

```
/* Svolgimento Compito 4 - 21/12/05 */
```

```
import java.io.DataInputStream;

// Thread lanciato per ogni richiesta accettata
// versione per il trasferimento di file binari
class Server_thread extends Thread {

    private Socket clientSocket = null;

    private Stanza[] s;

    // costruttore
    public Server_thread(Socket clientSocket, Stanza[] s) {
        this.clientSocket = clientSocket;
        this.s = s;
    }

    /**
     * Main invocabile con 0 o 1 argomenti. Argomento possibile -> porta su cui il
     * server ascolta.
     */
    public void run() {
        DataInputStream inSock;
        DataOutputStream outSock;
        String servizio;
        String stanza;

        try {
            // creazione stream di input/output da socket
            inSock = new DataInputStream(clientSocket.getInputStream());
            outSock = new DataOutputStream(clientSocket.getOutputStream());

            // lettura tipo di servizio richiesto
            servizio = inSock.readUTF();

            if (!servizio.equals("V") && !servizio.equals("S")) {
                // se ricevo una richiesta che non riesco a gestire chiudo in
                // modo anomalo
                clientSocket.close();
                return;
            } else {
                if (servizio.equals("V")) {
                    // chiudo in input perche' non mi serve
                    clientSocket.shutdownInput();
                    // invio il numero di righe
                    outSock.writeInt(s.length);
                    // spedisco la struttura dati
                    for (int i = 0; i < s.length; i++)
                        outSock.writeUTF(s[i].toString());
                    // temrinate le scritture; chiudo anche in output
                    clientSocket.shutdownOutput();
                } else {
                    stanza = inSock.readUTF();
                    // chiudo in input perche' ho finito di leggere
                    clientSocket.shutdownInput();
                    outSock.writeInt(Server.sospendi(stanza));
                    // temrinate le scritture; chiudo anche in output
                    clientSocket.shutdownOutput();
                }
            } // else
        } // try
        catch (IOException ioe) {
            System.out
                .println("Problemi nella creazione degli stream di input/output ")
        }
    }
}
```

```

        + "su socket: ");
ioe.printStackTrace();
// il server continua l'esecuzione riprendendo dall'inizio del ciclo
return;
}
catch (Exception e) {
    System.out
        .println("Problemi nella creazione degli stream di input/output "
            + "su socket: ");
    e.printStackTrace();
    // il server continua l'esecuzione riprendendo dall'inizio del ciclo
    return;
}

} // run

} // Server_stream_thread

public class Server {
    private static final int N = 10;

    static Stanza s[] = null;

    public static synchronized int sospendi(String nomeStanza) {
        for (int i = 0; i < N; i++)
            if (s[i].getNome().equals(nomeStanza)) {
                if (s[i].sospendi())
                    return 0;
                else
                    return -1;
            }
        return -1;
    }

    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket clientSocket = null;
        int port = -1;

        // Controllo argomenti
        try {
            if (args.length == 1) {
                // Controllare anche porta!!
                port = Integer.parseInt(args[0]);
            } else {
                System.out.println("Usage: java Server port");
                System.exit(1);
            }
        } //try
        catch (Exception e) {
            System.out.println("Problemi, i seguenti: ");
            e.printStackTrace();
            System.out.println("Usage: java Server port");
            System.exit(2);
        }

        // inizializzazione struttura dati
        s = new Stanza[N];
        for (int i = 0; i < N; i++) {
            s[i] = new Stanza();
        }
        s[0].setNome("Informatica");
        s[0].setStato("M");
        s[1].setNome("Ambiente");
        s[1].setStato("SM");
        s[2].setNome("Cucina");
    }
}

```

```

s[2].setStato("P");
s[3].setNome("Motori");
s[3].setStato("SP");

System.out.println("Stanze inizializzate");

try {
    serverSocket = new ServerSocket(port);
    serverSocket.setReuseAddress(true);
    System.out.println("Server: avviato ");
    System.out.println("Server: creata la server socket: " + serverSocket);
}
catch (Exception e) {
    System.err
        .println("Server: problemi nella creazione della server socket: "
            + e.getMessage());
    e.printStackTrace();
    System.exit(1);
}

try {
    while (true) {
        System.out.println("Server: in attesa di richieste...\n");

        try {
            // bloccante finche' non avviene una connessione
            clientSocket = serverSocket.accept();
            clientSocket.setSoTimeout(60000);
            // qui non e' piu' cosa indispensabile, ma e' comunque meglio
            // evitare che un thread si blocchi indefinitamente
            System.out.println("Server: connessione accettata: " + clientSocket);
            new Server_thread(clientSocket, s).start();
        }
        catch (Exception e) {
            System.err
                .println("Server: problemi nella accettazione della connessione: "
                    + e.getMessage());
            e.printStackTrace();
            continue;
            // il server continua a fornire il servizio ricominciando dall'inizio
            // del ciclo
        }
    } // while
}

// qui catturo le eccezioni non catturate all'interno del while
// in seguito alle quali il server termina l'esecuzione
catch (Exception e) {
    e.printStackTrace();
    // chiusura di stream e socket
    System.out.println("PutFileServerCon: termino...");
    System.exit(2);
}

}
} // Server_stream

```