidi() {
133     try {
134         sequencer = MidiSystem.getSequencer();
135         sequencer.open();
136         // sequencer.addMetaEventListener(this);
137         sequence = new Sequence(Sequence.PPQ,4);
138         track = sequence.createTrack();
139         sequencer.setTempoInBPM(120);
140
141     } catch (Exception e) {e.printStackTrace();}
142 } // close method
143
144 public void buildTrackAndStart()
145 {
146     // this will hold the instruments for each vertical column,
147     // in other words, each tick (may have multiple instruments)
148     ArrayList<Integer> trackList = null;
149
150     sequence.deleteTrack(track);
151     track = sequence.createTrack();
152

```

```

153     for (int i = 0; i < 16; i++){
154         trackList = new ArrayList<Integer>();
155         for (int j = 0; j < 16; j++){
156             JCheckBox jc = (JCheckBox) checkboxList.get(j + (16 * i));
157             if (jc.isSelected()){
158                 int key = instruments[i];
159                 trackList.add(key);
160             }
161             else
162             {
163                 trackList.add(null);
164             }
165         } // close inner
166
167         makeTracks(trackList);
168     } // close outer
169
170     track.add(makeEvent(192,9,1,0,15)); // - so we always go to full 16 beats
171
172     try {
173
174         sequencer.setSequence(sequence);
175         sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
176         sequencer.start();
177         sequencer.setTempoInBPM(120);
178     } catch (Exception e) {e.printStackTrace();}
179
180 } // close method
181
182 //===== inner class listeners
183
184 public class MyStartListener implements ActionListener {
185     public void actionPerformed(ActionEvent a) {
186         buildTrackAndStart();
187     }
188 }
189
190 public class MyStopListener implements ActionListener {

```

```

191     public void actionPerformed(ActionEvent a) {
192         sequencer.stop();
193     }
194 }
195
196 public class MyUpTempoListener implements ActionListener {
197     public void actionPerformed(ActionEvent a) {
198         float tempoFactor = sequencer.getTempoFactor();
199         sequencer.setTempoFactor((float)(tempoFactor * 1.03));
200     }
201 }
202
203 public class MyDownTempoListener implements ActionListener {
204     public void actionPerformed(ActionEvent a) {
205         float tempoFactor = sequencer.getTempoFactor();
206         sequencer.setTempoFactor((float)(tempoFactor * .97));
207     }
208 }
209
210 public class MySendListener implements ActionListener { // new - save
211     public void actionPerformed(ActionEvent a) {
212         // make an arraylist of just the STATE of the checkboxes
213         boolean[] checkboxState = new boolean[256];
214
215         for (int i = 0; i < 256; i++) {
216             JCheckBox check = (JCheckBox) checkboxList.get(i);
217             if (check.isSelected()) {
218                 checkboxState[i] = true;
219             }
220         }
221
222         try {
223             out.writeObject(userName + nextNum++ + ": " + userMessage.getText());
224             out.writeObject(checkboxState);
225         } catch (Exception ex) {
226             ex.printStackTrace();
227             System.out.println("sorry dude. Could not send it to the server");
228         }

```

```

229
230     } // close method
231 } // close inner class
232
233 public class MyListSelectionListener implements ListSelectionListener {
234     public void valueChanged(ListSelectionEvent le) {
235         if (!le.getValueIsAdjusting()) {
236             String selected = (String) incomingList.getSelectedValue();
237             if (selected != null) {
238                 boolean[] selectedState = (boolean[]) otherSeqsMap.get(selected);
239                 changeSequence(selectedState);
240                 sequencer.stop();
241                 buildTrackAndStart();
242             }
243         }
244     }
245 }
246
247 public class RemoteReader implements Runnable {
248     boolean[] checkboxState = null;
249     String nameToShow = null;
250     Object obj = null;
251
252     public void run() {
253         try {
254             while ((obj=in.readObject()) != null) {
255                 System.out.println("got an object from server");
256                 System.out.println(obj.getClass());
257                 String nameToShow = (String) obj;
258                 checkboxState = (boolean[]) in.readObject();
259                 otherSeqsMap.put(nameToShow, checkboxState);
260                 listVector.add(nameToShow);
261                 incomingList.setListData(listVector);
262             }
263         } catch (Exception e) { e.printStackTrace(); }
264     }
265 }
266
267 //=====
268
269 public void changeSequence(boolean[] checkboxState) {
270     for (int i = 0; i < 256; i++) {
271         JCheckBox check = (JCheckBox) checkboxList.get(i);
272         if (checkboxState[i]) {
273             check.setSelected(true);
274         }
275         else
276         {
277             check.setSelected(false);
278         }
279     }
280 }
281
282 public void makeTracks(ArrayList<Integer> list) {
283     Iterator it = list.iterator();
284     for (int i = 0; i < 16; i++) {
285         Integer num = (Integer) it.next();
286         if (num != null) {
287             int numKey = num.intValue();
288             track.add(makeEvent(144, 9, numKey, 100, i));
289             track.add(makeEvent(128, 9, numKey, 100, i+1));
290         }
291     }
292 }
293
294 public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
295     MidiEvent event = null;
296     try {
297         ShortMessage a = new ShortMessage();
298         a.setMessage(comd, chan, one, two);
299         event = new MidiEvent(a, tick);
300     }
301     catch(Exception e) {}
302     return event;
303 }
304 }

```

MusicServer.java

```
1  package com.skimok;
2
3  import java.io.*;
4  import java.net.*;
5  import java.util.*;
6
7
8  public class MusicServer
9  {
10     ArrayList clientOutputStreams;
11
12     public static void main(String[] args) {
13         new MusicServer().go();
14     }
15
16     public class ClientHandler implements Runnable {
17         ObjectInputStream in;
18         Socket sock;
19
20         public ClientHandler(Socket clientSocket) {
21             try {
22                 sock = clientSocket;
23                 in = new ObjectInputStream(sock.getInputStream());
24
25             } catch (Exception ex) { ex.printStackTrace(); }
26         }
27
28         public void run() {
29             Object o1;
30             Object o2;
31             try {
32                 while ((o1 = in.readObject()) != null) {
33                     o2 = in.readObject();
34                     System.out.println("read two objects -- 홍길동 서버");
```

```

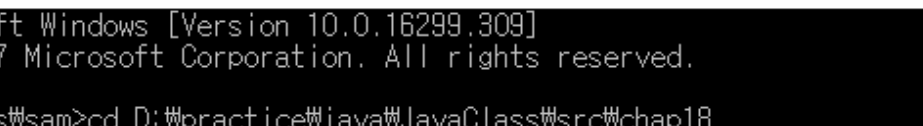
35         tellEveryone(o1, o2);
36     }
37     } catch (Exception ex) { ex.printStackTrace(); }
38 }
39 }
40
41
42 public void go() {
43     clientOutputStreams = new ArrayList();
44     try {
45         ServerSocket serverSock = new ServerSocket(4242);
46         while(true) {
47             Socket clientSocket = serverSock.accept();
48             ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
49             clientOutputStreams.add(out);
50
51             Thread t = new Thread(new ClientHandler(clientSocket));
52             t.start();
53             System.out.println("got a connection -- 홍길동 서버");
54         }
55     } catch (Exception ex) { ex.printStackTrace(); }
56 }
57
58 public void tellEveryone(Object one, Object two) {
59     Iterator it = clientOutputStreams.iterator();
60     while (it.hasNext()) {
61         try {
62             ObjectOutputStream out = (ObjectOutputStream) it.next();
63             out.writeObject(one);
64             out.writeObject(two);
65         } catch (Exception ex) { ex.printStackTrace(); }
66     }
67 }
68 }

```

실습과제 16-4

C:\ 명령 프롬프트 - java chap18.MusicServer

```
D:\practice\java\JavaClass\src\chap18>java chap18.MusicServer
```



```
C:\>명령 프롬프트 - java chap18.BeatBoxFinal sam
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\ssam>cd D:\practice\java\JavaClass\src\chap18

C:\Users\ssam>d:

D:\practice\java\JavaClass\src\chap18>java chap18.BeatBoxFinal
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at chap18.BeatBoxFinal.main(BeatBoxFinal.java:42)

D:\practice\java\JavaClass\src\chap18>java chap18.BeatBoxFinal sam
```

Cyber BeatBox

Bass Drum

Closed Hi-Hat

Open Hi-Hat

Acoustic Snare

Crash Cymbal

Hand Clap

High Tom

Hi Bongo

Maracas

Whistle

Low Conga

Cowbell

Vibraslap

Low-mid Tom

High Agogo

Open Hi Conga

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Start

Stop

Tempo Up

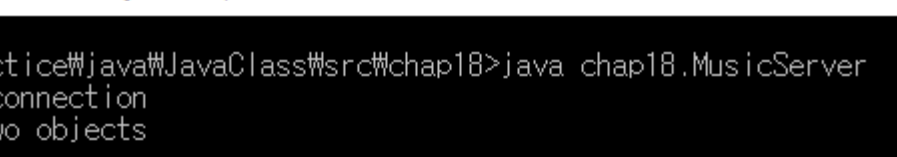
Tempo Down

sendit

Serialize It

명령 프롬프트 - java

D:\practice\java>java -cp .\lib*.jar org.javadev.beatbox.CyberBeatBox
got a connection
read two objects



The screenshot shows a Windows Command Prompt window with the title bar '명령 프롬프트 - java chap18.MusicServer'. The command prompt displays the following text:

```
D:\practice\java\JavaClass\src\chap18>java chap18.MusicServer
got a connection
read two objects
```



```
CA 명령 프롬프트 - java chap18.BeatBoxFinal jam
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Wsam>cd D:\practice\java\JavaClass\src\chap18

C:\Users\Wsam>d:

D:\practice\java\JavaClass\src\chap18>java chap18.BeatBoxFinal jam
got an object from server
class java.lang.String
```

The Cyber BeatBox interface features a 16x16 grid of square drum pads. On the right side, there are control buttons: 'Start', 'Stop', 'Tempo Up', 'Tempo Down', 'sendIt', and 'Serialize It'. Below the grid, there are two text input fields. The first field contains the text '안녕~' (Hello~). The second field contains the text 'jam0: 안녕~' (jam0: Hello~).

두 명 이상이 통신하는 것처럼 시뮬레이션 해보기

The Cyber BeatBox interface includes a list of drum types on the left side, each with a corresponding 4x4 grid of pads. The drum types are: Bass Drum, Closed Hi-Hat, Open Hi-Hat, Acoustic Snare, Crash Cymbal, Hand Clap, High Tom, Hi Bongo, Maracas, Whistle, Low Conga, Cowbell, Vibraslap, Low-mid Tom, High Agogo, and Open Hi Conga. Some pads in the grids are checked, indicating they are active.

The control buttons for the Cyber BeatBox interface are: 'Start', 'Stop', 'Tempo Up', 'Tempo Down', 'sendIt', and 'Serialize It'.

```
CA 명령 프롬프트 - java chap18.MusicServer
D:\practice\java\JavaClass\src\chap18>java chap18.MusicServer
got a connection
read two objects
got a connection
read two objects
```

The text input fields for the Cyber BeatBox interface are: 'hi~' and 'sam0: hi~ jam0: 안녕~'.

실습과제 16-5 생명 게임 (Game of Life)

아래 사이트의 소스 코드를 이용하여 Conway's Game of Life를 시뮬레이션 하시오.

<https://bitstorm.org/gameoflife/code/>

게임 설명: <http://www.bitstorm.org/gameoflife/code/>



실습과제 16-6 BeatBox 프로그램 개선하기(40점)

1. MusicServer.java를 컴파일하여 수행하면 아무런 메시지가 보이지 않는다. 아래처럼 서버가 시작되었다는 메시지가 표시되도록 수정하십시오.
2. 일단 패턴을 선택하면 현재 패턴이 무엇이든지 날라가 버린다. 그것이 현재 작업하고있는 새로운 패턴이었다면, 그냥 운이 없다고 생각해야 한다. 현재 패턴을 저장할지 묻는 대화 상자를 팝업으로 표시하면 좋지 않을까? 이 문제를 해결하십시오.
3. 커맨드라인에서 인자(argument)를 입력하지 않고 실행하면 예외가 발생한다! 커맨드라인에서 인자를 입력했는지 묻는 뭔가를 main 메소드에 넣으시오. 만일 사용자가 인자를 넣지 않고 실행을 시도하면, 디폴트 값을 선택하거나, 또는 인자를 넣고 다시 실행하라는 메시지를 표시해준다.
4. 버튼을 클릭하면 랜덤 패턴을 생성할 수 있는 기능이 있으면 참 좋을 것이다. 그 중에서 사용자가 정말 좋아하는 패턴을 선택할 수 있을 것이다.

[Project 4: Online Music Library and Playlist]

사이버캠퍼스 '과제' 참조!!

