JAVA

1. Stream
2. IO Stream
3. FileStream

1. Stream

- 스트림이란?
  - 데이터가 전송되는 통로
  - 노드 스트림과 프로세스 스트림으로 나눔
  - 노드 스트림에 연결된 대상에 데이터 전송이 가능(네트워크, 파일 등등...)
- 노드 스트림(node stream)
  - 물리적인 연결, 총 4가지의 추상클래스로 존재
  - 8bit -> OutputStream, InputStream
  - 16bit -> Reader, Writer
- 프로세스 스트림(process stream)
  - 입출력에 관련된 부가기능
  - 단독적으로 선언이 불가하여 노드스트림에 덧붙여서 사용

1. Stream

- 스트림 작성 패턴
  1. 노드 스트림 생성
  2. 프로세스 스트림 작성(선택)
  3. Exception 처리
  4. 입출력 처리
  5. finally 영역에서 close()

2. IO Stream 예제 클래스

- Scanner를 사용하던 문자열 입력 구현
- 콘솔 화면에 출력하는 println() 메서드 구현

2. IO Stream 예제 클래스

- Scanner를 사용하던 문자열 입력 구현
- 콘솔 화면에 출력하는 println() 메서드 구현

```java
public class IOTest {
    public String readline(){
        InputStreamReader isr = new InputStreamReader(System.in);
        char buffer[] = new char[10];
        String message = null;
        try {
            int idx = 0;
            do{
                if(idx == buffer.length-1)
                    buffer = reallocBuffer(buffer);
                buffer[idx] = (char) isr.read();
                idx++;
            }while(buffer[idx-1] != '\n');
            buffer[idx-1] = '\0';
            message = String.copyValueOf(buffer, 0, idx-2);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return message;
    }
```

```
    private char[] reallocBuffer(char[] buffer) {
        char []temp = buffer;
        buffer = new char[temp.length * 2];
        for(int i=0;i<temp.length;i++){
            buffer[i] = temp[i];
        }
        return buffer;
    }
    public void println(String msg){
        char buf[] = msg.toCharArray();
        OutputStreamWriter osw = new OutputStreamWriter(System.out);
        try {
            osw.write(buf);
            osw.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class TestMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        IOTest io = new IOTest();
        System.out.println(io.readline());
        String str = "message";
        io.println(str);
    }
}
```

3. File Stream

- Node Stream을 파일에다가 연결하면 파일 입출력이 가능
- FileInputStream, FileOutputStream, FileReader, FileWriter를 접근 가능
- 텍스트 입출력과 이진형태로 입출력이 가능

```java
import java.io.BufferedReader;

public class FileReadMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        FileReader readFile = null;
        BufferedReader br = null;
        try {
            readFile = new FileReader("gisa.txt");
            br = new BufferedReader(readFile);
            String str = null;

            while(true){
                str = br.readLine();
                if(str == null)
                    break;
                System.out.println(str);
            }
```

```
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            try {
                br.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }

}
```

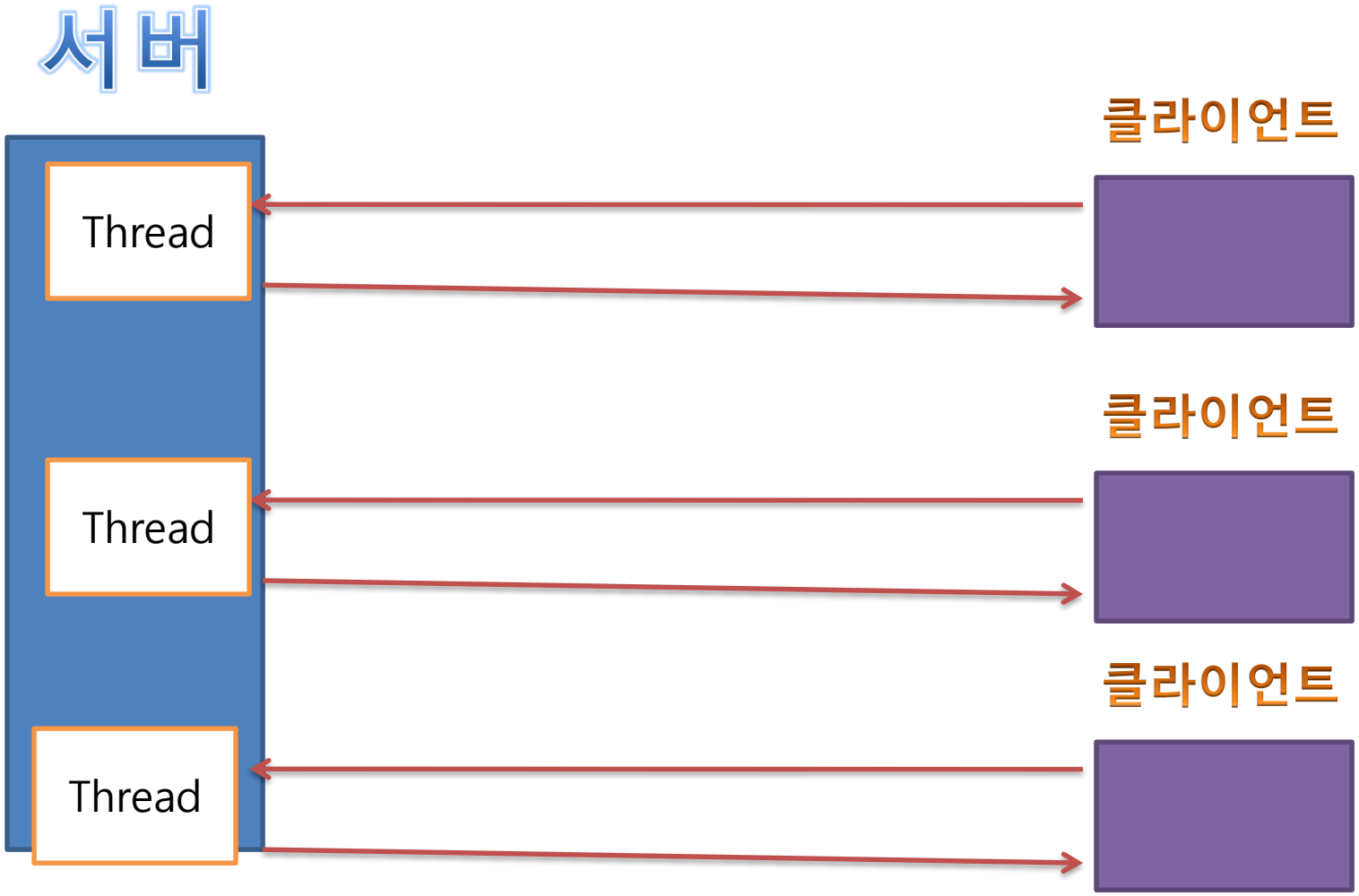
```java
public class FileWriteMain {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        FileWriter fw = null;
        PrintWriter pw = null;
        try {
            fw = new FileWriter("write.txt");
            pw = new PrintWriter(fw);
            pw.println("Hello World");
            pw.println("안녕하세요");
            pw.flush();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }finally {
            pw.close();
        }
    }
}
```

4. Echo Server/Client

- Echo Server
  1. client 접속 받음
  2. client로부터 메시지를 받은 후 다시 클라이언트로 재전송
  3. client로부터 받은 메시지가 exit면 서버를 종료
  4. 2와 3과정을 반복
- client
  1. 서버 접속
  2. 서버로 메시지 보낸 후 다시 메시지를 받음, exit 전송 후 접속 종료
  3. 2번 과정을 반복

5. MultiEchoServer

- 하나의 서버에 여러 개의 클라이언트가 접속
  - 여러 개의 클라이언트가 접속
  - 각각의 클라이언트의 메시지를 각각 되돌려줌
  - 서버에서는 클라이언트 접속을 받은 후 스레드를 생성
  - 스레드에서는 클라이언트에게 메시지를 받아서 그대로 되돌려 줌

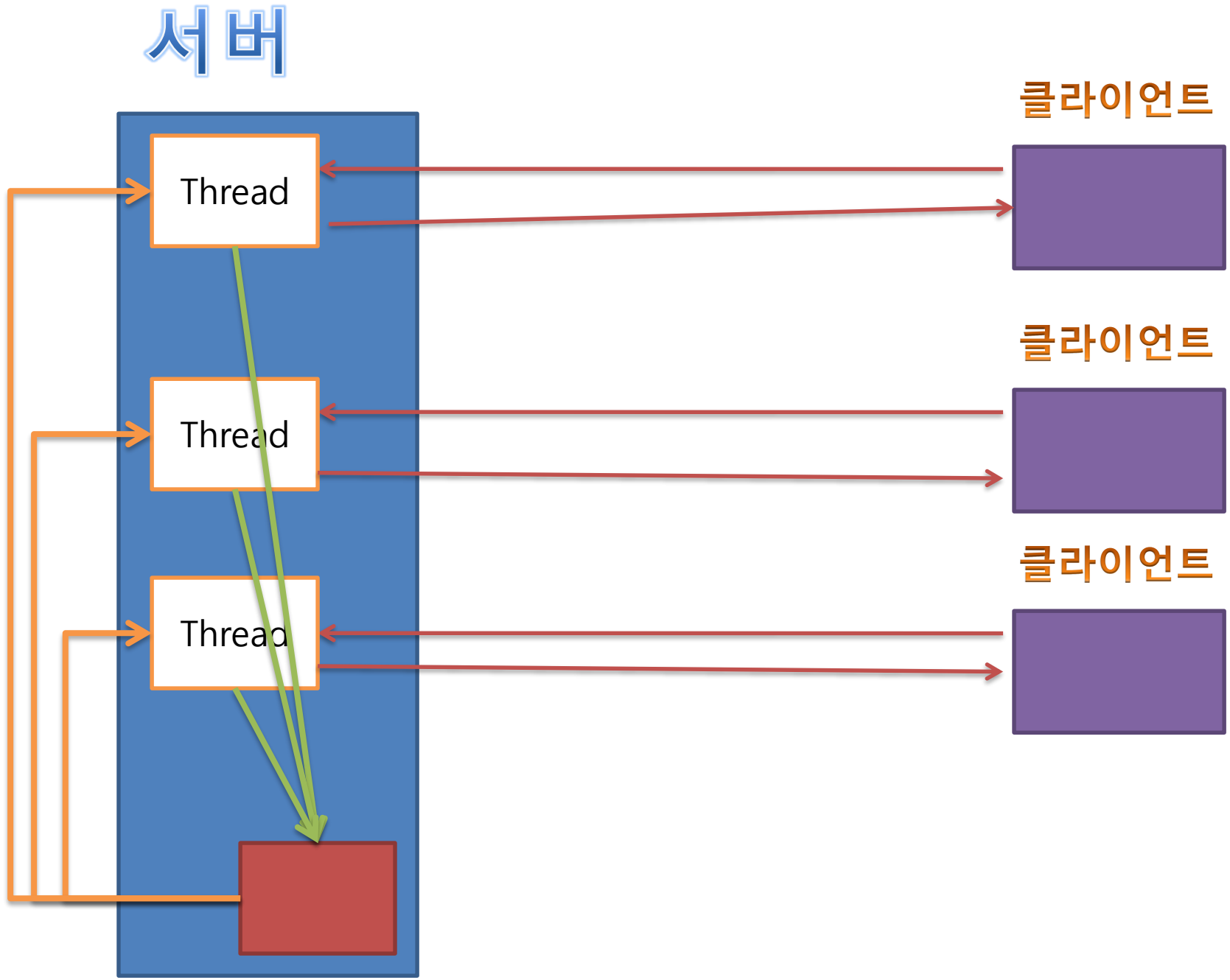

서버
Thread
Thread
Thread
클라이언트
클라이언트
클라이언트

6. 1:1 채팅

1. 서버와 클라이언트가 1:1로 채팅
   - 한쪽이 서버가 되고 클라이언트
   - 1:1로 입출력을 함
   - 한쪽이 메시지를 전송하지 않아도 메시지를 수신 할 수 있어야함
   - 메시지 작성 중에도 메시지를 수신 할 수 있어야함
   - 입출력 작업이 별개로 이루어져야 함
     - 입력과 출력 작업 중 하나가 스레드로 분리가 되어야 함

7. 채팅 서버

1. ServerMain
    1. 서버 오픈
    2. 클라이언트 접속, 스레드 실행, 해당 스레드를 리스트에다가 저장
    3. 스레드에서 메시지를 받으면 전체 클라이언트에게 메시지 전달(static method)
2. Serverworker
    1. 입출력 스트림 초기화
    2. 메시지를 받으면 ServerMain의 전체클라이언트에게 메시지 전달하는 메서드 호출
    3. 클라이언트에게 메시지를 전달할 메서드 하나 작성


서버
Thread
Thread
Thread
클라이언트
클라이언트
클라이언트