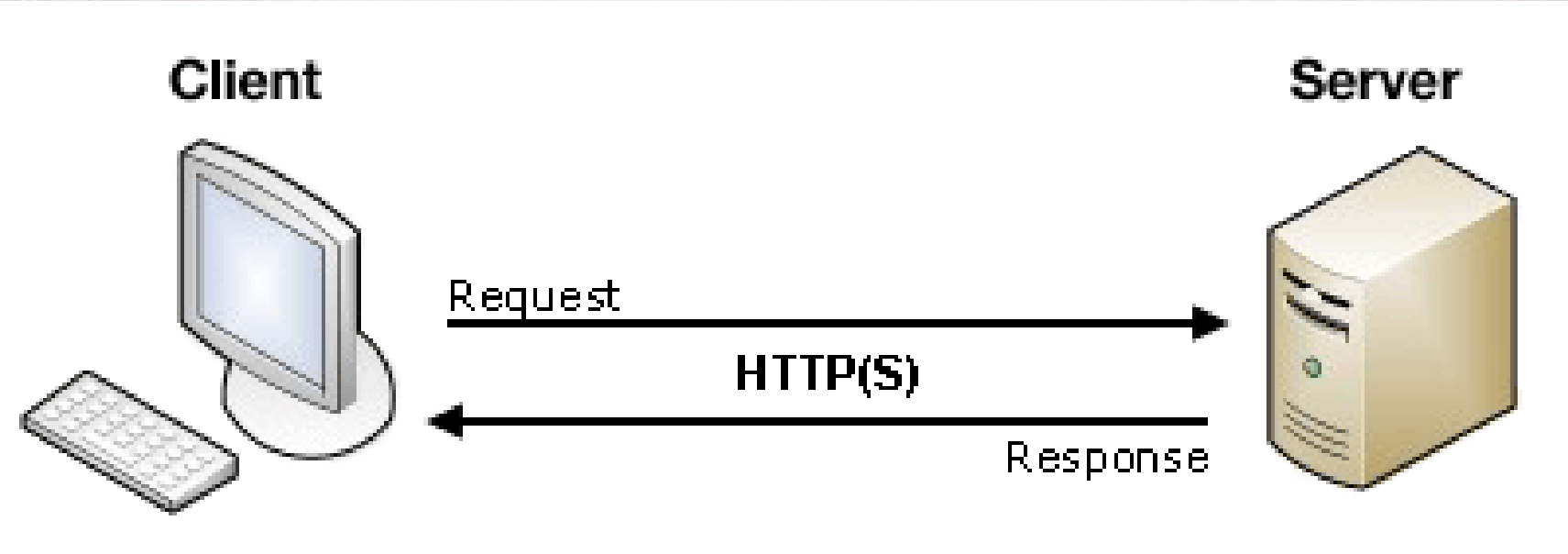
멀티태스킹-서버

comsi.java@gmail.com
박 경 태

1. 서버와 클라이언트

- 서버와 클라이언트
  - 클라이언트가 정보를 제공하는 서버에 접속하여 각종 요청(Request)을 보내고 응답(Response)을 받는 구조



■ 동시접속자가 무제한으로 증가할 경우 발생하는 현상

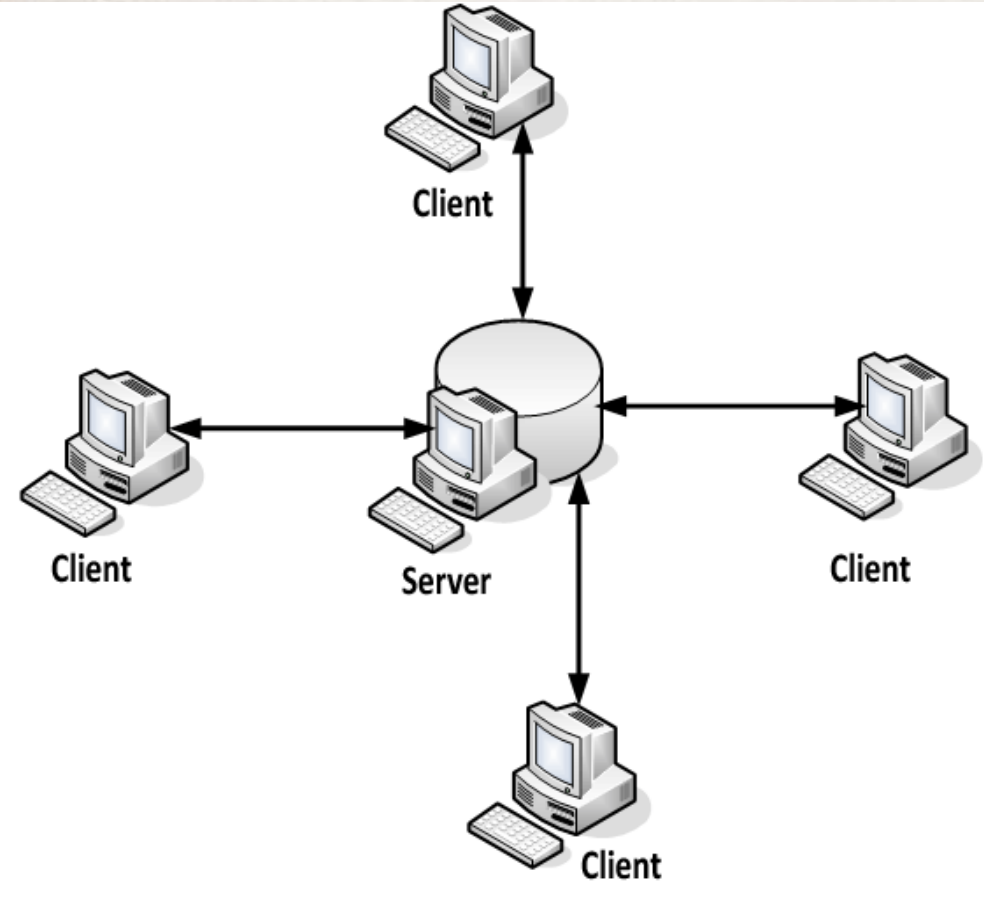
- 클라이언트
  - 서버로 보낸 메시지에 대한 처리 응답이 늦게 도착함
  - 서버로의 접속 과정이 매우 오래 걸림
  - 서버와의 연결이 돌발 해제됨
    - TCP retransmission timeout
    - 사용자 정의 keep alive 메시징의 타임아웃
  - 서버로의 접속 자체가 실패함(타임아웃)

■ 동시접속자가 무제한으로 증가할 경우 발생하는 현상

- 서버
  - CPU 사용량이 증가
  - 클라이언트로부터 메시지를 받는 속도보다 메시지를 처리하는 속도가 느린 경우 RAM 사용량이 증가
  - 클라이언트에게 보낼 메시지의 발생속도보다 실제로 메시지를 보내는 속도가 느린 경우 RAM 사용량이 증가
  - 즉, CPU의 과부하는 RAM의 사용 증가로 이어짐

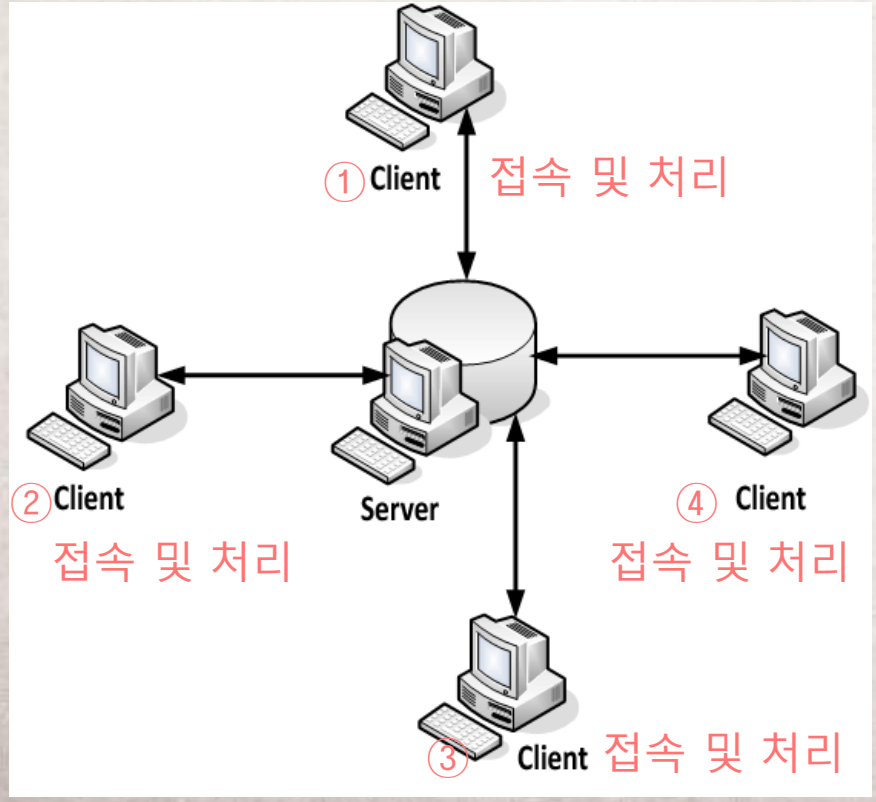
1.1. 반복 서버

- 반복서버
  - 한 번에 한 클라이언트만 처리하는 서버 형태
  - 접속된 클라이언트에 대한 서비스가 끝날 때가지 새 클라이언트의 작업을 수행할 수 없다.



1.2. 다중서버

- 다중서버
  - 다른 클라이언트의 연결에 간섭 없이, 각각의 연결을 독립적으로 수행하는 서버 형태
  - 두 가지 접근 방식
    - 클라이언트당 스레드(thread-per-client) - 각 클라이언트와 연결하기 위해 새 스레드를 만들어내는 방식
    - 스레드 풀(thread pool) - 미리 고정된 개수의 스레드 집합을 준비하는 방식



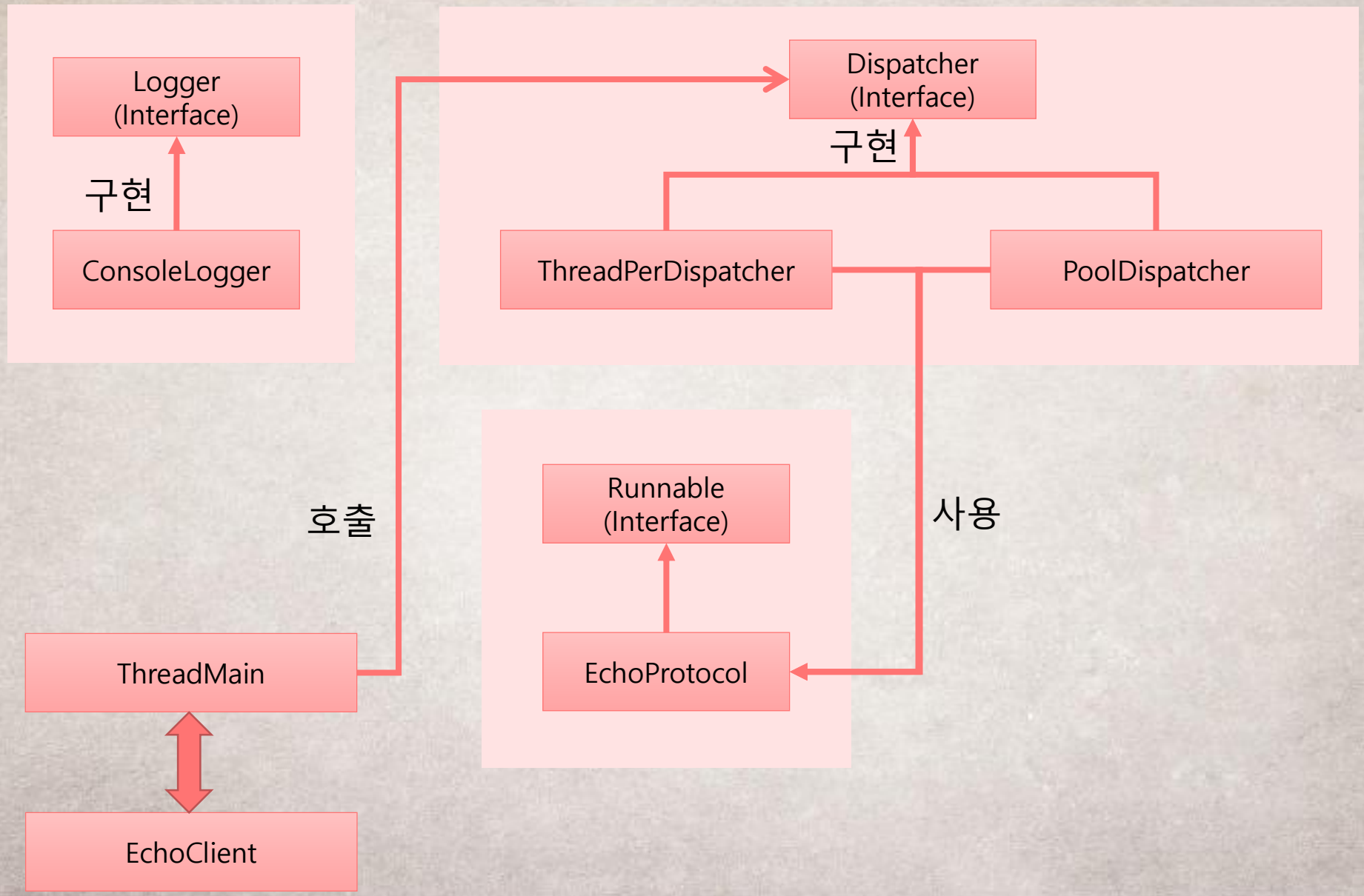
예제12.1~2 다중 서버 구성

- 클라이언트당 스레드(thread-per-client)
- 스레드 풀(thread pool)

2.1.서버 프로토콜 클래스

- 서버와 클라이언트가 주고 받을 데이터에 대한 처리
- 서버와 클라이언트가 연결되면 서버 프로토콜 클래스 인스턴스가 생성되고, run() 함수 호출 시 시작
- 클래스가 Runnable 인터페이스를 구현
  - run() 함수를 수행하는 스레드를 생성
  - run() 함수를 직접 실행

클래스 다이어그램


Logger
(Interface)
구현
ConsoleLogger
Dispatcher
(Interface)
구현
ThreadPerDispatcher
PoolDispatcher
호출
사용
Runnable
(Interface)
ThreadMain
EchoProtocol
EchoClient

Logger 클래스와 ConsoleLogger 클래스

Logger.java ConsoleLogger.java

```java
import java.util.Collection;


public interface Logger {
    // entry의 모든 라인 출력
    public void writeEntry(Collection<String> entry);
    // 라인 한 줄 출력
    public void writeEntry(String entry);
}
```

Logger.java ConsoleLogger.java

```java
import java.util.Collection;
import java.util.Iterator;


public class ConsoleLogger implements Logger {

    @Override
    public synchronized void writeEntry(Collection<String> entry) {
        for(Iterator<String> line = entry.iterator();line.hasNext();)
            System.out.println(line.next());
            System.out.println();
    }

    @Override
    public synchronized void writeEntry(String entry){
        System.out.println(entry);
        System.out.println();
    }

}
```

Iterator 인터페이스:
Collection으로부터 객체를 가져올 때 사용

EchoProtocol 클래스

클라이언트와 연결 후 클라이언트 정보/스레드 정보 그리고 수신 정보 출력

Logger.java | ConsoleLogger.java | EchoProtocol.java

```
1  import java.io.IOException;
2  import java.io.InputStream;
3  import java.io.OutputStream;
4  import java.net.Socket;
5  import java.util.ArrayList;
6
7
8  class EchoProtocol implements Runnable {
9
10     // 입출력 버퍼(byte) 크기
11     static public final int BUFF_SIZE_BYTE = 32;
12
13     private Socket connSocket; // 연결 소켓
14     private Logger logger; // 로거
15
16     public EchoProtocol(Socket connSocket, Logger logger) {
17         this.connSocket = connSocket;
18         this.logger = logger;
19     }
20
21     @Override
22     public void run() {
23         ArrayList<String> entry = new ArrayList<String>();
24         entry.add("Client address and port = " +
25                 connSocket.getInetAddress().getHostAddress() + ":" +
26                 connSocket.getPort());
27
28         entry.add("Thread = " + Thread.currentThread().getName());
29
```

에코 프로토콜 구현

ArrayList에 로거에 대한 내용을 저장

EchoProtocol 클래스

```
29
30          try{
31              // 소켓으로부터 입출력 스트림 획득
32              InputStream inStream = connSocket.getInputStream();
33              OutputStream outStream = connSocket.getOutputStream();
34
35              int receiveMsgSize; // 받은 메시지의 크기
36              int totalBytesEchoed = 0;   // client로부터 받은 바이트 수
37              byte[] echoBuffer = new byte[BUFF_SIZE_BYTE]; // 수신버퍼 생성
38
39              // 클라이언트가 연결을 닫을 때가지(-1) 데이터를 받는다.
40              while((receiveMsgSize = inStream.read(echoBuffer)) != -1){
41                  outStream.write(echoBuffer, 0, receiveMsgSize);
42                  totalBytesEchoed += receiveMsgSize;
43              }
44
45              entry.add("Client finished; echoed " + totalBytesEchoed + " bytes.");
46
47          }catch(IOException e){
48              entry.add("Exception = " + e.getMessage());
49          }
50
51          try {
52              connSocket.close();
53          } catch (IOException e) {
54              entry.add("Exception = " + e.getMessage());
55          }
56
57          logger.writeEntry(entry);
58      }
59
60  }
61
```

입출력스트림 생성

입력스트림을 통해서 받은 데이터를 바로 클라이언트로 출력 (최대 32byte).

에코프로토콜 실행

Dispatcher 클래스

Dispatcher.java

```java
import java.net.ServerSocket;


public interface Dispatcher {
    public void startDispatching(ServerSocket serverSocket, Logger logger);
}

```

ServerSocket을 통해 클라이언트를 처리한다.

ThreadPerDispatcher 클래스

Dispatcher.java | ThreadPerDispatcher.java

```java
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;


public class ThreadPerDispatcher implements Dispatcher {

    @Override
    public void startDispatching(ServerSocket serverSocket, Logger logger){

        // 무한 반복, 클라이언트의 각 연결을 위한 스레드를 만들고 연결을 받아들인다.
        logger.writeEntry("Thread-per-client Server Starting...");
        for(;;){
            try{
                // 연결을 기다리며 대기 상태
                Socket socket = serverSocket.accept();
                Thread thread = new Thread(new EchoProtocol(socket, logger));
                thread.start();
                logger.writeEntry("Created and started Thread = " + thread.getName());

            }catch(IOException e){
                logger.writeEntry("Exception = "+ e.getMessage());
            }
        }

    }
}
```

PoolDispatcher 클래스

Dispatcher.java | ThreadPerDispatcher.java | PoolDispatcher.java

```
1  import java.io.IOException;
2  import java.net.ServerSocket;
3  import java.net.Socket;
4
5
6  class PoolDispatcher implements Dispatcher {
7
8      // 기본 스레드 풀 크기
9      static final int NUMBER_OF_THREADS = 8;
10
11     private int numThreads;
12
13     public PoolDispatcher() {
14         numThreads = NUMBER_OF_THREADS;
15     }
16
17     @Override
18     public void startDispatching(final ServerSocket serverSocket, final Logger logger) {
19         // N-1개의 스레드를 만든다.
20         // 각 스레드가 박복서버로 시작
21         for(int i = 0 ; i < numThreads - 1 ; i++){
22             Thread thread = new Thread(){
23                 public void run(){
24                     dispatchLoop(serverSocket, logger);
25                 }
26             };
27             thread.start();
28             logger.writeEntry("Created and started Thread = " + thread.getName());
29         }
30         logger.writeEntry("Iterative server starting in main thread " + Thread.currentThread().getName());
31         // N 번째 스레드로 생성
32         dispatchLoop(serverSocket, logger);
33     }   // end of method
34
```

PoolDispatcher 클래스

```
34
35      private void dispatchLoop(ServerSocket serverSocket, Logger logger){
36          // 무한 반복, 클라이언트의 각 연결을 위한 스레드를 만들고 연결을 받아들인다.
37          for(;;){
38              try{
39                  Socket socket = serverSocket.accept();
40                  Runnable protocol = new EchoProtocol(socket, logger);
41                  protocol.run();
42              }catch(IOException e){
43                  logger.writeEntry("Exception = " + e.getMessage());
44              }
45          }
46      } // end of method
47
48  }
49
```

ThreadMain 클래스

Dispatcher.java | ThreadPerDispatcher.java | PoolDispatcher.java | ThreadMain.java

```java
import java.io.IOException;
import java.net.ServerSocket;


public class ThreadMain {

    public final static int SERVER_PORT = 5000;

    public static void main(String[] args) {

        try {
            ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
            Logger logger = new ConsoleLogger();
            Dispatcher dispatcher = new ThreadPerDispatcher();
            //Dispatcher dispatcher = new PoolDispatcher();

            dispatcher.startDispatching(serverSocket, logger);

        } catch (IOException e) {

            e.printStackTrace();
        }

    }

}
```

EchoClient 클래스

EchoClient.java

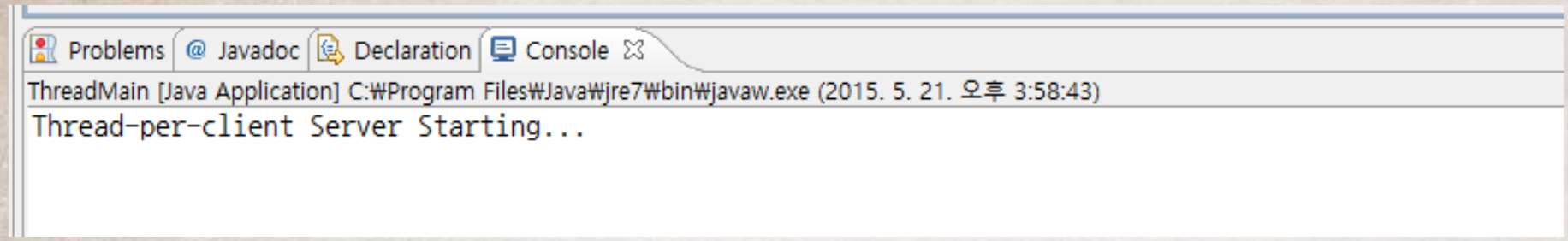
```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;


public class EchoClient {

    // 클라이언트에 요청을 보내기 위한 서버소켓 포트
    private static final int SERVER_SOCKET_PORT = 5000;
    private static final String SERVER_ADDRESS = "localhost";


    public static void main(String[] args) {

        byte[] sendByteBuffer = "Hello World!".getBytes();
        Socket socket = null;

        try {
            // 해당 서버(SERVER_ADDRESS)의 포트(SERVER_SOCKET_PORT)에 연결한다.
            socket = new Socket(SERVER_ADDRESS, SERVER_SOCKET_PORT);

            InputStream inStream = socket.getInputStream();
            OutputStream outStream = socket.getOutputStream();

            // 서버에 문자열을 보낸다.
            outStream.write(sendByteBuffer);
```

EchoClient 클래스

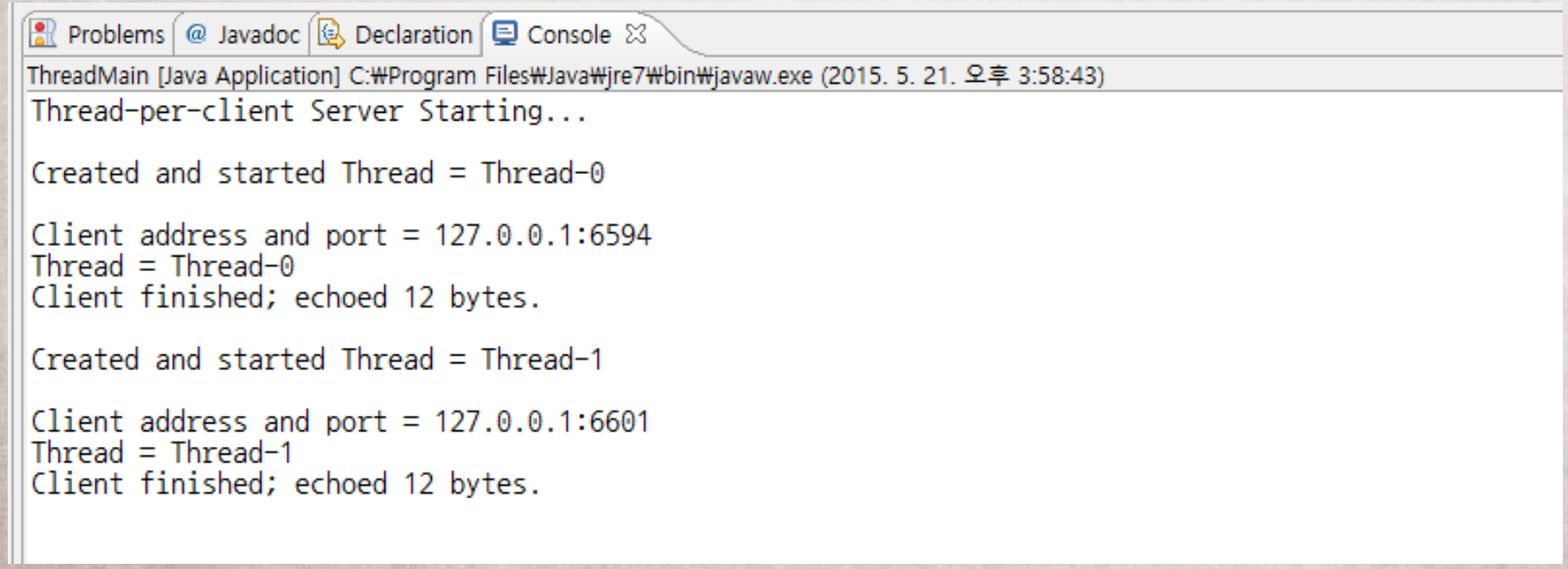
```
30
31              // 서버로부터 돌려 받기 위한 처리
32              int receivedTotalByte = 0; // 전체 받은 바이트 수
33              int bytesRcvd; // 마지막에 read에서 받은 바이트 수
34              while(receivedTotalByte < sendByteBuffer.length){
35                  if((bytesRcvd = inStream.read(sendByteBuffer, receivedTotalByte,
36                          sendByteBuffer.length - receivedTotalByte)) == -1)
37                      throw new SocketException("Connection close prematurely");
38                  receivedTotalByte += bytesRcvd;
39              }
40
41              System.out.println("Received: " + new String(sendByteBuffer));
42
43              socket.close();
44          } catch (UnknownHostException e) {
45              System.out.println("정상적인 Host가 아닙니다.");
46              System.out.println(e.getMessage());
47          } catch (IOException e) {
48              System.out.println("스트림입출력 중에 오류가 발생했습니다.");
49              System.out.println(e.getMessage());
50
51          }
52
53      }
54
55  }
56
```

Thread-per-client: ThreadPerDispatcher 실행

- ThreadPerDispatcher 실행 후 로그 상태

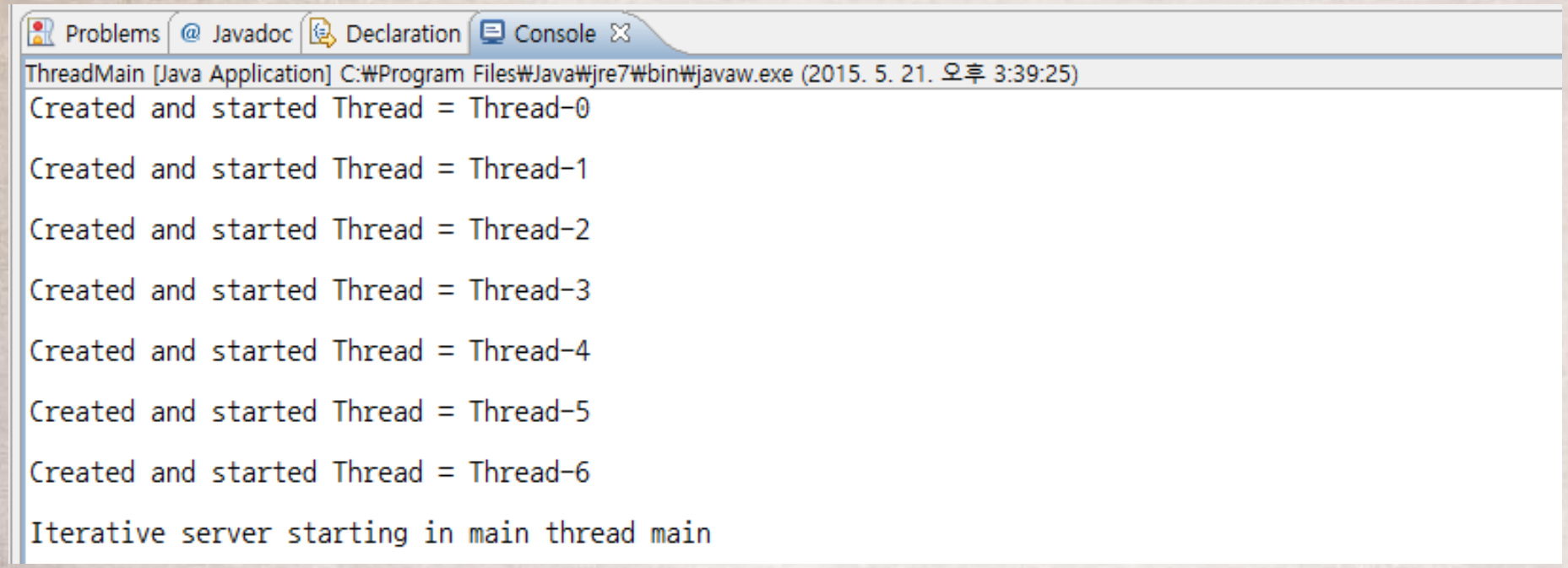


- 클라이언트 접속 후 로그 상태



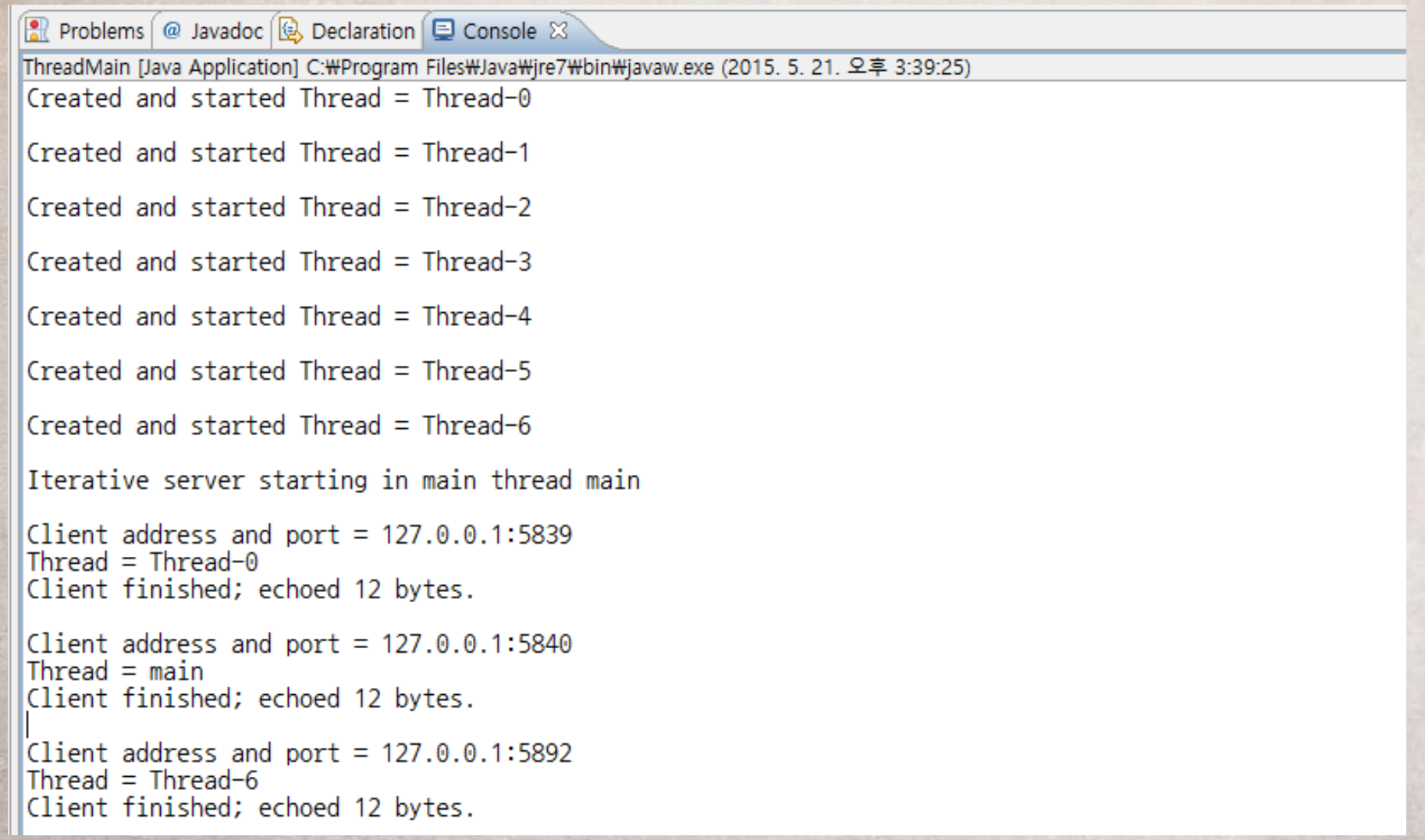
Thread pool: PoolDispatcher 실행

- PoolDispatcher 실행 후 로그 상태



Thread pool: PoolDispatcher 실행

- 클라이언트 접속 로그 상태



- 한글로 자기이름을 서버로 보내고 받기