

Reactor

**간단한 서버를 구현할거임.
잼?**

서버 초기화 코드임.
ServerInitializer 인스턴스에 dispatch 하셈.

```
public class ServerInitializer {  
    public static void main(String[] args) {  
        try {  
            new ServerInitializer().dispatch();  
        } catch (IOException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

IOException을 잡아내야 한다는걸 잊지마셈.

```
public class ServerInitializer {  
    public static void main(String[] args) {  
        try {  
            new ServerInitializer().dispatch();  
        } catch (IOException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

손교수님이 try-catch 생활화 하렘.
try-catch는 과학임.

```
public class ServerInitializer {  
    public static void main(String[] args) {  
        try {  
            new ServerInitializer().dispatch();  
        } catch (IOException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

이게 ServerInitializer 클래스의 dispatch 메서드임.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


잘 보셈. IOException을 던지고 있음.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

보고 있음? 이걸 방금 메인에서 try-catch 했음.
고로 신경쓰지마셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


서버 소켓을 생성하셈.
지금 소켓을 생성했단걸 기억하셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

무한 루프임. 빠져나갈 수 없음.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

큰일이다. TTT

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


생성한 소켓에서 이벤트 발생하길 손꼽아 기다림

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

■

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


■ ■

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

...

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

....

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


.....

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

.....?

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


!.....?

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

이벤트 발생함. 소켓이 반환됨.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

소켓? 이미 만들었잖슴? 왜 또 생김?

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


구글에 검색하셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

got it?

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


이번엔 도착한 메시지에서 정보 빼돌릴 거임.
조용히 말하테니 잘 알아먹으셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

소켓에서 `getInputStream`하면 스트림 형태로 데이터 나옴.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

막 계속 나옴. 줄줄 샘.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


변수 하나 정해서 흐르는거 받으셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

변수 하나 정해서 흐르는거 받으셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


엥? 근데 이 정보는 어디서 나옴?

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

보낸 놈이 넣어 댔겠지. 뭐.
알간?

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

이제 정보를 분석해볼거임. 레츠고.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


이게 머리 부분임.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


이건 몸.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

한국말이 불편하면 영어론 헤더와 바디임.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

빼돌린 정보에서 헤더 정보 얻을거임.
6바이트 만큼 공간 만드셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


스트림에서 그 크기만큼 읽으셈.
buffer 데이터가 채워짐.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

그 데이터를 문자열로 바꿈.
따라하셈.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

바디는 헤더랑 똑같이 반복하셈.
단, 바디는 1024 바이트임.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```


빨리빨리 하셈. 시간이 없음.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

자, 이제 이 정보로 뭘 좀 해보자.

```
public void dispatch() throws IOException {  
    ServerSocket serverSocket = new ServerSocket(5000);  
  
    while (true) {  
        Socket socket = serverSocket.accept();  
        InputStream inputStream = socket.getInputStream();  
  
        byte[] buffer = new byte[6];  
        inputStream.read(buffer);  
        String header = new String(buffer);  
  
        buffer = new byte[1024];  
        inputStream.read(buffer);  
        String data = new String(buffer);  
  
        execute(header, data);  
    }  
}
```

이게 바로 그 뭘좀 해보는 메서드임.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```


헤더와 바디 받음.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

일단 바디에 들어있는 정보 분석할거임.
잘 따라오셈.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

일단 바디에 들어있는 정보 분석할거임.
잘 따라오셈.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```


바디는 최대 10개로 쪼갤거임.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

I <- 이거 기준으로 쪼갤거임.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

쪼개서 배열에 넣으셈.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```


이제 헤더를 이용해 볼거임.
잘 보셈.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

헤더는 16진수로 표현된 문자열임.
그걸 숫자로 바꾸셈.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

그 숫자를 기준으로 케이스를 나눌거임.
0x5001과 0x6001이 있음.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```


헤더에 따라서 바디를 조작할 메서드가 다름.

```
private void execute(String header, String data) {  
    String[] params = new String[10];  
    StringTokenizer token = new StringTokenizer(data, "|");  
    int i = 0;  
  
    while (token.hasMoreTokens())  
        params[i++] = token.nextToken();  
  
    switch (Integer.valueOf(header.split("x")[1], 16)) {  
        case 0x5001:  
            sayHello(params);  
            break;  
        case 0x6001:  
            updateProfile(params);  
            break;  
    }  
}
```

0x5001은 sayHello임.

```
private void sayHello(String[] params) {  
    System.out.println("SayHello -> name : " + params[0] + " age : " + params[1]);  
}  
  
private void updateProfile(String[] params) {  
    System.out.println("UpdateProfile -> " +  
        " id : " + params[0] +  
        " password : " + params[1] +  
        " name : " + params[2] +  
        " age : " + params[3] +  
        " gender: " + params[4]);  
}
```

0x6001은 updateProfile임.

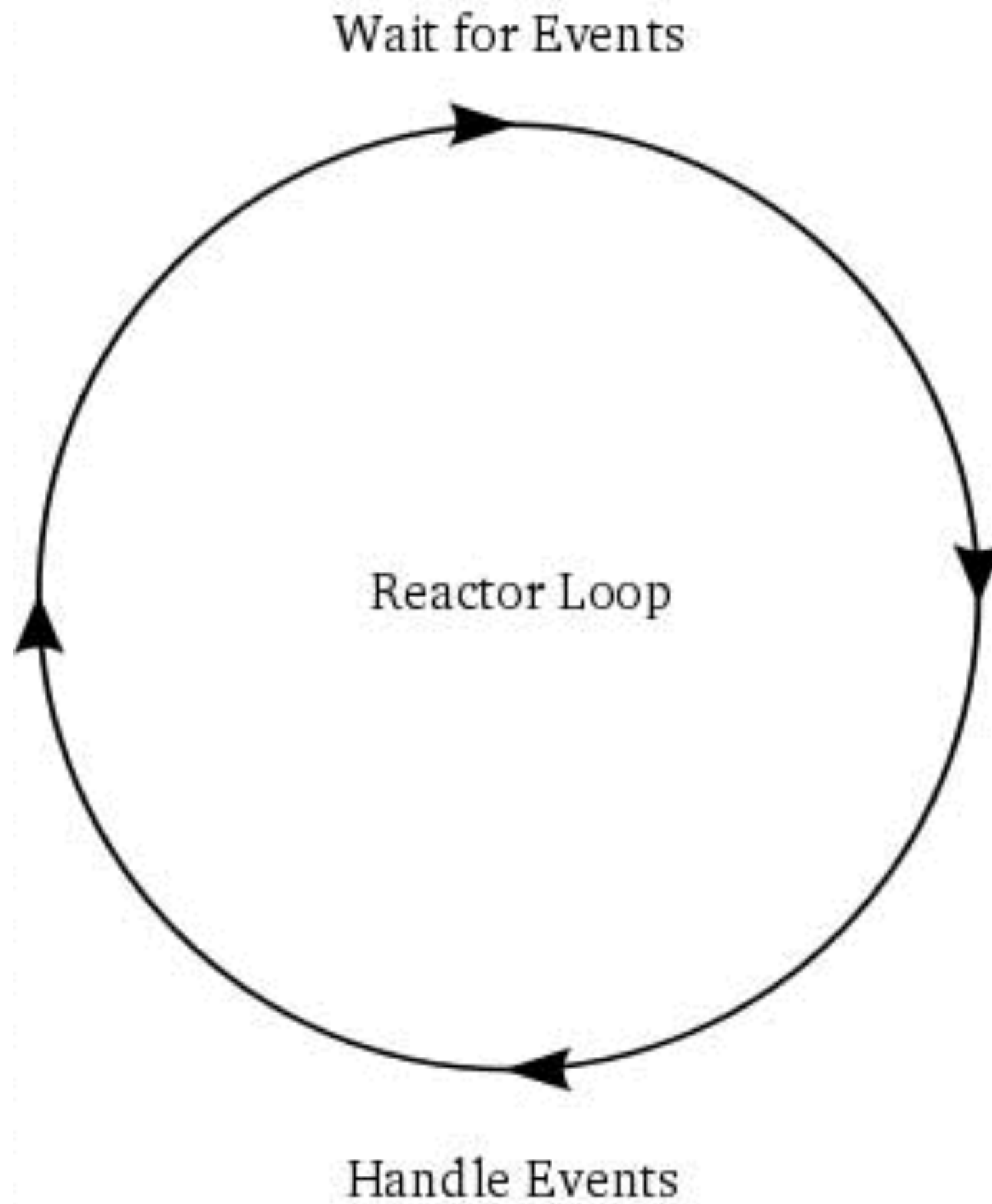
```
private void sayHello(String[] params) {  
    System.out.println("SayHello -> name : " + params[0] + " age : " + params[1]);  
}  
  
private void updateProfile(String[] params) {  
    System.out.println("UpdateProfile -> " +  
        " id :" + params[0] +  
        " password : " + params[1] +  
        " name : " + params[2] +  
        " age : " + params[3] +  
        " gender: " + params[4]);  
}
```


각자의 방식으로 데이터 조작함.
오케.

```
private void sayHello(String[] params) {  
    System.out.println("SayHello -> name : " + params[0] + " age : " + params[1]);  
}  
  
private void updateProfile(String[] params) {  
    System.out.println("UpdateProfile -> " +  
        " id : " + params[0] +  
        " password : " + params[1] +  
        " name : " + params[2] +  
        " age : " + params[3] +  
        " gender: " + params[4]);  
}
```

Reactor 패턴이란?

서버로 들어온 요청을 처리하기 위한
event handling pattern



Reactor

EventHandler를 관리하는 객체. event demultiplexing과 dispatching 작업을 한다.

1개 이상의 Handle에 발생한 이벤트에 반응하여 EventHandler에게 dispatch 한다.

EventHandler

Reactor가 특정한 이벤트를 처리하기 위해 사용하는 인터페이스

Concrete EventHandler

이벤트를 처리하는 구현된 이벤트 핸들러

- Reactor는 event가 들어오면 알맞는 handler로 dispatch
- Handler는 이 dispatch 된 event를 받아서 처리

**Server
Initialize**

**Event
Handler**

Reactor

Demux



Initialize

registerHandler

getHandle

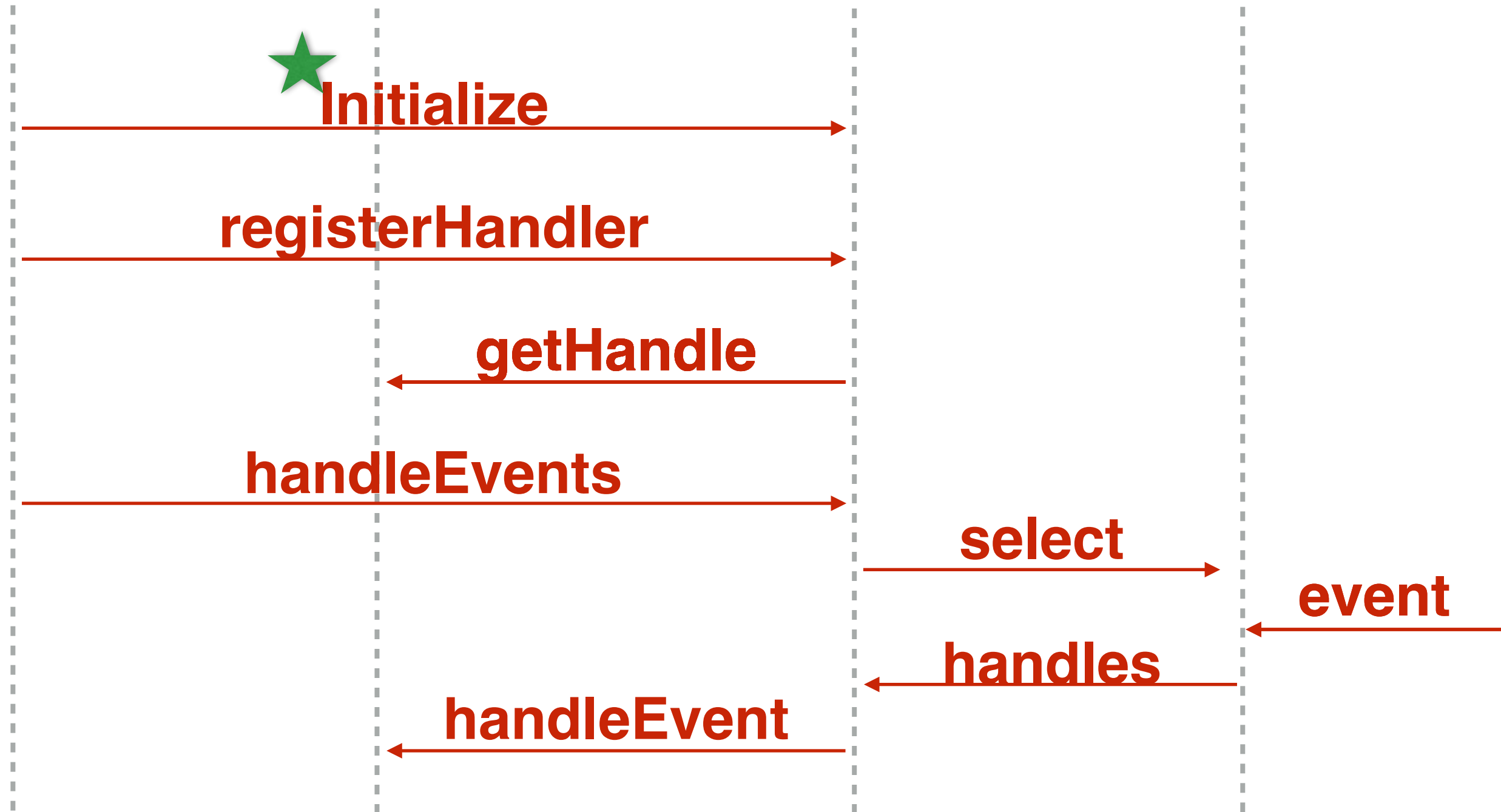
handleEvents

select

event

handles

handleEvent



Main 메서드에서 ServerInitialize를 초기화합니다.


```
public static void main(String[] args) {  
    ServerInitialize serverInitialize = new ServerInitialize();  
    serverInitialize.dispatch();  
}
```


serverInitialize에 dispatch 메시지를 전달합니다.

```
public static void main(String[] args) {  
    ServerInitialize serverInitialize = new ServerInitialize();  
    serverInitialize.dispatch();  
}
```

ServerInitialize 클래스의 dispatch 메서드입니다.
Reactor를 초기화해요.

```
private void dispatch() {  
    reactor = new Reactor();  
  
    reactor.registerHandler(  
        reactor.registerHandler(  
  
    try {  
        reactor.handle_events();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
}
```



```
public Reactor() {  
    handlers = new HandleMap();  
}
```

Reactor는 HandleMap을 초기화합니다.
HandleMap은 EventHandler와 그 키값을 관리해요.

```
public Reactor() {  
    handlers = new HandleMap();  
}
```

```
public class HandleMap extends HashMap<String, EventHandler> {  
}
```


**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize



registerHandler

getHandle

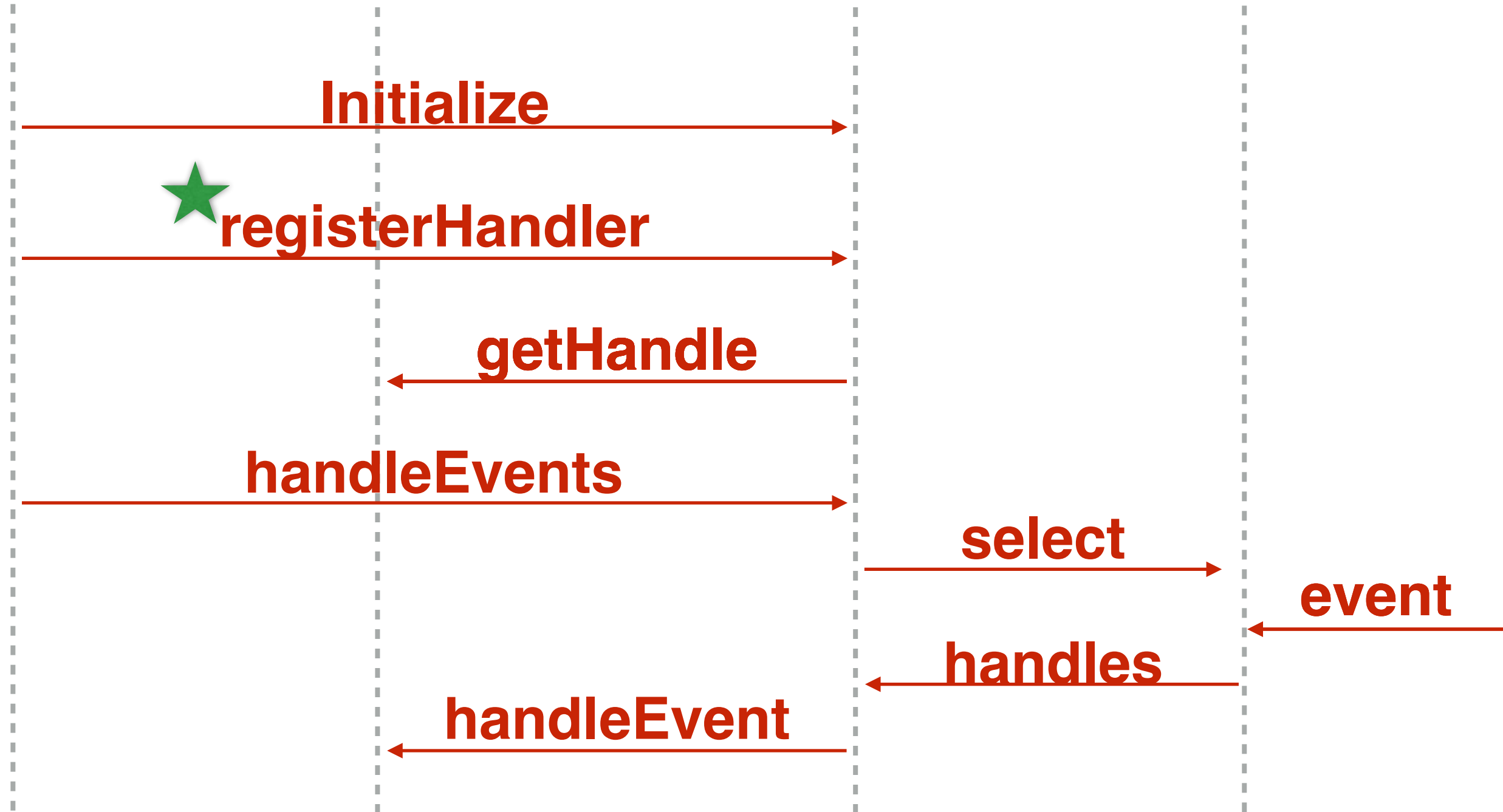
handleEvents

select

event

handles

handleEvent



**ServerInitialize에서 Reactor에 핸들러를 등록합니다.
EventHandler 객체를 전달해요.**

```
private void dispatch() {  
    reactor = new Reactor();  
  
    reactor.registerHandler(new SayHelloEventHandler());  
    reactor.registerHandler(new UpdateProfileEventHandler());  
  
    try {  
        reactor.handle_events();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

handler에서 getHandle 메시지로 key값을 얻어옵니다.
이를 HandleMap에 저장해요.

```
/**
 * @brief Reactor에 이벤트 핸들러를 등록한다.
 * @details handler에서 handle을 반환받는다. 반환받은 handle을 키값으로 handler를 저장한다.
 * @param handler 특정 이벤트를 처리할 핸들러
 * @return Nothing
 */
public void registerHandler(EventHandler handler) {
    handlers.put(handler.getHandle(), handler);
}
```

**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize

registerHandler



getHandle

handleEvents

select

event

handles

handleEvent

handler에서 getHandle 메시지로 key값을 얻어옵니다.
이를 HandleMap에 저장해요.

```
/**
 * @brief Reactor에 이벤트 핸들러를 등록한다.
 * @details handler에서 handle을 반환받는다. 반환받은 handle을 키값으로 handler를 저장한다.
 * @param handler 특정 이벤트를 처리할 핸들러
 * @return Nothing
 */
public void registerHandler(EventHandler handler) {
    handlers.put(handler.getHandle(), handler);
}
```

EventHandler는 인터페이스입니다.

```
public interface EventHandler {  
    /**  
     * @brief 이벤트 핸들러의 키값을 반환한다.  
     * @return HandleKey 특정 이벤트 핸들러의 키값  
     */  
    public String getHandle();  
}
```

**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize

registerHandler

getHandle



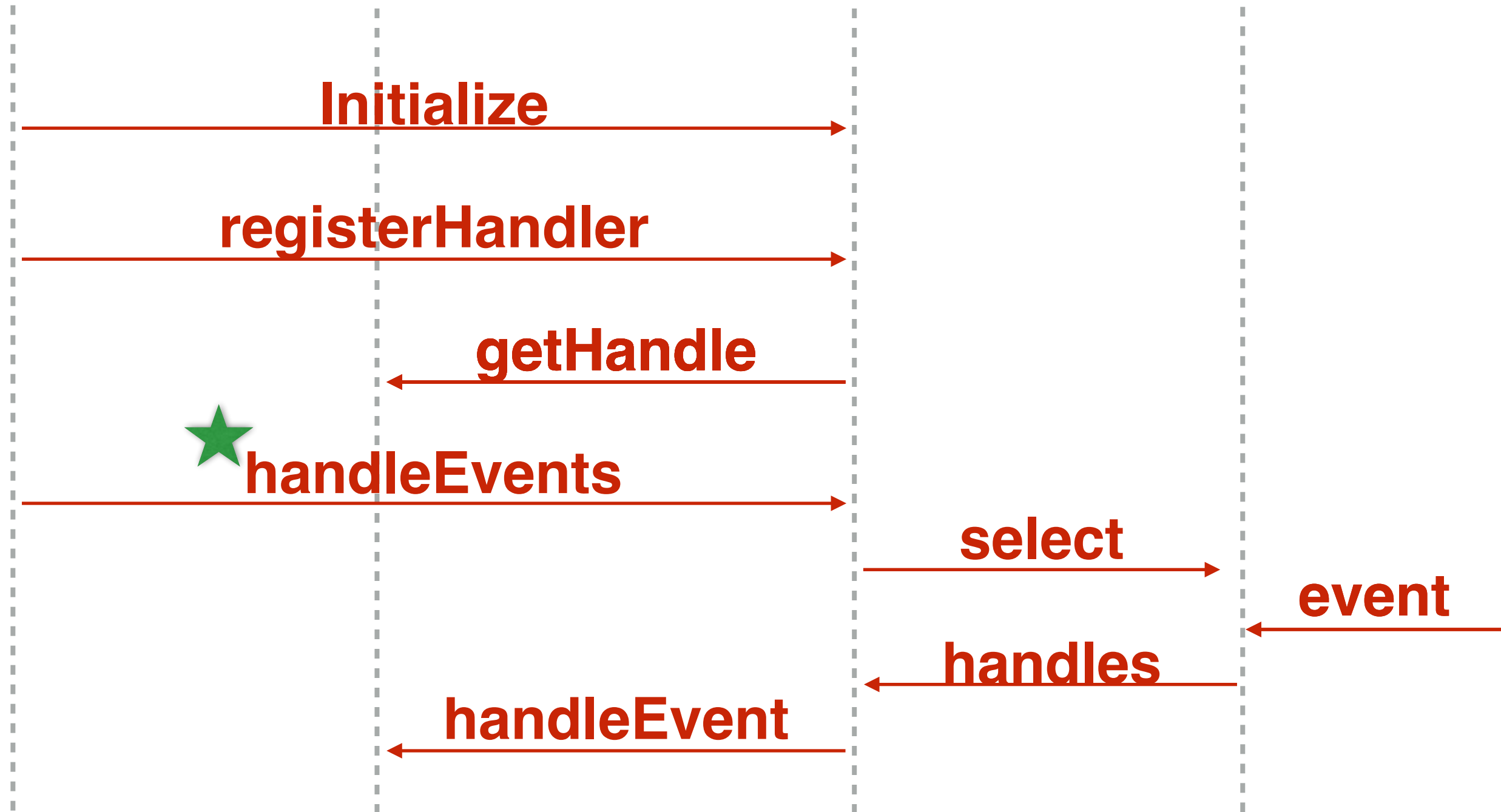
handleEvents

select

event

handles

handleEvent



ServerInitialize에서 Reactor에 handle_events 메시지를 보냅니다
try-catch 문을 달아줘요.

```
/**
 * @brief Reactor를 실행한다.
 * @details Reactor를 초기화하고 이벤트 핸들러를 등록한다. Reactor에 이벤트 핸들링을 명령한다.
 * @return Nothing
 * @exception java.io.IOException 서버 소켓에서 IO 에러 발생 가능.
 */
private void dispatch() {
    reactor = new Reactor();

    reactor.registerHandler(new SayHelloEventHandler());
    reactor.registerHandler(new UpdateProfileEventHandler());

    try {
        reactor.handle_events();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



```
/**
 * @brief Demultiplexer에 select 명령을 한다.
 * @details Demultiplexer에 핸들러들을 전달한다. 반복적으로 수행한다.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void handle_events() throws IOException {
    Demultiplexer demultiplexer = new Demultiplexer();
    while (true) {
        demultiplexer.select(handlers);
    }
}
```

**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize

registerHandler

getHandle

handleEvents



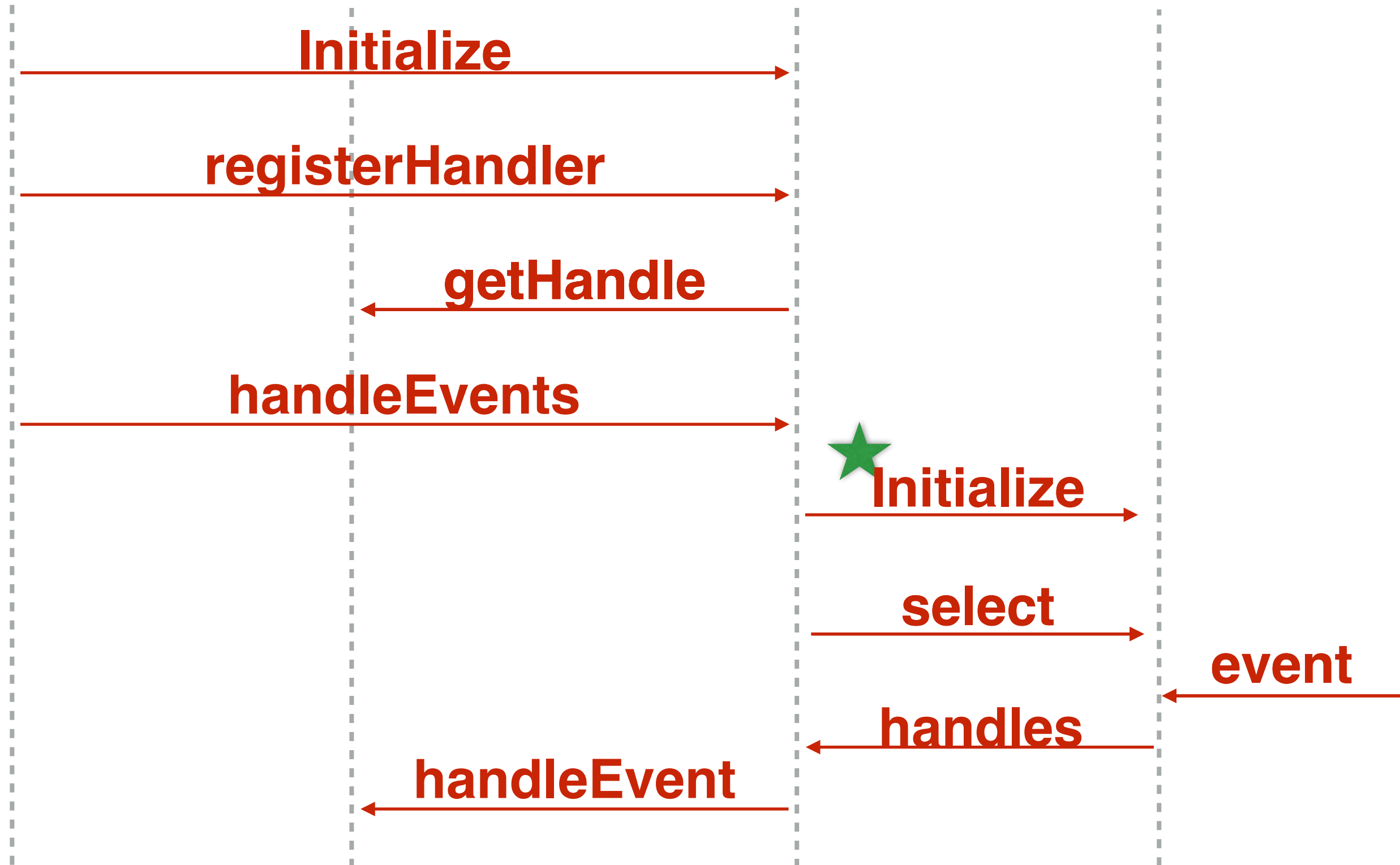
Initialize

select

event

handles

handleEvent



Demultiplexer를 초기화합니다.

```
/**
 * @brief Demultiplexer에 select 명령을 한다.
 * @details Demultiplexer에 핸들러들을 전달한다. 반복적으로 수행한다.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void handle_events() throws IOException {
    Demultiplexer demultiplexer = new Demultiplexer();
    while (true) {
        demultiplexer.select(handlers);
    }
}
```

Demux 초기화

```
/**
 * @brief 이벤트를 디코딩하여 적절한 핸들러에 전달한다.
 * @details 이벤트가 발생하면 header와 data 부분으로 나눈다.
 */
public class Demultiplexer {
    private ServerSocket serverSocket;
    private final int HEADER_SIZE = 6;
```


**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize

registerHandler

getHandle

handleEvents

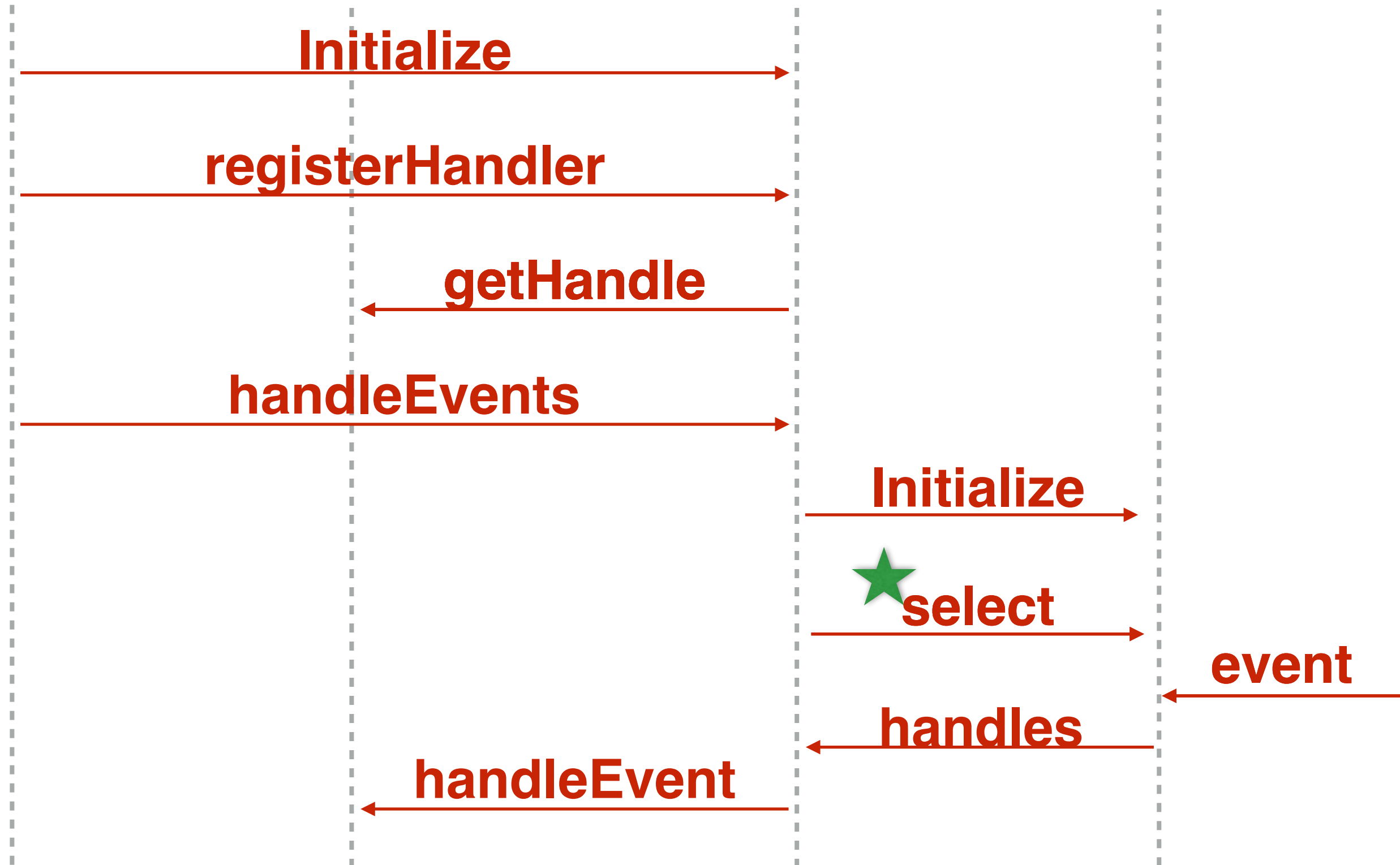
Initialize

 **select**

event

handles

handleEvent



Demultiplexer에 select 메시지를 보냅니다.
무한루프 안에 있어요.

```
/**
 * @brief Demultiplexer에 select 명령을 한다.
 * @details Demultiplexer에 핸들러들을 전달한다. 반복적으로 수행한다.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void handle_events() throws IOException {
    Demultiplexer demultiplexer = new Demultiplexer();
    while (true) {
        demultiplexer.select(handlers);
    }
}
```

select 메시지는 HandleMap을 전달받아요.

```
/**
 * @brief 이벤트 발생시 Demultiplex 한다.
 * @details 이벤트 발생시, header와 data로 나눈다. 이중 header는 적절한 handler를 찾는데 쓰이고, data는 handler에 전달한다.
 * @param handlers Reactor에 등록된 핸들맵 객체.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void select(HandleMap handlers) throws IOException {
    Socket socket = serverSocket.accept();
    InputStream inputStream = socket.getInputStream();

    byte[] buffer = new byte[HEADER_SIZE];
    inputStream.read(buffer);
    String header = new String(buffer);

    handlers.get(header).handleEvent(inputStream);
}
```

**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize

registerHandler

getHandle

handleEvents

Initialize

select



event

handles

handleEvent

Event가 발생할 때 까지 기다립니다.

```
/**
 * @brief 이벤트 발생시 Demultiplex 한다.
 * @details 이벤트 발생시, header와 data로 나눈다. 이중 header는 적절한 handler를 찾는데 쓰이고, data는 handler에 전달한다.
 * @param handlers Reactor에 등록된 핸들맵 객체.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void select(HandleMap handlers) throws IOException {
    Socket socket = serverSocket.accept();
    InputStream inputStream = socket.getInputStream();

    byte[] buffer = new byte[HEADER_SIZE];
    inputStream.read(buffer);
    String header = new String(buffer);

    handlers.get(header).handleEvent(inputStream);
}
```

**Server
Initialize**

**Event
Handler**

Reactor

Demux

Initialize

registerHandler

getHandle

handleEvents

Initialize

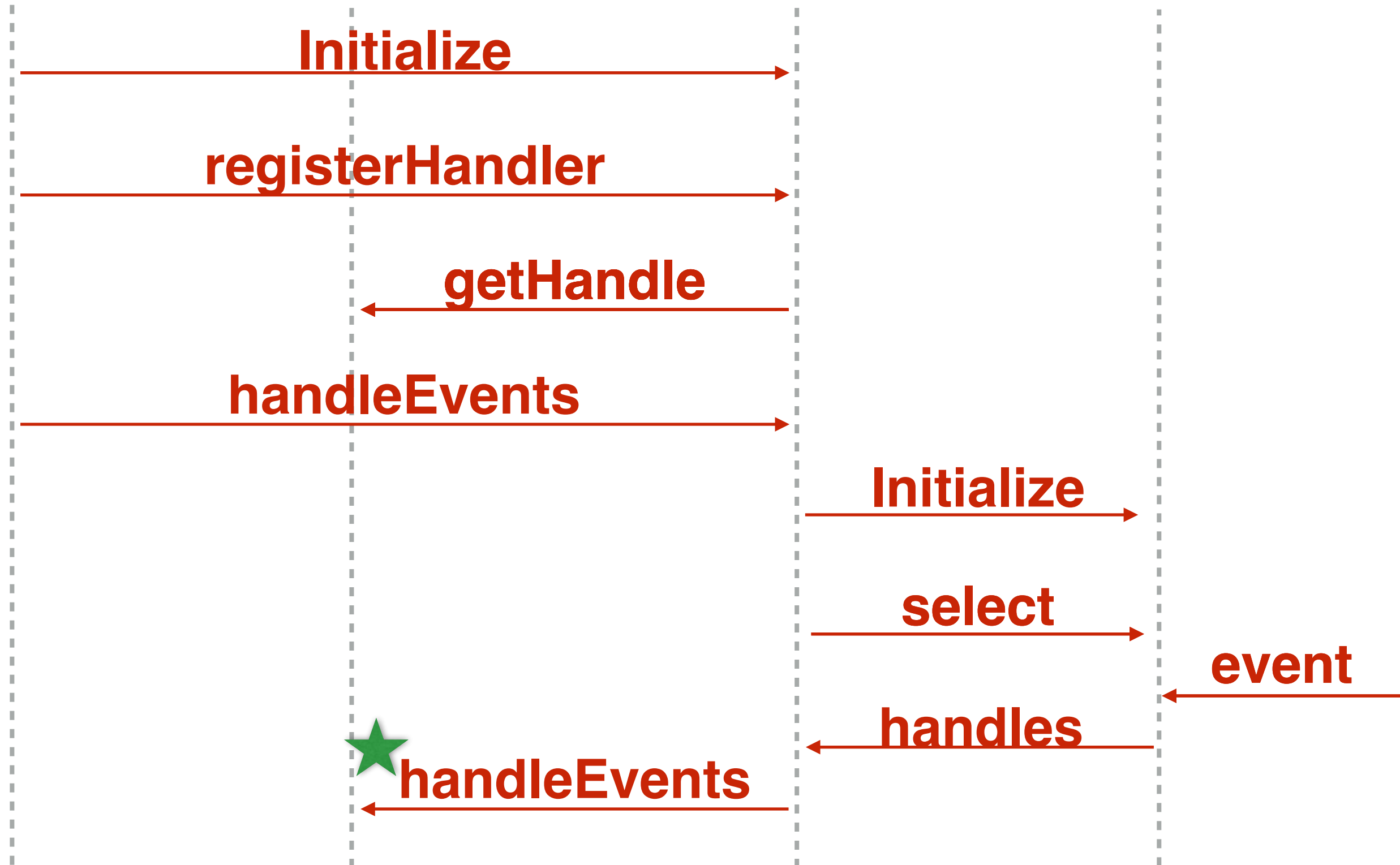
select

event

handles



handleEvents



이벤트 발생시, 헤더부분과 데이터부분을 분리합니다.
정해진 **HEADER_SIZE**만큼 읽어요.

```
/**
 * @brief 이벤트 발생시 Demultiplex 한다.
 * @details 이벤트 발생시, header와 data로 나눈다. 이중 header는 적절한 handler를 찾는데 쓰이고, data는 handler에 전달한다.
 * @param handlers Reactor에 등록된 핸들맵 객체.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void select(HandleMap handlers) throws IOException {
    Socket socket = serverSocket.accept();
    InputStream inputStream = socket.getInputStream();

    byte[] buffer = new byte[HEADER_SIZE];
    inputStream.read(buffer);
    String header = new String(buffer);

    handlers.get(header).handleEvent(inputStream);
}
```

header로 적절한 핸들러를 찾아냅니다.
그 핸들러에 데이터를 전달해요.

```
/**
 * @brief 이벤트 발생시 Demultiplex 한다.
 * @details 이벤트 발생시, header와 data로 나눈다. 이중 header는 적절한 handler를 찾는데 쓰이고, data는 handler에 전달한다.
 * @param handlers Reactor에 등록된 핸들맵 객체.
 * @return Nothing
 * @throws IOException IOException 발생시 던진다.
 */
public void select(HandleMap handlers) throws IOException {
    Socket socket = serverSocket.accept();
    InputStream inputStream = socket.getInputStream();

    byte[] buffer = new byte[HEADER_SIZE];
    inputStream.read(buffer);
    String header = new String(buffer);

    handlers.get(header).handleEvent(inputStream);
}
```


음냐/

```
/**  
 * @brief 핸들러에 데이터를 파싱하여 처리한다.  
 * @details '|' 를 기준으로 data1|data2|... 로 나누어서 처리한다.  
 * @param data 핸들러가 파싱하여 처리할 스트림이다.  
 * @return Nothing  
 */  
public void handleEvent(InputStream data) throws IOException;
```