Software Engineering

Sockets

Prof. Agostino Poggi

What Are Sockets?

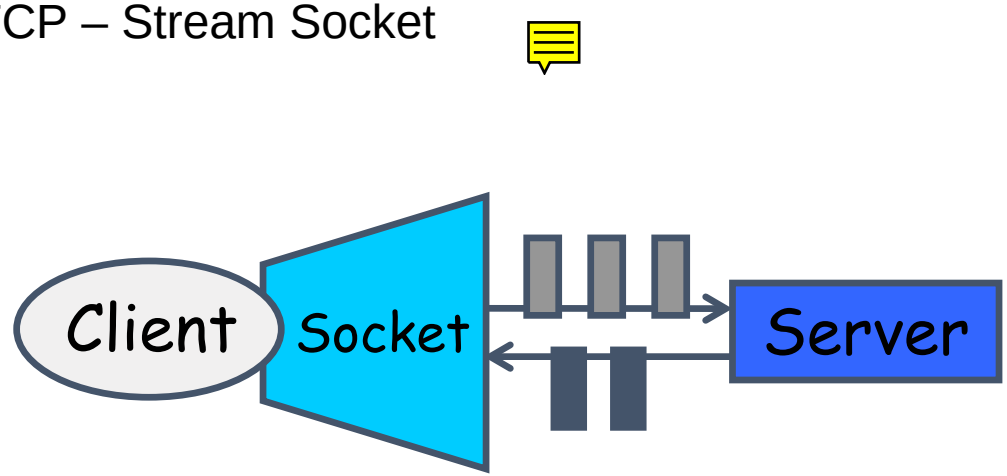
- Provide an interface for programming **networks** at the **transport layer**

- Their use is **often** very **similar** to performing **file I/O**

- Their use is programming **language independent**

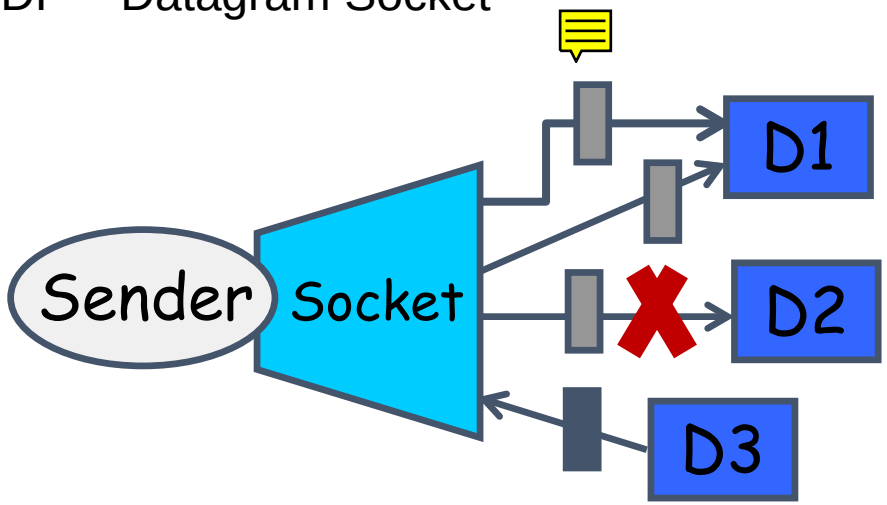
How Work Sockets?

TCP – Stream Socket



- Reliable delivery
- In-order guaranteed
- Connection-oriented
- Bidirectional

UDP – Datagram Socket



- Unreliable delivery
- No order guarantees
- No notion of “connection”
- Mono-directional

Server Program Normal Behavior With TCP

- Runs on a **specific computer**
- Has a **socket** that is **bound** to a **specific port**
- **Listens** to the socket **waiting for** a **connection request** from a client
- **Accepts** the **connection**
- **Gets** a **new socket bound** to a **different port** and associates it with the **connection**

TCP Server

- Create the server socket
- Wait for the client request
- Create I/O communication streams
- Perform communication with client
- Close sockets

DataServer Class

```java
public class DataServer
{
  private static final int SPORT = 4444;

  public void reply()
  {
    try
    {
      ServerSocket server = new ServerSocket(SPORT);

      Socket client = server.accept();

      BufferedReader   is = new BufferedReader(
          new InputStreamReader(client.getInputStream()));
      DataOutputStream os = new DataOutputStream(client.getOutputStream());

      System.out.println("Server received: " + is.readLine());

      os.writeBytes("Done\n");

      client.close();
      server.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new DataServer().reply();
  }
}
```

TCP Client

- Create a socket
- Create communication I/O streams
- Perform communication with server
- Close the socket when done

DataClient Class

```java
public class DataClient
{
  private static final int SPORT = 4444;
  private static final String SHOST = "localhost";

  public void send()
  {
    try
    {
      Socket client = new Socket(SHOST, SPORT);

      BufferedReader   is = new BufferedReader(
          new InputStreamReader(client.getInputStream()));
      DataOutputStream os = new DataOutputStream(client.getOutputStream());

      os.writeBytes("Hello\n");

      System.out.println("Client received: " + is.readLine());

      client.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new DataClient().send();
  }
}
```

127.0.0.1

Socket Programming With UDP

- **No** need for a **welcoming socket**
- **No streams** are attached to the sockets
- Sending processes create **packets**
  - IP destination address (224.0.0.0 - 239.255.255.255)
  - Port number
  - Content bytes
- Receiving processes extract **content bytes**

UDP Receiver

- Create a multicast socket and join the group
- Build a datagram packet to be received
- Receive the datagram packet
- Close the socket when done

DataReceiver Class

```java
public class DataReceiver
{
  private static final String ADDRESS = "230.0.0.1";
  private static final int DPORT = 4446;
  private static final int SIZE = 256;

  public void receive()
  {
    try
    {
      MulticastSocket socket = new MulticastSocket(DPORT);

      socket.joinGroup(InetAddress.getByName(ADDRESS));

      byte[] buf = new byte[SIZE];

      DatagramPacket packet = new DatagramPacket(buf, buf.length);

      socket.receive(packet);

      System.out.println("Receiver received: " + new String(packet.getData()));

      socket.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new DataReceiver().receive();
  }
}
```

UDP Sender

- Create a datagram socket and group address
- Build a datagram packet to be sent
- Send the datagram packet
- Close the socket when done

DataSender Class

```java
public class DataSender
{
  private static final int SPORT = 4444;
  private static final String ADDRESS = "230.0.0.1";
  private static final int DPORT = 4446;

  public void send()
  {
    try
    {
      DatagramSocket socket = new DatagramSocket(SPORT);
      InetAddress    group  = InetAddress.getByName(ADDRESS);

      String s = "Hello\n";
      byte[] b = s.getBytes();

      DatagramPacket packet = new DatagramPacket(b, b.length, group, DPORT);

      socket.send(packet);

      socket.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new DataSender().send();
  }
}
```

Message Class

```java
public final class Message implements Serializable
{
  private static final long serialVersionUID = 1L;

  private User user;
  private String content;

  public Message(final User u, final String c)
  {
    this.user    = u;
    this.content = c;
  }

  public User getUser()
  {
    return this.user;
  }

  public String getContent()
  {
    return this.content;
  }

  public void setContent(final String c)
  {
    this.content = c;
  }
}
```

User Class

```java
public final class User implements Serializable
{
  private static final long serialVersionUID = 1L;

  private final String firstName;
  private final String lastName;
  private final String address;
  private final transient String password;

  public User(final String f, final String l, final String a, final String p)
  {
    this.firstName = f;
    this.lastName  = l;
    this.address   = a;
    this.password  = p;
  }

  public String getFirstName()
  {
    return this.firstName;
  }

  public String getLastName()
  {
    return this.lastName;
  }

  public String getAddress()
  {
    return this.address;
  }

  public String getPassword()
  {
    return this.password;
  }
}
```

Write an Object

- Create a stream where write the object
- Create an object output stream on the previous stream
- Write the object into the object output stream
- Close the streams

Read an Object

- Open the stream from which read the object
- Create an object input stream on the previous stream
- Read the object from the object input stream
- Close file streams

ObjectServer Class (1/2)

```java
public class ObjectServer
{
  private static final int SPORT  = 4444;
  private static final double MIN = 0.1;

  public void reply()
  {
    try
    {
      ServerSocket server = new ServerSocket(SPORT);

      Socket client = server.accept();

      ObjectInputStream  is = new ObjectInputStream(client.getInputStream());
      ObjectOutputStream os = new ObjectOutputStream(client.getOutputStream());
      Random             r  = new Random();

      while (true)
      {

      }

      client.close();
      server.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new ObjectServer().reply();
  }
}
```

ObjectServer Class (2/2)

```java
public class ObjectServer
{
  private static final int SPORT  = 4444;
  private static final double MIN = 0.1;

  public void reply()
  {
    try
    {
      ServerSocket server = new ServerSocket(SPORT);

      Socket client = server.accept();

      ObjectInputStream  is = new ObjectInp
      ObjectOutputStream os = new ObjectOut
      Random             r  = new Random();

      while (true)
      {
        // Reads until there are messages or
        // until it sends an "end" message

        Object o = is.readObject();

        if ((o != null) && (o instanceof Message))
        {
          Message m = (Message) o;

          System.out.format("Server received: %s from Client\n",
              m.getContent());

          if (r.nextDouble() > MIN)
          {
            os.writeObject(new Message(m.getUser(), "done"));
            os.flush();
          }
          else
          {
            os.writeObject(new Message(m.getUser(), "end"));
            os.flush();
            break;
          }
        }
        else
        {
          break;
        }
      }

      client.close();
      server.close();
    }
    catch (IOException | ClassNotFoundExcep
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new ObjectServer().reply();
  }
}
```

ObjectClient Class (1/2)

```java
public class ObjectClient
{
  private static final int SPORT = 4444;
  private static final String SHOST = "localhost";
  private String[] greetings = {"hello", "hi", "ciao", "hey", "aloha",
      "shalom"};

  public void send()
  {
    try
    {
      Socket  client = new Socket(SHOST, SPORT);
      Random  r      = new Random();
      Message m      = new Message(
          new User("Agostino", "Poggi", "agostino.poggi@unipr.it", "agostino"),
          "");

      ObjectOutputStream os = new ObjectOutputStream(client.getOutputStream());
      ObjectInputStream  is = new ObjectInputStream(client.getInputStream());

      while (true)
      {

      }

      client.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] v)
  {
    new ObjectClient().send();
  }
}
```

ObjectClient Class (2/2)

```java
public class ObjectClient
{
  private static final int SPORT = 4444;
  private static final String SHOST = "localhost";
  private String[] greetings = {"hello", "hi", "ciao", "hey", "aloha",
      "shalom"};

  public void send()
  {
    try
    {
      Socket  client = new Socket(SHOST, SPORT);
      Random  r      = new Random();
      Message m      = new Message(
          new User("Agostino", "Poggi", "agostino.poggi@unipr.it", "agostino"),
          "");

      ObjectOutputStream os = new ObjectO
      ObjectInputStream  is = new ObjectI

      while (true)
      {

      }

      client.close();
    }
    catch (IOException | ClassNotFoundExc
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[]
  {
    new ObjectClient().send();
  }
}
```

```java
// Sends messages until it receives an end message

m.setContent(greetings[r.nextInt(greetings.length)]);

System.out.format("Client sends: %s to Server", m.getContent());

os.writeObject(m);
os.flush();

Object o = is.readObject();

if ((o != null) && (o instanceof Message))
{
  Message n = (Message) o;

  System.out.format("\tClient received: %s from Server\n",
      n.getContent());

  if (n.getContent().equals("end"))
  {
    break;
  }
}
```

ObjectServer and ObjectClient Output

```
Client sends: hello to Server and received: done from Server
Client sends: shalom to Server and received: done from Server
Client sends: aloha to Server and received: done from Server
Client sends: hi to Server and received: done from Server
Client sends: hi to Server and received: done from Server
Client sends: shalom to Server and received: end from Server
```

```
Server received: hello from Client
Server received: hello from Client
Server received: hello from Client
Server received: hello from Client
Server received: hello from Client
Server received: hello from Client
```

Clients sends a new content object, but in the same message object

Serialization stores an object and then references to its storage when the same object is stored multiple times (no circular reference issues)

```
m.setContent(greetings[r.nextInt(greetings.length)]);
```

ObjectReceiver Class (1/2)

```java
public class ObjectReceiver
{
  private static final String ADDRESS = "230.0.0.1";
  private static final int DPORT = 4446;
  private static final int SIZE = 1024;

  public void receive()
  {
    try
    {
      MulticastSocket socket = new MulticastSocket(DPORT);
      socket.joinGroup(InetAddress.getByName(ADDRESS));

      byte[]          buf    = new byte[SIZE];
      DatagramPacket packet = new DatagramPacket(buf, buf.length);

      socket.receive(packet);

      Object o = toObject(packet.getData());

      if (o instanceof Message)
      {
        Message m = (Message) o;

        System.out.format("Receiver received: %s from user with password: %s\n",
            m.getContent(), m.getUser().getPassword());
      }

      socket.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
      e.printStackTrace();
    }
  }

}
```

ObjectReceiver Class (2/2)

```java
public class ObjectReceiver
{
  private static final String ADDRESS = "230.0.0.1";
  private static final int DPORT = 4446;
  private static final int SIZE = 1024;

  public void receive()
  {
    try
    {
      MulticastSocket socket = new MulticastSocket(DPORT);
      socket.joinGroup(InetAddress.getByName(ADDRESS));

      byte[]          buf    = new byte[SIZE];
      DatagramPacket packet = new DatagramPacket(buf, buf.length);

      socket.receive(packet);

      Object o = toObject(packet.getData());

      if (o instanceof Message)
      {
        Message m = (Message) o;

        System.out.format("Receiver receive
            m.getContent(), m.getUser().get
      }

      socket.close();
    }
    catch (IOException | ClassNotFoundExcep
    {
      e.printStackTrace();
    }
  }
}
```

```java
private Object toObject(final byte[] b)
        throws IOException, ClassNotFoundException
{
  ObjectInputStream s =new ObjectInputStream(
      new ByteArrayInputStream(b));

  Object o = s.readObject();
  s.close();

  return o;
}

public static void main(final String[] v)
{
  new ObjectReceiver().receive();
}
```

ObjectSender Class (1/2)

```java
public class ObjectSender
{
  private static final int SPORT = 4444;
  private static final String ADDRESS = "230.0.0.1";
  private static final int DPORT = 4446;

  public void send()
  {
    try
    {
      DatagramSocket socket = new DatagramSocket(SPORT);

      InetAddress group = InetAddress.getByName(ADDRESS);

      Message m = new Message(new User("Agostino", "Poggi",
          "agostino.poggi@unipr.it", "agostino"), "hello");

      System.out.format("Sender sends %s for user with password: %s\n",
          m.getContent(), m.getUser().getPassword());

      byte[] buf = toByteArray(m);

      DatagramPacket packet = new DatagramPacket(
          buf, buf.length, group, DPORT);

      socket.send(packet);
      socket.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }

}
```

ObjectSender Class (2/2)

```java
public class ObjectSender
{
  private static final int SPORT = 4444;
  private static final String ADDRESS = "230.0.0.1";
  private static final int DPORT = 4446;

  public void send()
  {
    try
    {
      DatagramSocket socket = new DatagramSocket(SPORT);

      InetAddress group = InetAddress.getByName(ADDRESS);

      Message m = new Message(new User("Agostino", "Poggi",
          "agostino.poggi@unipr.it", "agostino"), "hello");

      System.out.format("Sender sends %s for user with password: %s\n",
          m.getContent(), m.getUser().getPassword());

      byte[] buf = toByteArray(m);

      DatagramPacket packet = new Dat
          buf, buf.length, group, DPO

      socket.send(packet);
      socket.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }
}
```

```java
private byte[] toByteArray(final Object o) throws IOException
{
  ByteArrayOutputStream b = new ByteArrayOutputStream();
  ObjectOutputStream    s = new ObjectOutputStream(b);

  s.writeObject(o);
  s.flush();
  s.close();
  b.close();

  return b.toByteArray();
}

public static void main(final String[] v)
{
  new ObjectSender().send();
}
```

ObjectSender and ObjectReceiver Output

```
Sender sends hello for user with password: agostino

Receiver received: hello from user with password: null
```

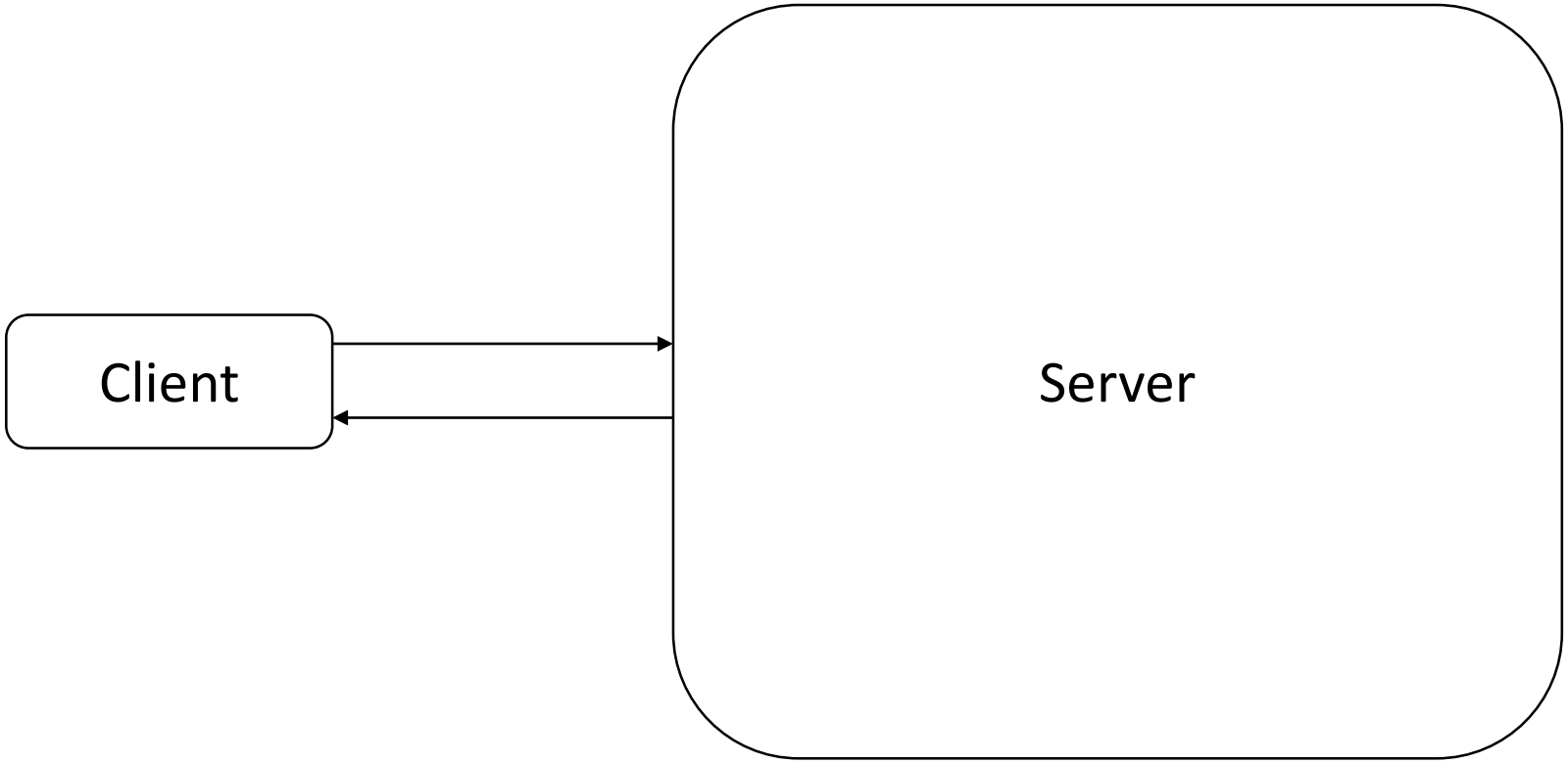
User class password field is defined as transient

```
private final transient String password;
```

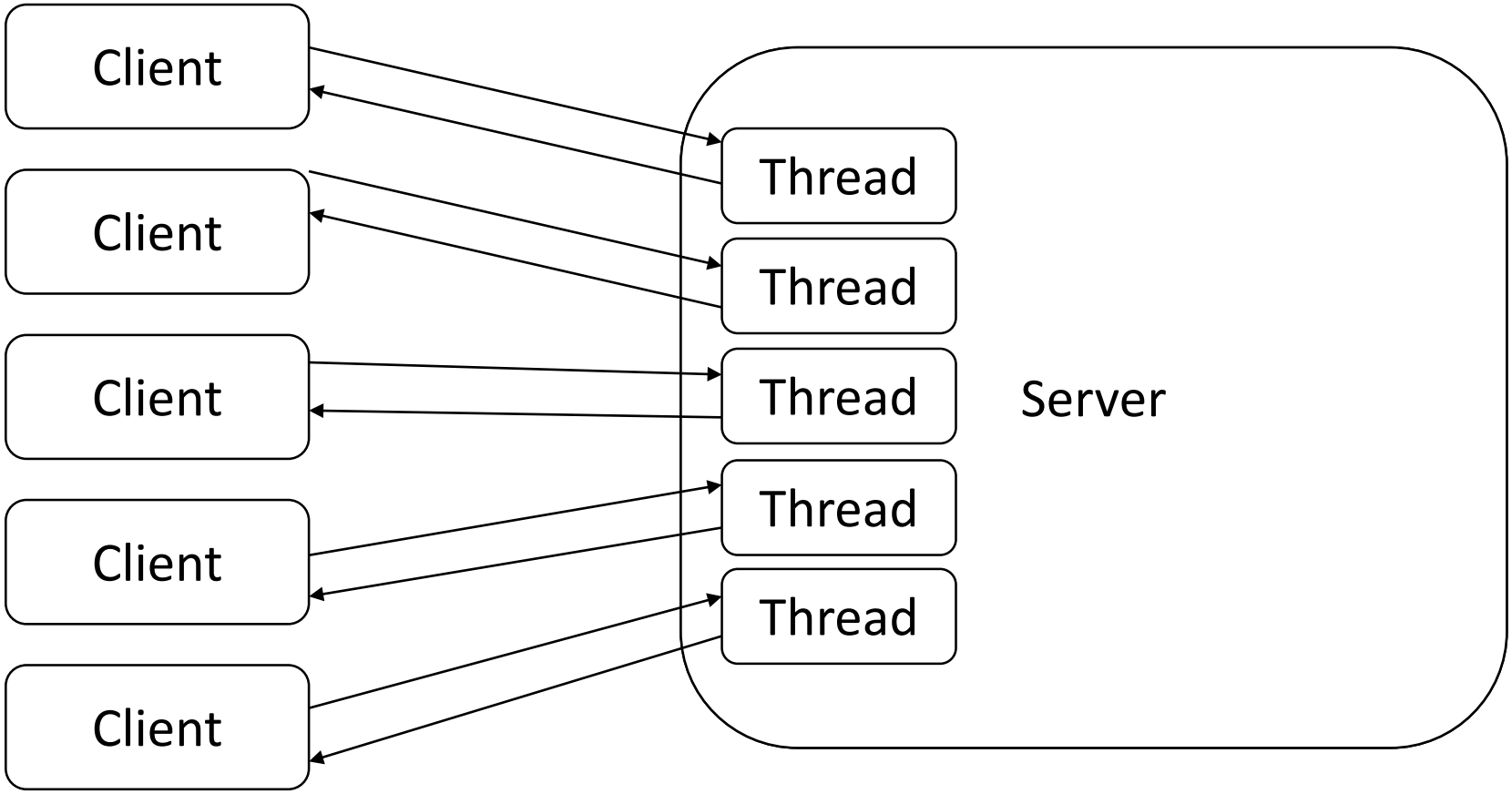
Managing Exceptions

- When **opening** a **stream** or a **socket: IOException**
- When **opening** a **client socket: UnknownHostException**
  - UnknownHostException extends IOException
- When there is a **security manager: SecurityException**
- When **reading** an **object: ClassNotFoundException**

Client – Server Socket Applications



Multi-Client – Server Socket Application



Request and Response Classes

```java
public class Request implements Serializable
{
  private static final long serialVersionUID = 1L;

  private final int value;

  public Request(final int v)
  {
    this.value = v;
  }

  public int getValue()
  {
    return this.value;
  }
}

public class Response implements Serializable
{
  private static final long serialVersionUID = 1L;

  private final int value;

  public Response(final int v)
  {
    this.value = v;
  }

  public int getValue()
  {
    return this.value;
  }
}
```

Client Class (1/2)

```java
public class Client
{
  private static final int SPORT = 4444;
  private static final String SHOST = "localhost";

  private static final int MAX = 100;

  public void run()
  {
    try
    {
      Socket  client = new Socket(SHOST, SPORT);

      ObjectOutputStream os = new ObjectOutputStream(client.getOutputStream());
      ObjectInputStream  is = null;

      Random r = new Random();

      while (true)
      {

      }

      client.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
      e.printStackTrace();
    }
  }

  public static void main(final String[] args)
  {
    new Client().run();
  }
}
```

Client Class (2/2)

```java
public class Client
{
  private static final int SPORT = 4444;
  private static final String SHOST = "localhost";

  private static final int MAX = 100;

  public void run()
  {
    try
    {
      Socket  client = n

      ObjectOutputStream
      ObjectInputStream

      Random r = new Ran

      while (true)
      {

      }

      client.close();
    }
    catch (IOException |
    {
      e.printStackTrace(
    }
  }

  public static void mai
  {
    new Client().run();
  }
}
```

```java
Request rq = new Request(r.nextInt(MAX));

System.out.format("Client sends: %s to Server", rq.getValue());

os.writeObject(rq);
os.flush();

if (is == null)
{
  is = new ObjectInputStream(new BufferedInputStream(
      client.getInputStream()));
}

Object o = is.readObject();

if (o instanceof Response)
{
  Response rs = (Response) o;

  System.out.format(" and received: %s from Server%n", rs.getValue());

  if (rs.getValue() == 0)
  {
    break;
  }
}
```

Server Class

```java
public class Server
{
  private static final int COREPOOL = 5;
  private static final int MAXPOOL = 100;
  private static final long IDLETIME = 5000;

  private static final int SPORT = 4444;

  private ServerSocket socket;
  private ThreadPoolExecutor pool;


  public Server() throws IOException
  {
    this.socket = new ServerSocket(SPORT);
  }

  private void run()
  {

  }

  public ThreadPoolExecutor getPool()
  {
    return this.pool;
  }

  public void close()
  {

  }

  public static void main(final String[] args) throws IOException
  {
    new Server().run();
  }
}
```

```java
this.pool = new ThreadPoolExecutor(COREPOOL, MAXPOOL, IDLETIME,
    TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());

while (true)
{
  try
  {
    Socket s = this.socket.accept();

    this.pool.execute(new ServerThread(this, s));
  }
  catch (Exception e)
  {
    break;
  }
}

this.pool.shutdown();
```

```java
try
{
  this.socket.close();
}
catch (Exception e)
{
  e.printStackTrace();
}
```

ServerThread Class (1/2)

```java
public class ServerThread implements Runnable
{
  private static final int MAX = 100;
  private static final long SLEEPTIME = 200;

  private Server server;
  private Socket socket;

  public ServerThread(final Server s, final Socket c)
  {
    this.server = s;
    this.socket = c;
  }

  @Override
  public void run()
  {
    ObjectInputStream  is = null;
    ObjectOutputStream os = null;

    try
    {
      is = new ObjectInputStream(new BufferedInputStream(
          this.socket.getInputStream()));
    }
    catch (Exception e)
    {
      e.printStackTrace();

      return;
    }

    String id = String.valueOf(this.hashCode());

    Random r = new Random();
  }
}
```

```java
while (true)
{
  try
  {

  }
  catch (Exception e)
  {
    e.printStackTrace();
    System.exit(0);
  }
}
```

ServerThread Class (2/2)

```java
public class ServerThread implements Runnable
{
  private static final int MAX = 1
  private static final long SLEEPT

  private Server server;
  private Socket socket;

  public ServerThread(final Server
  {
    this.server = s;
    this.socket = c;
  }

  @Override
  public void run()
  {
    ObjectInputStream  is = null;
    ObjectOutputStream os = null;

    try
    {
      is = new ObjectInputStream(n
           this.socket.getInputStre
    }
    catc
    {
      e.

      re
    }

    Stri

    Rand

  }
}
```

```java
while (true)
{
  try
  {

  }
  catch (Exception e)
  {
    e.printStackTrace()
    System.exit(0);
  }
}
```

```java
Object i = is.readObject();

if (i instanceof Request)
{
  Request rq = (Request) i;

  System.out.format("thread %s receives: %s from its client%n",
      id, rq.getValue());
  Thread.sleep(SLEEPTIME);

  if (os == null)
  {
    os = new ObjectOutputStream(new BufferedOutputStream(
        this.socket.getOutputStream()));
  }

  Response rs = new Response(r.nextInt(MAX));

  System.out.format("thread %s sends: %s to its client%n",
      id, rs.getValue());
  os.writeObject(rs);
  os.flush();

  if (rs.getValue() == 0)
  {
    if (this.server.getPool().getActiveCount() == 1)
    {
      this.server.close();
    }

    this.socket.close();
    return;
  }
}
```