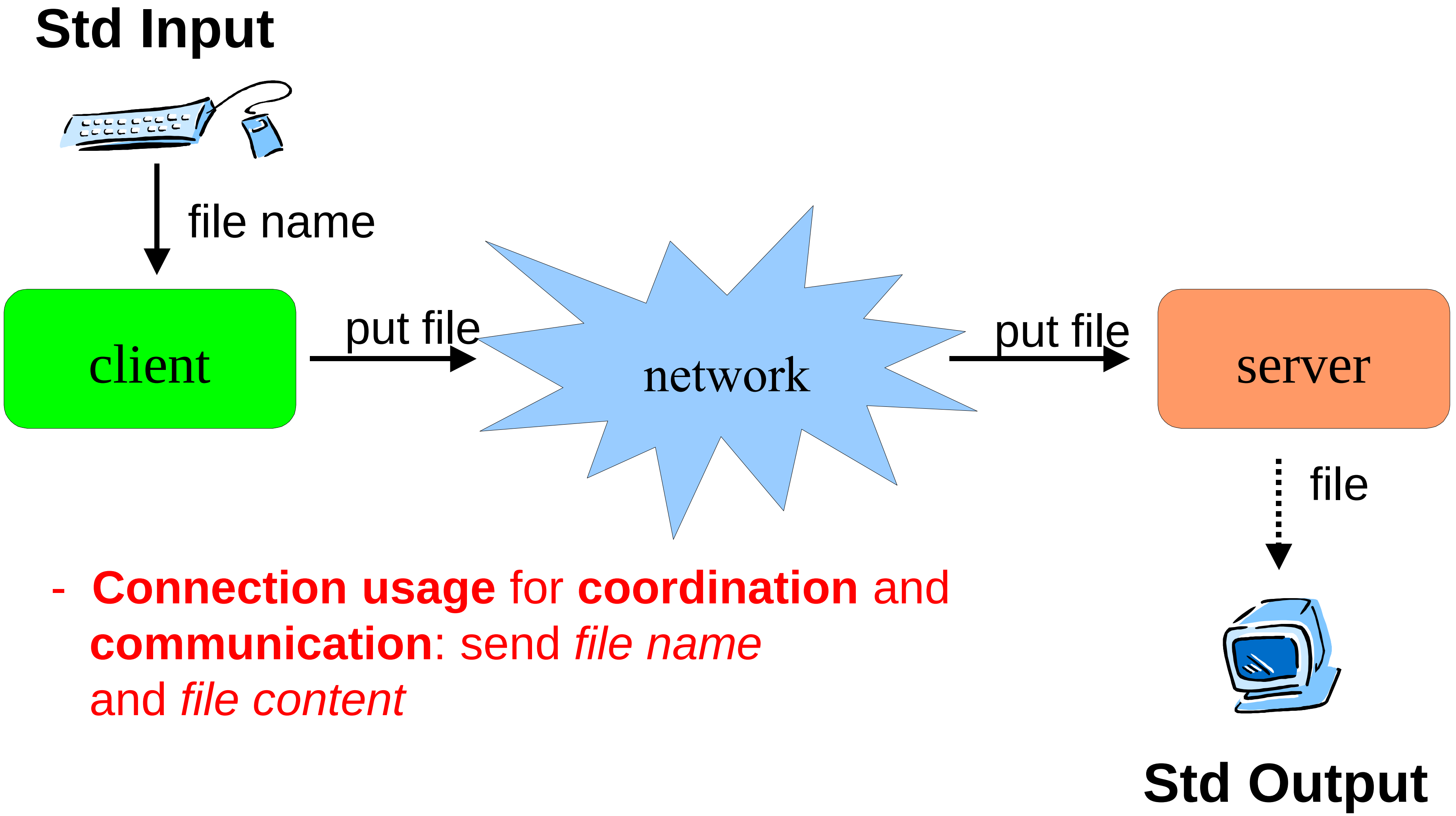
Università degli Studi di Bologna
Facoltà di Ingegneria

Principles, Models, and Applications for Distributed Systems M

Lab assignment 4 (worked-out)
Connection-oriented Java Sockets

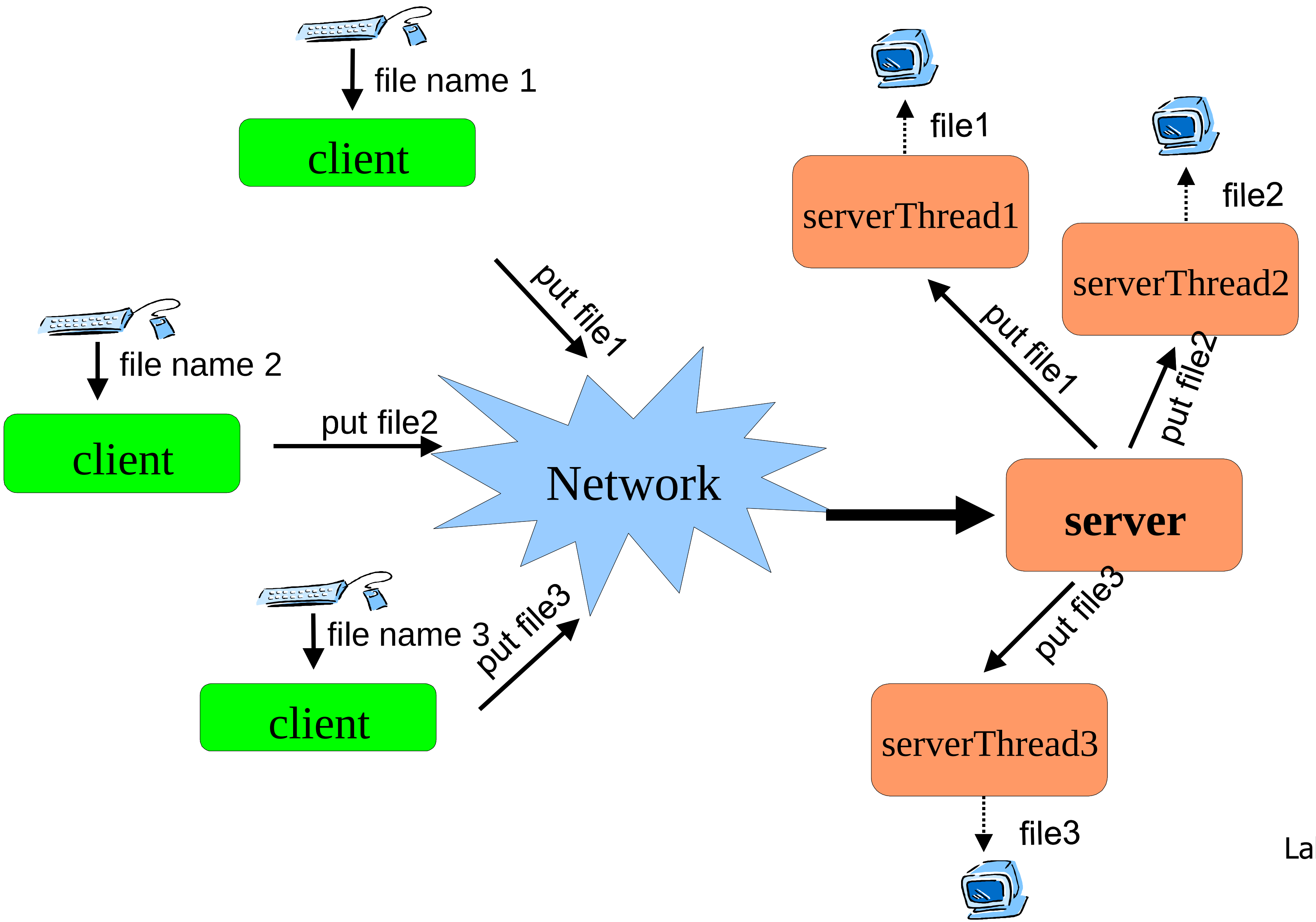
Luca Foschini

Distributed architecture for the file transfer: Sequential Server



- **Connection usage** for **coordination** and **communication**: send *file name* and *file content*

Distributed architecture for the file transfer: Concurrent Server



Implementation details

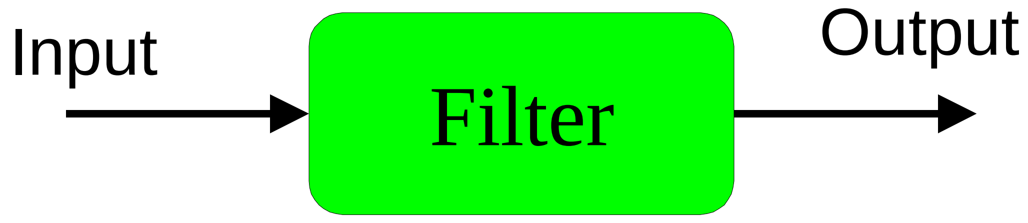
Develop a C/S application to **transfer a binary file** from client to server (**put**).

The **Client** asks the user the file name of the file to transfer; then, it connects to the server (using stream sockets: **`java.net.Socket`**), builds an **output stream** over the socket connection, and sends the **file name** and the **file content**. Finally, it waits and prints the server reply (success/failure) and prepares to receive other user requests until EOF.

The **Server** listens for connection requests (using **`java.net.ServerSocket`** class), accepts incoming connection request (**`java.net.Socket`**), and builds an **input stream** on the connected socket, that it uses to read the **file name** and the **file content**, by **saving the file in its local file system** (current working directory). Then, it sends a reply and closes the connection. There are two possible responses: **file overwriting** (if the file already exists) and **file creation** (if it has been created) and each of them can be either successful or not.

Filter

A filter is an **application** thate **consumes the input stream** and **writes on the output stream**



Filters can be combined to build a **pipeline**.

For example a filter could **read until the End Of File (EOF) an input stream** and **copy read input data to its output stream** (see the following slides).

Many filter types: **character/line/byte/… oriented**.

In the following we use a simple **line-oriented filter**: **`SimpleFilter`**

then, we use a **byte-oriented filter**: **`byte_file_transfer`**

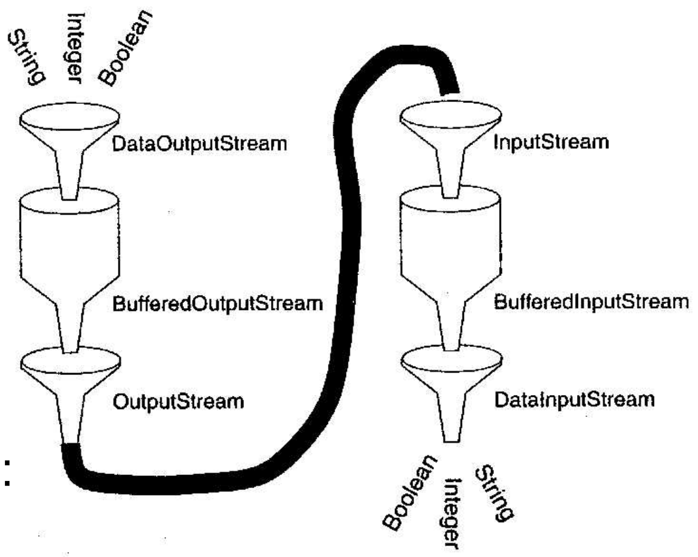
Line-Oriented Filter

This filter **reads lines from standard input**, and **writes to the standard output only those lines that contain the ‘a’ character :**

```
public class SimpleFilter {
  public static void main(String[] args) {
    String line;
    BufferedReader input =
      new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter output =
      new BufferedWriter(new OutputStreamWriter(System.out));
    System.err.println("\nSimpleFilter message:");
    try {
     while ((line = input.readLine()) != null)
        if (line.lastIndexOf('a') > 0) output.write(line + "\n");
     output.flush(); // flush buffer
    }
    catch (IOException e) {
      System.out.println("Error: ");
      e.printStackTrace();
    }
  }
}
```

Filters and Streams

Input/output streams as filters…



Examples of input/output streams wrapping a **socket**:

```
DataInputStream inSock =
  new DataInputStream(socket.getInputStream());
DataOutputStream outSock =
  new DataOutputStream(socket.getOutputStream());
```

Examples of input/output streams wrapping a **binary file**:

```
DataInputStream inSock =
  new DataInputStream(new FileInputStream(filename));
DataOutputStream outSock =
  new DataOutputStream(new FileOutputStream(filename));
```

FileUtility: Byte-oriented filter for file transfer

```
// Static method: byte_file_transfer

static protected void byte_file_transfer
    (DataInputStream src,
     DataOutputStream dest) throws IOException {

    // loop: read from src and write to dest
    int buffer = 0;
    try {
        // break from loop if we read
        // a negative value -> EOF
        while ( (buffer = src.read()) >= 0)
            dest.write(buffer);
        dest.flush();
    }
    catch (EOFException e) {
        System.out.println("Error: ");
        e.printStackTrace();
    }
}
```

Solution sketch: Client (1)

1. Create a socket with **implicit bind** (and set options, if needed):

```
socket = new Socket(addr, port);
socket.setxxx(…);
```

2. User interaction:

```
BufferedReader stdIn = new BufferedReader(new
     InputStreamReader(System.in));
System.out.print("Insert a file name: ");
String filename = null;
while( filename=stdIn.readLine() )!=null)
```

3. Wrap socket output stream:

```
outSock =
  new DataOutputStream(socket.getOutputStream());
```

Solution sketch: Client (2)

4. Wrap binary file in a input stream:

```
inFile =
  new DataInputStream(new FileInputStream(filename));
```

5. Send data to server:

```
outSock.writeUTF(filename);
FileUtility.byte_file_transfer(inFile, outSock);
```

6. Graceful socket close and receive response:

```
socket.shutdownOutput();
...
response = inSock.readUTF();
socket.shutdownInput();
```

Solution sketch: Server (1)

1. Create a socket with **explicit bind** (and set options, if needed):

```
serverSocket = new ServerSocket(port);
serverSocket.setReuseAddress(true);
```

2. Wait for a connection request and accept it:

```
clientSocket = serverSocket.accept();
```

3. Wrap socket input stream:

```
inSock =
new DataInputStream(clientSocket.getInputStream());
```

4. Wrap binary file in an output stream:

```
filename=inSock.readUTF();
outFile = new DataOutputStream(
                      new FileOutputStream(filename));
```

Solution sketch: Server (2)

5. Receive data from client and write to output file:

```
FileUtility.byte_file_transfer(inSock,outFile);
```

6. Close file and socket, send response:

```
outFile.close();
socket.shutdownInput();
...
outSock.writeUTF(response);
socket.shutdownOutput();
```

Remember to close always all sockets and files no longer in use!

PutFileClient for binary files 1/3

```
public class PutFileClient {

public static void main(String[] args) throws IOException {
  InetAddress addr = null;
  int port = -1;
  try{   // check invocation arguments
    if(args.length == 2){
      addr = InetAddress.getByName(args[0]);
      port = Integer.parseInt(args[1]);
    } else{ System.out.println("Usage: ..."); System.exit(1); }
  } //try
  catch(Exception e){ ... }

  // objects for network communication and file access
  Socket socket = null;
  FileInputStream inFile = null; String filename = null;
  DataInputStream inSock = null; DataOutputStream outSock = null;
  BufferedReader stdIn =
    new BufferedReader(new InputStreamReader(System.in));
  System.out.print("\n^D(Unix)/^Z(Win)+enter …. File name?");
```

PutFileClient for binary files2/3

```
try{
   while ((filename=stdIn.readLine())!=null){
    if(new File(filename).exists()){
    try{  // socket creation
        socket = new Socket(addr, port);
        socket.setSoTimeout(30000);
        inSock = new DataInputStream(socket.getInputStream());
        outSock = new DataOutputStream(socket.getOutputStream());
      } catch(Exception e){... continue;}
    }
    else{System.out.println("File " + filename + " does not exist");
    System.out.print("\n^D(Unix)/^Z(Win)...");  continue;
   }
   // Send file
   try{ inFile = new FileInputStream(filename); }
   catch(FileNotFoundException e){…}
```

PutFileClient for binary files 3/3

```
  try{
    outSock.writeUTF(filename);
    FileUtility.byte_file_transfer(
                         new DataInputStream(inFile), outSock);
    inFile.close();     // close file
    socket.shutdownOutput(); // close socket output stream, i.e., send EOF
  }
  catch (SocketTimeoutException te) {... continue;}
  catch(Exception e){... continue;}
  String response;  // read reply
  try{
    response = inSock.readUTF();
    socket.shutdownInput(); // close socket input stream
  }
  catch (SocketTimeoutException te) {... continue;}
  catch(Exception e){... continue;}
  System.out.print("\n^D(Unix)/^Z(Win)...");  // new request
 }  // while
}  // try
catch(Exception e){... System.exit(3);}
}  // main
}  // class
```

PutFileServerSeq for binary files1/3

```
public class PutFileServerSeq {
public static final int PORT = 54321; // default port

public static void main(String[] args)
throws IOException {
  int port = -1;
  try // check arguments
  { if (args.length == 1) {
      port = Integer.parseInt(args[0]);
    } else if (args.length == 0) {
      port = PORT;
    } else { // Msg errore...  }
  } //try
  catch (Exception e) {...}
  ServerSocket serverSocket = null; // build socket and in/out streams
  try
 { serverSocket = new ServerSocket(port);
    serverSocket.setReuseAddress(true);
  }
  catch (Exception e) {...}
  try
  { while (true)   // server endless loop
    { Socket clientSocket = null;
      DataInputStream inSock = null;  DataOutputStream outSock = null;
```

PutFileServerSeq for binary files 2/3

```
try
{ clientSocket = serverSocket.accept();
  clientSocket.setSoTimeout(30000);
}
catch (Exception e) {... continue;}
String filename;
try // build I/O streams
{ inSock =new DataInputStream(clientSocket.getInputStream());
  outSock =new DataOutputStream(clientSocket.getOutputStream());
  filename = inSock.readUTF();
}
catch (SocketTimeoutException te) {... continue;}
catch (IOException e) {... continue;}
FileOutputStream outFile = null; String response; // receive file
if (filename == null) { clientSocket.close(); continue;}
else {
  File curFile = new File(filename);
  if (curFile.exists()) {
    try
    { response = "File overwritten";
      curFile.delete(); // delete the file
    } catch (Exception e) {... continue;}
```

PutFileServerSeq for binary files 3/3

```
} else response = "New file created";
       outFile = new FileOutputStream(filename);
}
 try  // receive file
    { FileUtility.byte_file_trasfer(
      inSock, new DataOutputStream(outFile));
                  // Note: the function above consumes the EOF
     outFile.close();  // close file
     clientSocket.shutdownInput();
     outSock.writeUTF(response+", file saved on server");
     clientSocket.shutdownOutput();
    }
     catch (SocketTimeoutException te) {... continue;}
     catch (Exception e) {...continue;}
   }
  }
  catch (Exception e) {... System.exit(3);}
 }
}
```

PutFileServerCon for binary files 1/4

```
class PutFileServerThread extends Thread {
private Socket clientSocket = null;
public PutFileServerThread(Socket clientSocket)
  { this.clientSocket = clientSocket; }

public void run() // Child process that manages a connection
{ DataInputStream inSock;
  DataOutputStream outSock;
  try
  { String filename;
    try // build stream
    { inSock = new DataInputStream(clientSocket.getInputStream());
      outSock = new DataOutputStream(clientSocket.getOutputStream());
      filename = inSock.readUTF();
    }
    catch (SocketTimeoutException te) {...}
    catch (IOException ioe) {...} catch (Exception e) {...}
    FileOutputStream outFile = null; String response;
    // receive file: error
    if (filename == null) {clientSocket.close(); return; }
```

PutFileServerCon for binary files 2/4

```
    else {  // checks if file exsists
      File curFile = new File(filename);
      if (curFile.exists()) {
        try  // delete old file
        { response = "File overwritten"; curFile.delete(); }
        catch (Exception e) {... return;}
      } else response = "New file created";
      outFile = new FileOutputStream(filename);
    }
    try {
     FileUtility.byte_file_transfer
          (inSock, new DataOutputStream(outFile));
     outFile.close();  // close file and socket
      // IMPORTANT: socket closed by child process
     clientSocket.shutdownInput();
     outSock.writeUTF(response + ", file saved");
     clientSocket.shutdownOutput();
    }
    catch (Exception e) {...}
  } catch (Exception e) {... System.exit(3);}
 }  // run
}  // PutFileServerThread
```

PutFileServerCon for binary files 3/4

```
public class PutFileServerCon {
  public static final int PORT = 1050;
public static void main (String[] args) throws IOException {
  int port = -1;
  try // check arguments
  { if (args.length == 1) {port = Integer.parseInt(args[0]); }
    else if (args.length == 0) {port = PORT; }
    else { System.out.println("Usage: ...");  System.exit(1); }
  } //try
  catch (Exception e) {... System.exit(1);}
  ServerSocket serverSocket = null; Socket clientSocket = null;
  try {
    serverSocket = new ServerSocket(port);
serverSocket.setReuseAddress(true);
  }
  catch (Exception e) {... System.exit(1);}
  try {
    while (true) {
      try
      { clientSocket = serverSocket.accept();
        clientSocket.setSoTimeout(30000);
      } catch (Exception e) {... continue;}
```

PutFileServerCon for binary files 4/4

```
try {  // service delegated to a new thread
 new PutFileServerThread(clientSocket).start();

/* IMPORTANT!!! The socket is closed ONLY by the child process. If the father
tries to close the socket you could incur in race conditions (due to shared
memory).
*/
      }
      catch (Exception e) {... continue;}
    }  // while
  }
  catch (Exception e) {... System.exit(2);}
}  // main
}  // PutFileServerCon
```