디지털컨버전스 기반 UXUI Front 전문 개발자 양성과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - JungHyun LEE(이정현)
akdl911215@naver.com

1)try ~ catch

링크 :

예외란 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인해 발생하는 프로그램 오류를 말합니다. 예외가 발생하면 프로그램이 종료가 된다는 것은 에러와 동일하지만 예외는 예외처리(Exception Handling)을 통해 프로그램을 종료되지 않고 정상적으로 작동되게 만들어줄 수 있습니다. 자바에서는 예외처리를 try catch문을 통해 해줄 수 있습니다.

try블록에는 예외가 발생할 수 있는 코드가 위치해야합니다. try 블록의 코드가 예외없이 정상 실행되면 catch블록의 코드는 실행되지 않고 finally블록의 코드를 실행하게 됩니다. 하지만 try 블록의 코드에서 예외가 발생하면 즉시 실행을 멈추고 catch 블록으로 이동하여 예외처리 코드를 실행합니다. 그리고 finally 블록의 코드를 실행합니다.

try catch 문은 주로 데이터베이스에 데이터를 주고받을 경우에 많이 사용합니다. 데이터 베이스를 거쳐올때는 변수가 많이 새이기 때문에 try catch문은 필수적입니다. 그리고 finally에는 데이터베이스와의 연결을 끊어주는 코드를 주로 삽입하게 됩니다.

특정 예외가 발생하여 데이터베이스와의 연결이 끊어지지 않으면 여러가지 문제를 야기할 수 있기 때문입니다.

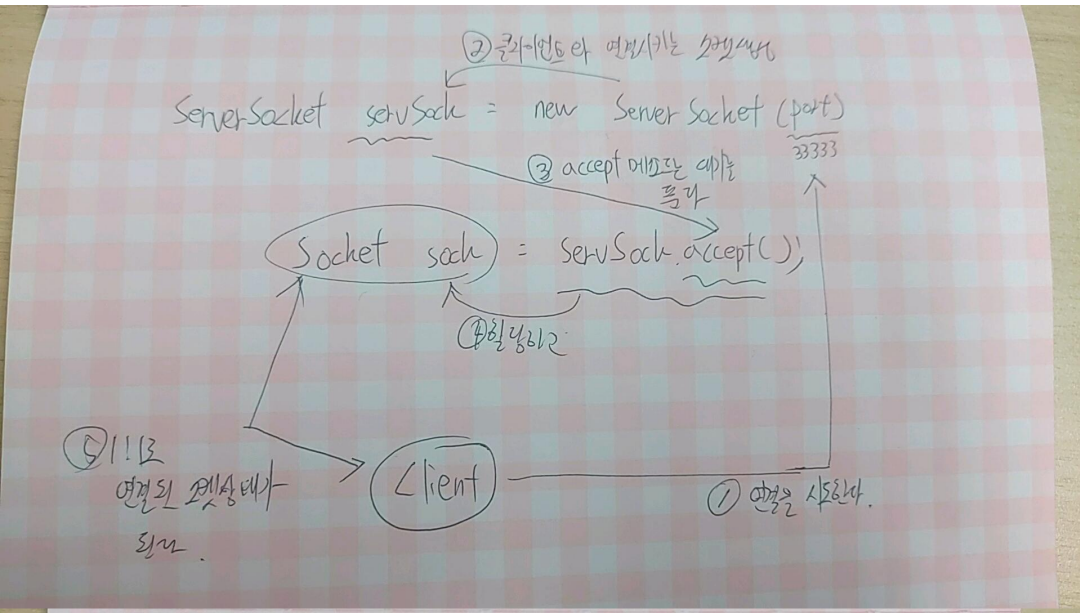
2)SocketServerTest 주석 및 궁금점 페이지

링크 :

```java
public class SocketServerTest {
    public static void main(String[] args) {
        // 문자열을 숫자로 바꾸는 기법
        // "33333"은 문자열로 인식 되기 때문에 밑에 Integer.parseInt 가 문자열을
        // 정수 33333 으로 변경해서 int port 값으로 할당한다.
        int port = Integer.parseInt( s: "33333");

        try {
            // 소켓이란 ? 네트워킹을 할 수 있는 클래스 객체
            // 서버 소켓 생성시 서비스 번호를 부여해야 하는데
            // 이 서비스 번호로 port(3333)을 설정했다.
            // 네트워크 포트가 16비트를 사용하므로 제한은 0 ~ 65535
            // (단, 고정된 포트를 사용하면 안됨)

            // 프로그래밍을 실행하게 되면 포트를 할당하게 된다. 그리고
            // 위의 int port는 33333으로 할당해주었기 때문에
            // 지금 만들려는 서버의 포트 넘버는 33333번이 된다.
            ServerSocket servSock = new ServerSocket(port);

            System.out.println("Server: Listening - " + port);

            // while이 없어도 돌아가는데 존재의 이유가 무엇이지??
            while(true) {
                // accept()의 경우 클라이언트가 접속을 요청했는지 체크
                // (accept()는 블로킹 연산이다)

                // 결국 sock은 클라이언트 소켓을 의미하게 된다.
                // 그래서 좀 더 가독성이 좋은 코드는 Socket clntSock으로 작성하면 좋을듯
                // (전화왔을때 통화하기 슬라이드가 accept()라 보면됨)
                Socket sock = servSock.accept();
```

```
// 접속한 클라이언트의 IP를 확인하는 코드
System.out.println(
        "[" + sock.getInetAddress() +
                "] client connected"
);

// 출력을 위한 객체를 만듭니다.
// 클라이언트에게 출력할 객체를 만든다.
// accept로 수신이 활성화된 데이터가 sock 으로 할당되고
// sock 이 OutputStream out 으로 할당하기 때문에 클라이언트에
// 데이터를 출력할수 있게 됩니다.
OutputStream out = sock.getOutputStream();

// println의 결과를 out으로 전송한다라는 뜻
// 내가 서버에 용청하는것: 출력이 아닌 입력이다.
// 서버가 처리해서 도려주는 것: 이것이 출력이다.
PrintWriter writer = new PrintWriter(out,  autoFlush: true);
// 즉 여기서 println의 출력은 클라이언트에게 간다.
// Date()는 시간을 가져온다.
// toString()은 문자열로 만듬.

// 접속 상대에게 보내고 싶은 데이터를 이곳에 기로간다.
writer.println(new Date().toString());

// 입력: 클라이언트
// 클라이언트가 서버에게 보낸것
// 클라이언트의 OutputStream out 값이 할당하게 된다.
InputStream in = sock.getInputStream();
```

실행의 구조는 밑에 그림처럼 될듯하다.



```
        // 소켓을 통해 데이터를 읽은땐 무조건 아래 형식으로 진행합니다.
        // 바뀌지 않으니 항상 고정해서 사용하면 됩니다.
        // 이 버퍼에 클라이언트가 보낸 내용이 들어있다.
        BufferedReader reader =
                new BufferedReader(new InputStreamReader(in));

        // 그러므로 rader.redLine()을 통해서
        // 내용을 읽으면 클라이언트가 보낸 내용을 출력할 수 있게 된다.

        // 클라이언트의 InputStream > reader > .readLine으로 통해서
        // 바이트 단위의 문자열을 읽어서 출력하게 된다.
        System.out.println("msg: " + reader.readLine());
    }

    // catch(어떤 예외?) - Exception은 모든 예외
} catch (IOException e) {
    // Exception은 예외 처리로
    // I/O 예외가 발생하면 무엇인가 잘못되었음을 감지하고
    // 어디가 잘못되었는지 출력하도록 구성된다.
    // ex) 통신중에 갑자기 네트워크 불안정으로 통신이 끊기면
    //     catch가 I/O 예외를 감지하고 아래 코드가 동작하게 된다.
    // 아래 코드는 에러 메시지를 출력하는 코드로 언제나 동이랗게 작성하면 됨
    System.out.println("Server Exception: " + e.getMessage());

    // 콜 스택(매서드 호출)이 어떤식으로 이뤄졌는지 상태를 보여줌
    // 디버깅을 위해서 많이 사용하는 편
    e.printStackTrace();
}
```

3)SocketClientTest 주석 및 궁금점 페이지

링크 :

```java
public class SocketClientTest {
    public static void main(String[] args) {
        // 내가 접속할 서버의 IP 주소를 적습니다.
        String hostname = "192.168.0.35";
        String hostname2 = "192.168.0.9";
        // 서버에 여러 서비스가 있을 수 있는데
        // 그 중에서 내가 사용하고자 하는 서비스의 포트 번호를 적습니다.

        // 33333을 입력한 이유는 SocketServerTest의 포트번호에 접속한듯 하다.
       int port = 33333;

        for(int i = 0; i < 10; i++) {
            try {
                // Socket 객체를 할당해서
                // 서버의 IP, 포트 번호를 가지고 접속을 요청합니다.
                // 서버에 대한 소켓을 획득하게 됩니다.
                // 이 요청이 들어갈때 서버의 accept()가 동작하게 됩니다.
                // 예를 들자면 이 행위는 전화를 거는것과 같다.
                // (서버쪽 주석을 살펴보면 감이 더 잘 올 것이다.)
                Socket sock = new Socket(hostname, port);

                // 서버의 출력을 획득 > 서버쪽에서 밑에 코드랑 동일한
                // 코드로 출력을 하기때문에 클라이언트요청하기 때문에
                // 동일한 코드로 수신할 준비를 설정하는 것으로 생각 한다.
                // 그렇기때문에 강사님이 밑에 주석같이 말씀하신듯 하다.
                // 즉 서버가 수신하게 만들도록 설정을 해주는 것
                OutputStream out = sock.getOutputStream();
```

```
    // 서버의 포트 33333에 연결해서 요청과 출력을 가능하게 만든 후
    // "Hello Network Programming" 을 String 으로 전송한다.
    String str = "Hello Network Programming";

    // 위의 문자열을 바이트 단위로 쪼개서 서버로 전송한다.

    // getBytes은 유니코드 문자열(String)을 바이트코드로 인코딩 해주는 메소드
    // 만약 getBytes()의 인자로 캐릭터셋을 넘기지 않으면 사용자 플랫폼의 기본
    // charset으로 인코딩 된다.

    // 바이트코드로 인코딩 해주는 메소드를 사용하는 이유는 스트링자체를 넘겨서
    // 바이트가 손상이 나면은 복원을 할 수 없기 때문이다.
    // 예를 들어서 1.234f 라는 실수가 int 형으로 변환이 되면 1이 될것이고
    // 이것이 다시 flaot형으로 변환된다면 1.0f가 되는것과 같은 이치라고 생각된다.
    out.write(str.getBytes());

    // 서버의 입력을 생성(수신)
    InputStream in = sock.getInputStream();
    BufferedReader reader = new BufferedReader(new InputStreamReader(in));

    // 서버가 보낸 내용을 time에 저장하고 출력한다.
    String time = reader.readLine();
    System.out.println(time);

    // UnknownHostException: 내가 접속하려는 IP를 찾지 못할 때때
} catch (UnknownHostException e) {
    System.out.println("Server Not Found : " + e.getMessage());
} catch (IOException e) {
    System.out.println("I/O Error: " + e.getMessage());
}
```