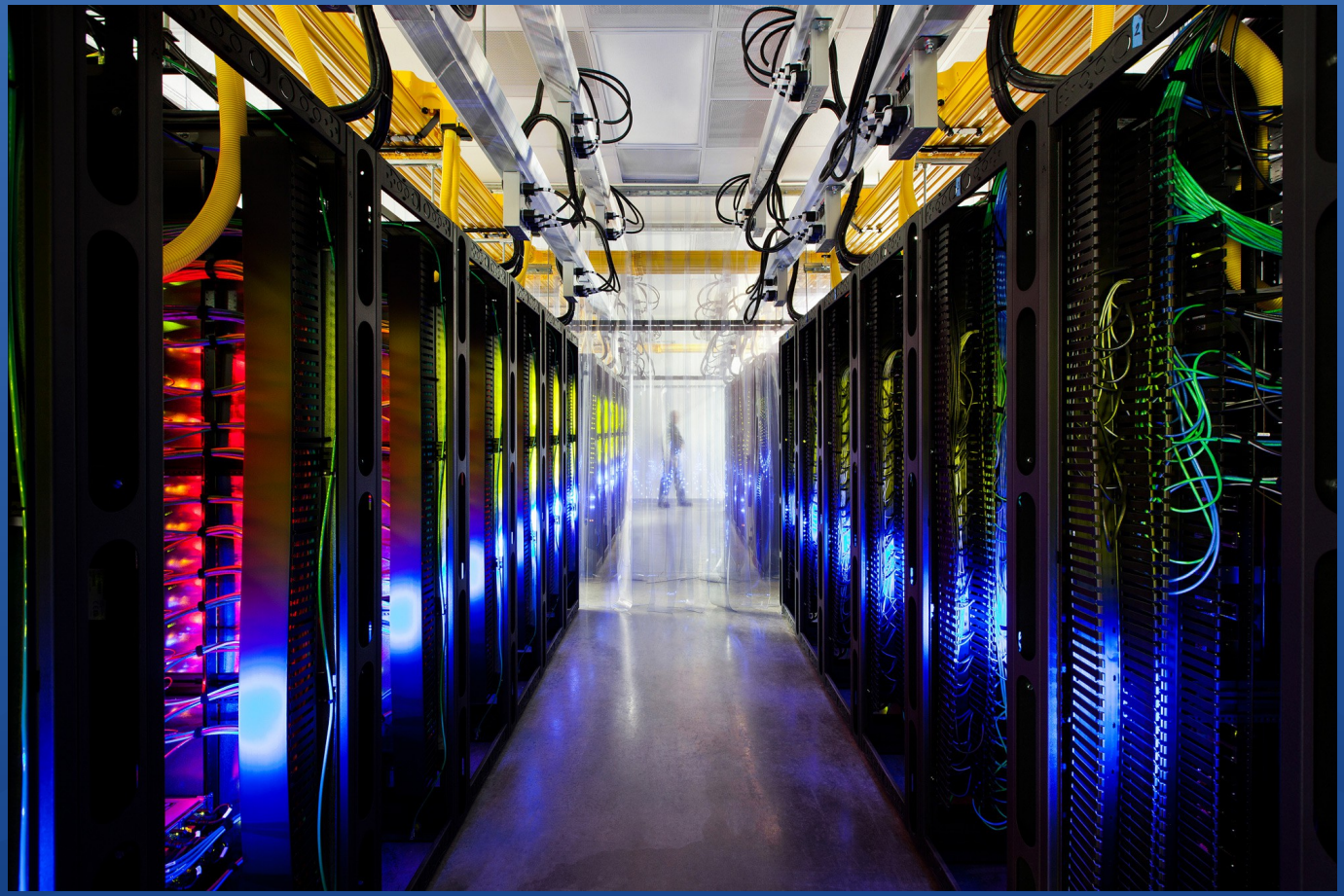
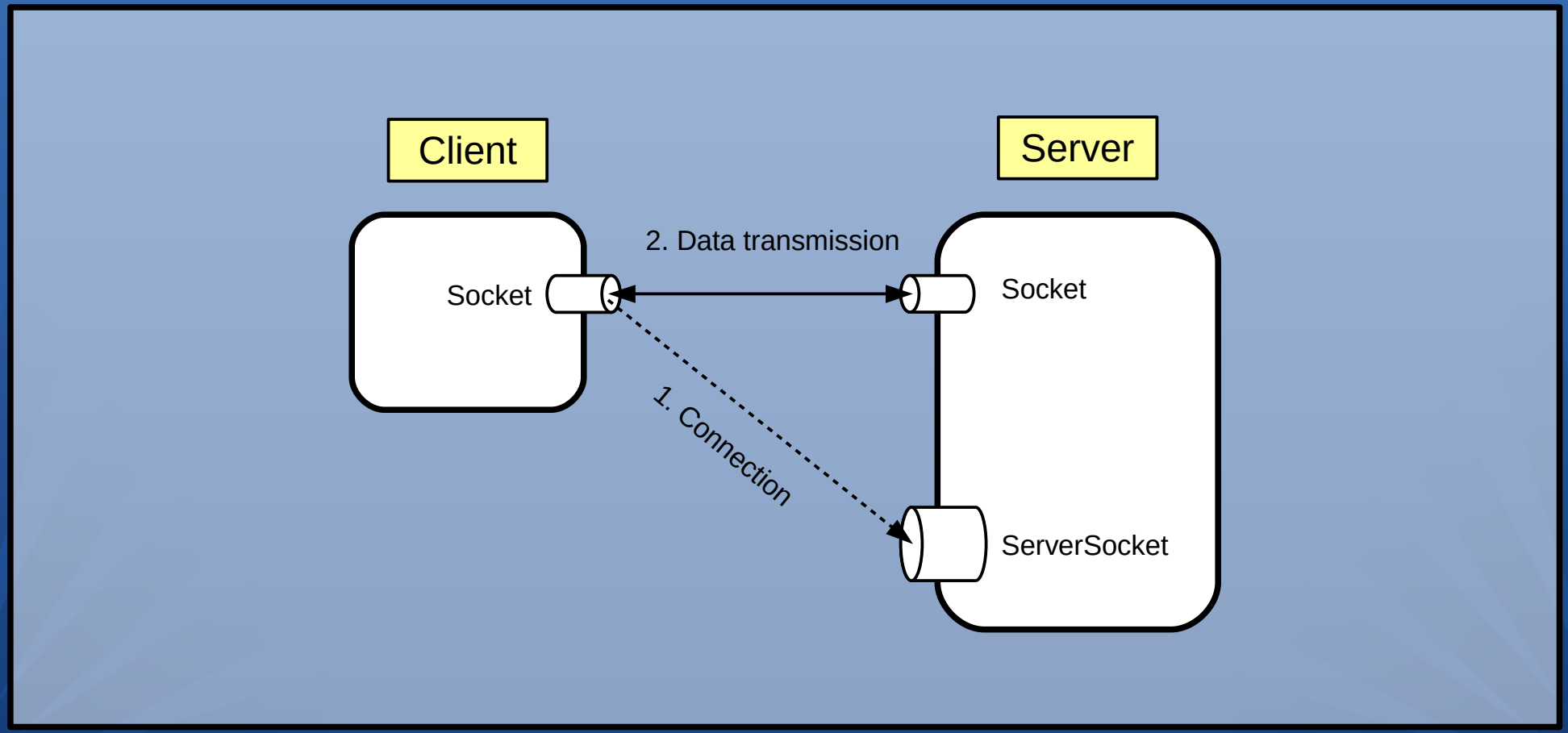
Iterative Servers



Iterative TCP Servers



ServerSocket class

- `public ServerSocket(int port) throws IOException`
  - A `ServerSocket` is opened at `port`
  - `If (port == 0){` A random port is chosen `}`
- `public ServerSocket(int port, int backlog) throws IOException`
  - A `ServerSocket` is opened at `port` with a queue of `backlog` requests
  - `If (queue is full) {` Connection is rejected `}`
- `public Socket accept() throws IOException`
  - It accepts connections (blocking call)
  - A new `Socket` object is created and `ServerSocket` keeps listening for incoming connections
- `public void close() throws IOException`

An iterative tcp server

```
import java.net.*;
import java.io.*;

class TCPServer {
  public static void main(String args[]){
    try {
      ServerSocket ss = new ServerSocket(7777);
      int clientID = 0;
      while(true) {
        Socket s = ss.accept();  //It waits for a new incoming conn.
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        out.println("Client ID: " + clientID++);
        s.close();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

An iterative tcp server

```
import java.net.*;
import java.io.*;

class TCPServer {
  public static void main(String args[]){
    try {
      ServerSocket ss = new ServerSocket(7777);
      int clientID = 0;
      while(true) {
        Socket s = ss.accept();  //It waits for a new incoming conn.
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        out.println("Client ID: " + clientID++);
        s.close();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

An iterative tcp server

```
import java.net.*;
import java.io.*;

class TCPServer {
  public static void main(String args[]){
    try {
      ServerSocket ss = new ServerSocket(7777);
      int clientID = 0;
      while(true) {
        Socket s = ss.accept();  //It waits for a new incoming conn.
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        out.println("Client ID: " + clientID++);
        s.close();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

An iterative tcp server

```
import java.net.*;
import java.io.*;

class TCPServer {
  public static void main(String args[]){
    try {
      ServerSocket ss = new ServerSocket(7777);
      int clientID = 0;
      while(true) {
        Socket s = ss.accept();  //It waits for a new incoming conn.
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        out.println("Client ID: " + clientID++);
        s.close();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

An iterative tcp server

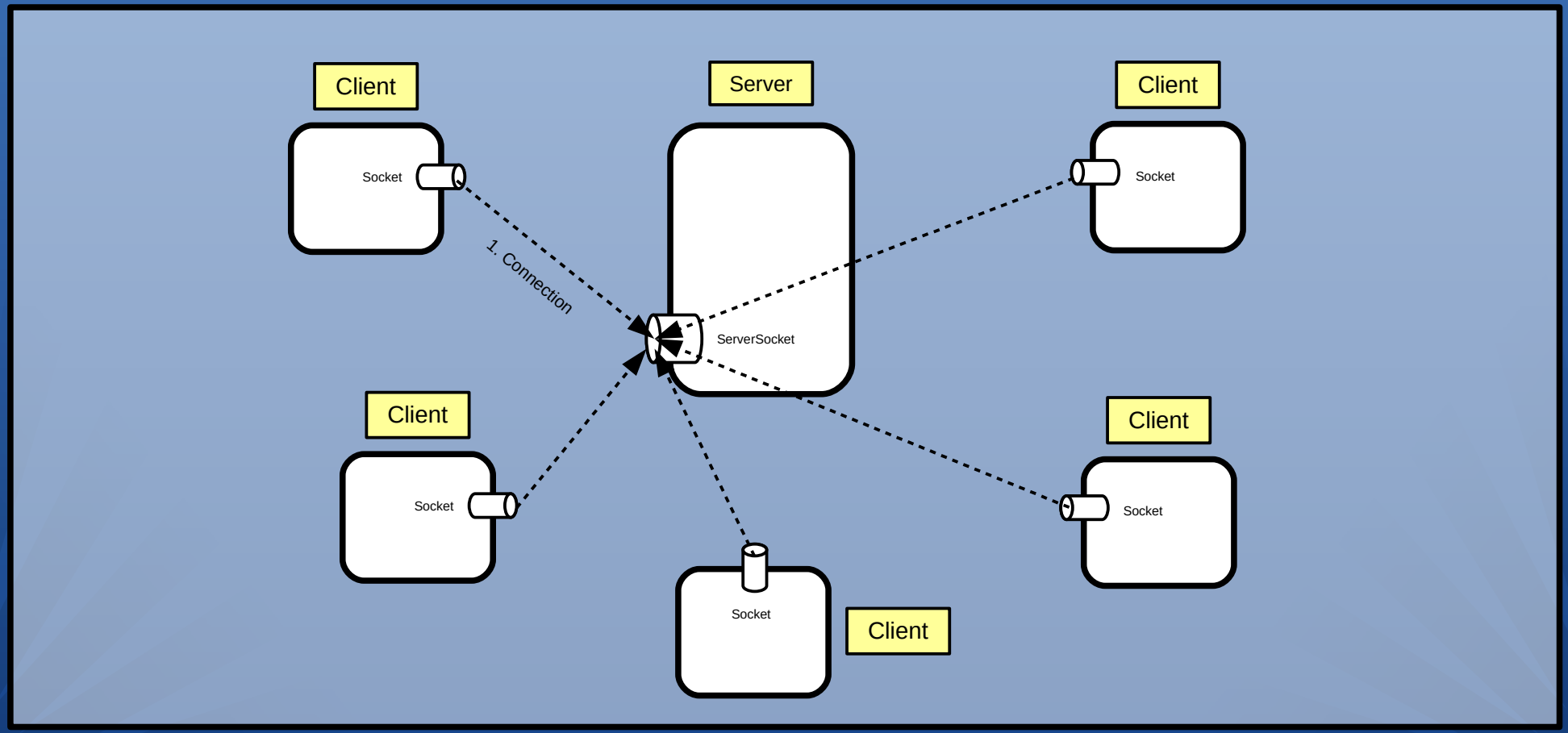
```
import java.net.*;
import java.io.*;

class TCPServer {
  public static void main(String args[]){
    try {
      ServerSocket ss = new ServerSocket(7777);
      int clientID = 0;
      while(true) {
        Socket s = ss.accept();  //It waits for a new incoming conn.
        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        out.println("Client ID: " + clientID++);
        s.close();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Concurrent TCP Servers

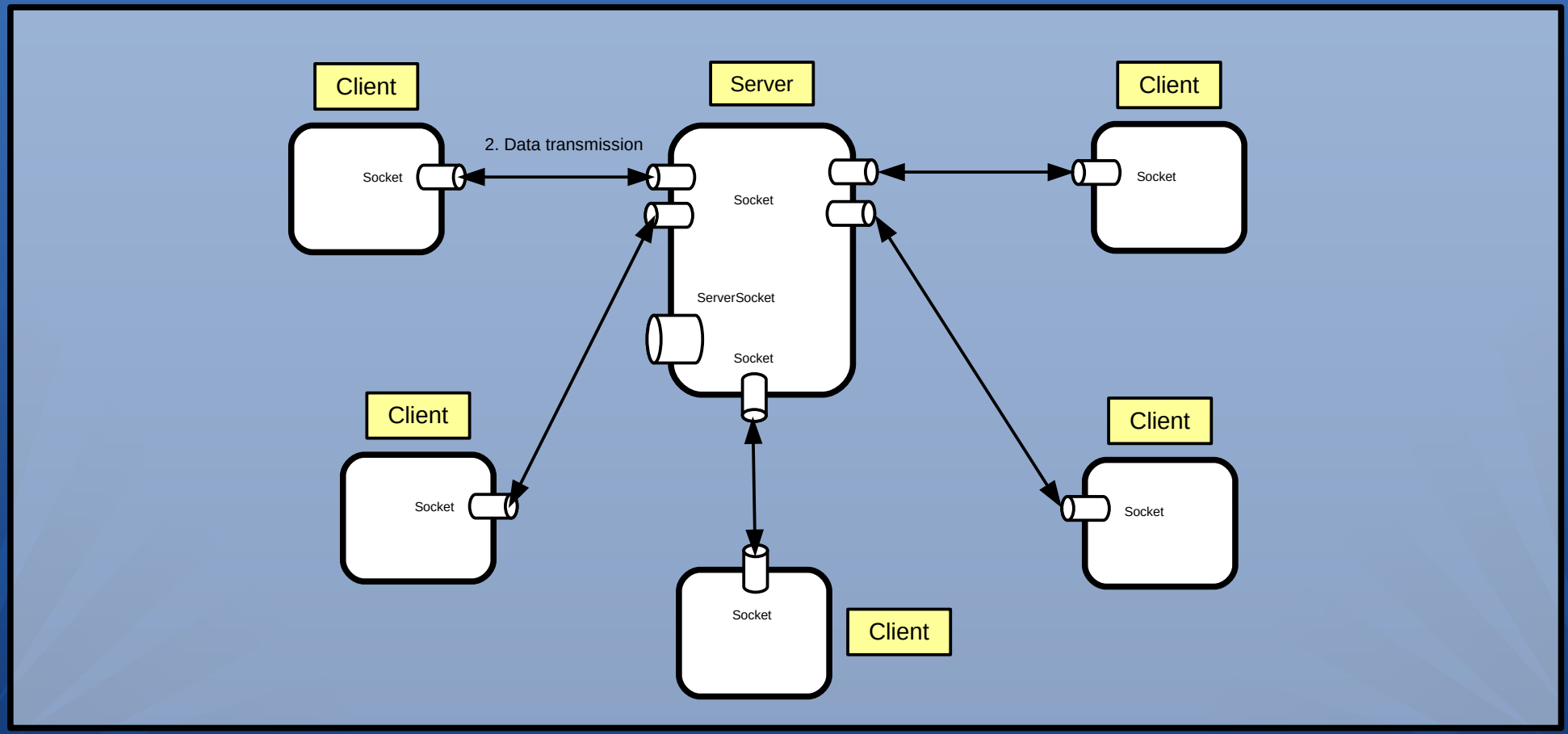
- Concurrent server can serve more than 1 client at time
- It creates a new process to attend to each new client
- The code that implements the server should be in the run method of an extended Thread class
- The main method receives the client requests.

  When a request is accepted, a "child" thread is spawned. The new thread will serve to that client

Concurrent TCP Servers



Concurrent TCP Servers



Threads

- Our Hilos class extends Thread class
- Our Hilos class constructor
- The code that implements the function provided by the Hilos class should be in the **run method** of an extended Thread class
- The "child" thread is spawned from the main method. This new thread will be a new process.

```
class Hilos extends Thread {
 int id;
 public Hilos(int i)
 {
    id=i;
 }

 public void run() {
  for(int i=0;i<100;i++)  {
   System.out.print(id);
   try {sleep(100);}
   catch(InterruptedException e) {}
  }
 }

 public static void main(String args[])
 {
  for(int i=0;i<3;i++) {
    Hilos h = new Hilos(i);
    h.start();
  }
 }
}
```

Concurrent TCP Server

- Our Server class extends Thread class
- Our Server class constructor
- The code that implements the service provided by our server should be in the **run method** of an extended Thread class
- The main method receives the client requests, when a request is accepted, a "child" thread is spawned. This new thread will serve to the client.

```
import java.net.*;
import java.io.*;

class SCTCP extends Thread {
 Socket id;
public SCTCP(Socket s) {id=s;}
public void run() {
 try {
  PrintWriter salida=new PrintWriter(id.getOutputStream(),true);
  while(true){
  salida.println(System.currentTimeMillis());
  sleep(100);
  }
 } catch(Exception e) {}
}

public static void main(String args[]) throws IOException{
 ServerSocket ss=new ServerSocket(8888);
 while(true) {
    Socket s = ss.accept();
    SCTCP t = new SCTCP(s);
    t.start();
   }
 }
}
```

We need to wrap all the code in the run() method inside a try/catch block