```
ssm.waitForClientRequest();
```

```
ClientSocketManager csm = new ClientSocketManager(ip, port);

System.out.println("접속 요청 완료!");
System.out.println("가위는 1, 바위는 2, 보는 3");
```

서버측에서 이 과정이
끝난상태에서 클라이언트가
다음과 같은 과정을 진행

이 과정이
accept 진행

```
System.out.println("모두 접속 완료!");

ssm.checkEachIpAddressInfo();
```

서버가 접속이 완료되면 출력 후, ip체크! 이후

```
ssm.recv(ssm.getClntSockArr(), ssm.getMaxClnt());
```
server

```
csm.send(csm.getClntSock());

System.out.println("전송 완료!");
```
client

```
csm.send(csm.getClntSock());

System.out.println("전송 완료!");
```
client

```
ssm.recv(ssm.getClntSockArr(), ssm.getMaxClnt());
```
server

왼쪽처럼 MaiServer.java 위에서 부터 차례대로 진행되는건지.
오른쪽처럼 MainClient.java 쪽으로 한번, 다시 MainServer.java 으로 진행되는지 잘 모르겠습니다.
제가 생각해본건 양쪽 프로세서(?)를 빠르게 진행하면서 'client쪽에서 값을 전송해야 recv할수있다' 생각해 오른쪽이 맞는거 같은데, 그렇다면 client에서 값을 주지 않을동안 accept랑 마찬가지로 블로킹연산(?)인건가요? 아니면 제가 너무 이상한 방향으로 생각하는건가요?

```
csm.send(csm.getClntSock());

System.out.println("전송 완료!");
```

위 내용이 오른쪽으로 진행했다고 가정하고
다음과 같이 진행

```
public void send(Socket sock) throws IOException {
    System.out.print("숫자를 입력하세요: ");
    String str = scan.nextLine();

    out[ZERO] = sock.getOutputStream();
    out[ZERO].write(str.getBytes());
}
```

```
ssm.recv(ssm.getClntSockArr(), ssm.getMaxClnt());
```

```
public void recv(Socket[] sock, int num) throws IOException {
    int tmp;

    for(int i = ZERO; i < num; i++) {
        in[i] = sock[i].getInputStream();

        reader = new BufferedReader(new InputStreamReader(in[i]));

        //...
        tmp = Integer.parseInt(reader.readLine());

        if(tmp == MAGICNUM) {
            arrRockScissorPaper[i] = Integer.toString( i: tmp + ONE);
        } else {
            arrRockScissorPaper[i] = Integer.toString(tmp);
        }

        System.out.println("msg: " + arrRockScissorPaper[i]);
    }
}
```

OR 연산으로 편하게 하기위한 덧셈

받은 값이 arrRSP로 배열되어
msg : arrRSP[i]로 출력

```java
public boolean canWeGetWinner(int num) {
    //...
    int bitOROfAllInputString = ZERO;

    for(int i = ZERO; i < num; i++) {
        bitOROfAllInputString |=
                Integer.parseInt(arrRockScissorPaper[i]);
    }

    if(bitOROfAllInputString == 7) {
        return false;
    } else if(bitOROfAllInputString == 1) {
        return false;
    } else if(bitOROfAllInputString == 2) {
        return false;
    } else if(bitOROfAllInputString == 4) {
        return false;
    }

    return true;
```

승부에 대한 결과 true/false로 아래 내용 출력 후 Client에게 다시 Send [convertNumber2RSP 해서 writer.print(str)]

```java
if(ssm.canWeGetWinner(ssm.getMaxClnt())) {
    System.out.println("승패가 결정되었습니다.");
} else {
    System.out.println("무승부: 게임을 다시 시작합니다.");
}

ssm.send(ssm.getClntSockArr(), ssm.getMaxClnt());
```

```java
public void send(Socket[] sock, int num) throws IOException {
    for(int i = ZERO; i < num; i++) {
        out[i] = sock[i].getOutputStream();

        writer = new PrintWriter(out[i], autoFlush: true);

        String str = convertNumber2RSP();

        writer.println(str);
    }
}
```