# Attività 1.1 (15 min):

Importa il codice su Netbeans o altro IDE e **senza eseguirlo** prova a capire cosa fanno i seguenti metodi:

* attendi();
* connetti() della classe Server;
* comunica();
* comunica() della classe Client.

attendi() (Server) -> il metodo attendi() collega il server alla porta 6789 per potersi collegare con un client: il server accetta la richiesta del client e instanza le variabili per poterci comunicare (DataInputStream e DataOutputStream)

connetti() (Server) -> il server rimane in attesa fino a che il client non spedisce la prima stringa, dove il server manderà un messaggio di conferma di ricezione del messaggio, se viene inserita la stringa “chiudi” viene interrotta la connessione.

connetti() (Client) -> il metodo connetti() collega il client al server sulla stessa porta per poter interagire con esso: come nel metodo attendi() per il server, questo metodo si collega al server e instanza tutte le variabili per poter comunicare con il server .

comunica() (Client) -> questo metodo invia stringhe fino a che non si inserisce la frase “chiudi” dal client al server: in un ciclo while infinito si chiede di mandare delle stringhe al server, che produrra una risposta a sua volta e il client riporterà la risposta dal server; se si inserisce la stringa “chiudi” si chiude la connessione.

**Attività 1.2 (15 min):**

Prima di eseguire il codice, rappresenta graficamente la logica dell’applicazione Client-Server. Riportandola di seguito:

**Attività 2 (10 min):**

Testa l’applicazione allegata:

* La logica che hai rappresentato graficamente nell’attività 1.2, è corretta?

………………………………………………………………………………………………………………

………………………………………………………………………………………………………………

………………………………………………………………………………………………………………

**Domande (15 min):**

Le domande si riferiscono al codice allegato:

1. Qual è o quali sono le differenze tra il metodo attendi() del Server e il metodo connetti() del Client?

………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………….

1. Qual è o quali sono le differenze tra il metodo comunica() del Server e quello del Client?

………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………….

**Attività 3 (20 minuti)**

Una volta eseguito e indagato il codice di base dell'applicazione client-server in Java, è il momento di modificarlo. L'obiettivo di questa attività è di esplorare vari scenari di comunicazione tra client e server, apportando delle modifiche al codice esistente:

1. Modifica il codice per cambiare la porta di comunicazione (ad esempio da 6789 a 1234);
2. Aggiungi un messaggio di benvenuto quando il client si connette (ad esempio, il server invia automaticamente "Benvenuto, client!" al momento della connessione).

**Esercizio (20 min)**

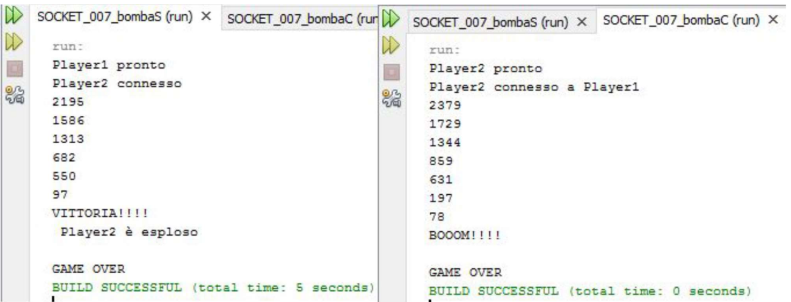
Esegui l’esercizio di seguito riportato:

Progettare un’applicazione client-server in Java per simulare un gioco chiamato "Bomba a tempo". L'obiettivo è evitare che il valore della "bomba" raggiunga zero durante il proprio turno. Chi porta il valore a zero fa esplodere la bomba e perde, mentre l’altro giocatore vince.

Il funzionamento è il seguente:

* Il **server** genera un numero casuale di partenza per la bomba, ad esempio 5000, e lo invia al client;
* **Client e Server** si alternano i turni, sottraendo dal numero della bomba un valore casuale a ogni scambio. Questo valore casuale, ad esempio tra 50 e 100, viene generato a ogni turno dal client e dal server.

Ogni scambio deve essere visualizzato, mostrando il valore aggiornato della bomba. Al termine, un messaggio indica il vincitore e il perdente prima di chiudere la connessione.



**Tabella utile**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metodo di scrittura** | **Descrizione** | **Tipo di dato** | **Metodo di lettura corrispondente** |
| writeBoolean(boolean v) | Scrive un valore booleano come 1 byte (1 per true, 0 per false). | boolean | readBoolean() |
| writeByte(int v) | Scrive un valore byte come 1 byte, usando i primi 8 bit di un int. | byte | readByte() |
| writeShort(int v) | Scrive un valore short di 2 byte in formato big-endian. | short | readShort() |
| writeChar(int v) | Scrive un carattere Unicode di 2 byte in formato big-endian. | char | readChar() |
| writeInt(int v) | Scrive un valore intero di 4 byte in formato big-endian. | int | readInt() |
| writeLong(long v) | Scrive un valore long di 8 byte in formato big-endian. | long | readLong() |
| writeFloat(float v) | Scrive un valore float di 4 byte in formato IEEE 754 (big-endian). | float | readFloat() |
| writeUTF(String s) | Scrive una stringa come testo UTF-8, con la lunghezza iniziale di 2 byte seguita dai caratteri UTF-8. | String | readUTF() |
| writeDouble(double v) | Scrive un valore double di 8 byte in formato IEEE 754 (big-endian). | double | readDouble() |
| writeBytes(String s) | Scrive una stringa come una sequenza di byte (un byte per carattere), senza lunghezza prefissata o codifica. | String | readFully(byte[] b) |
| writeChars(String s) | Scrive una stringa come una sequenza di caratteri Unicode, dove ogni carattere occupa 2 byte. | String | readChar() per ogni carattere |
| write(byte[] b) | Scrive un array di byte senza aggiungere lunghezza o delimitatori. | byte[] | readFully(byte[] b) |
| write(byte[] b, int off, int len) | Scrive una porzione specifica di un array di byte, partendo dall'offset off e scrivendo len byte. | byte[] | readFully(byte[] b, int off, int len) |