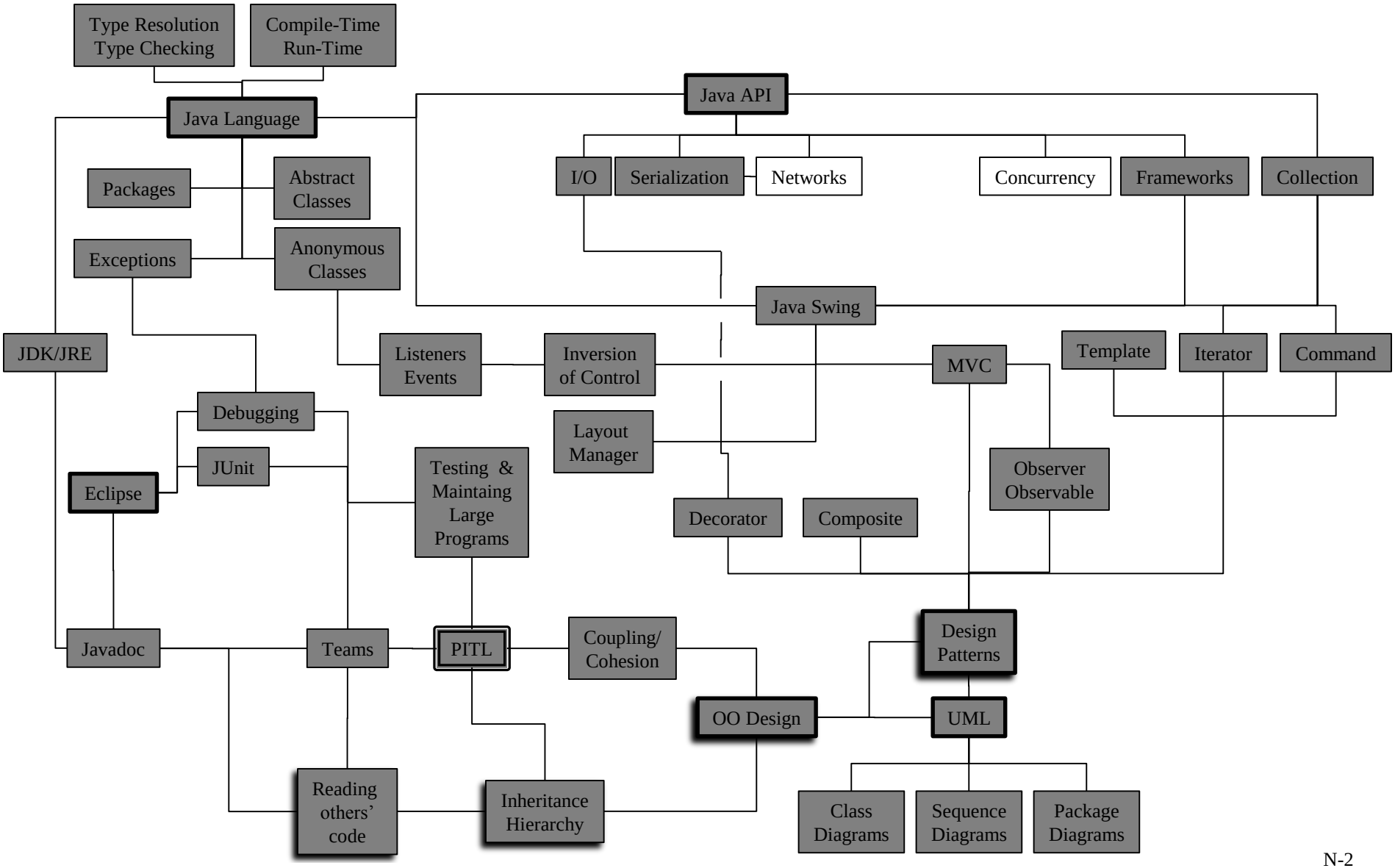
Networking with Java

CSc 335
Object-Oriented Programming and Design

Craig Barber, Ian Vasquez, Rick Snodgrass, Rick Mercer

Networking


Type Resolution
Type Checking
Compile-Time
Run-Time
Java Language
Java API
Packages
Abstract
Classes
I/O
Serialization
Networks
Concurrency
Frameworks
Collection
Exceptions
Anonymous
Classes
Java Swing
JDK/JRE
Listeners
Events
Inversion
of Control
MVC
Template
Iterator
Command
Debugging
Layout
Manager
JUnit
Testing &
Maintaing
Large
Programs
Observer
Observable
Eclipse
Decorator
Composite
Javadoc
Teams
PITL
Coupling/
Cohesion
Design
Patterns
OO Design
UML
Reading
others'
code
Inheritance
Hierarchy
Class
Diagrams
Sequence
Diagrams
Package
Diagrams

Outline

- Introduction to Networking Concepts
  - Client-Server and Peer-to-Peer
  - Sockets
  - Streams

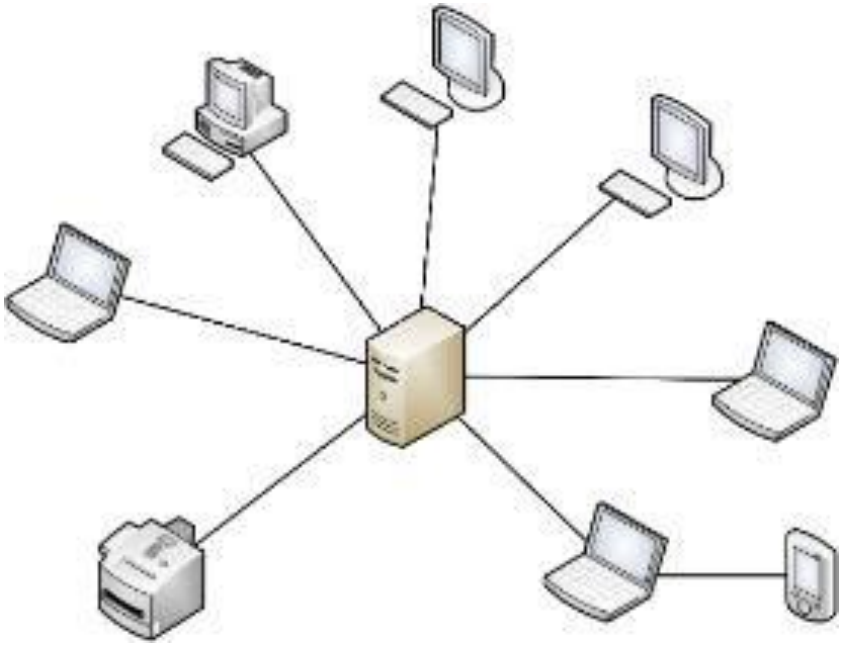
What is “Networking”

- What is “Networking”?
  - *G*etting two or more computers to send data (in Java--serialized objects) to each other
  - Having programs on separate computers interact with one another

- Types of Networking
  - *Client - Server*
    - Many clients connect with one server.
    - Clients communicate only with server.
  - *Peer-to-Peer*
    - Clients connect to a group of other clients, with no server.
    - Clients communicating directly with each-other.

Client - Server Networking

- Advantages:
  - Easier to implement
  - Less coordination involved
  - Easier to maintain control of users
- Disadvantage:
  - Relies on one main server for entire operation



How Can Networking Work?

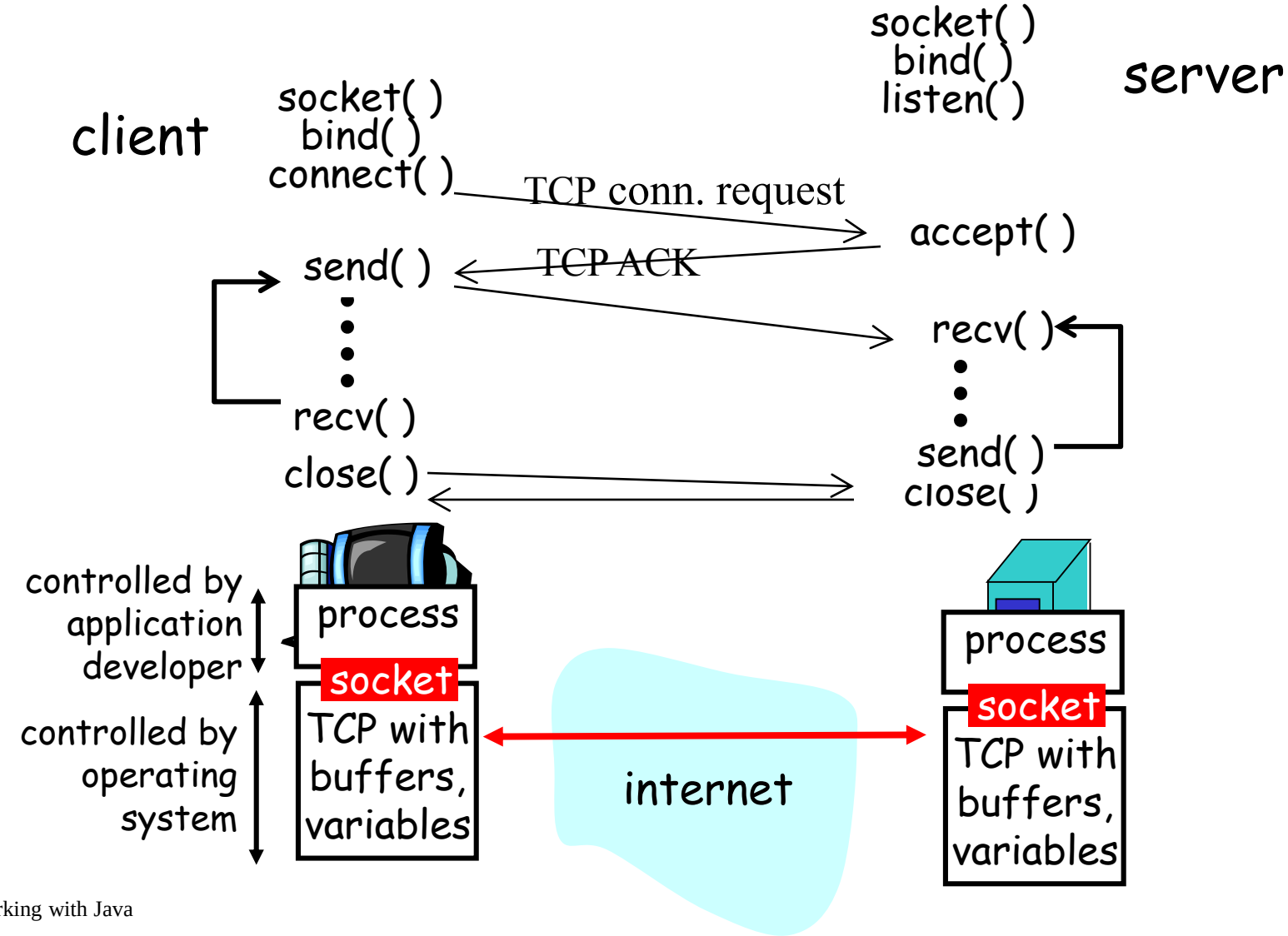
- Computers connect to each other through links called *sockets*, each associated with a single computer.
- A *network stream* is created by connecting a socket on one computer to a socket on another computer
- Applications communicate by sending data through streams to each other
  - Reading and writing objects over the network employs the same serialization you used for persistence

Sockets

- A socket is a connection on one computer used to send data back and forth
- The application consists of multiple processes, one running on each computer
- Sockets are created by the process on each computer
- The sockets then establish a connection to each other
  - One process sets up a server socket to receive a connection.
  - The other process sets up a client socket to establish the connection with the server socket.

Socket-programming using TCP

TCP service: reliable byte stream transfer



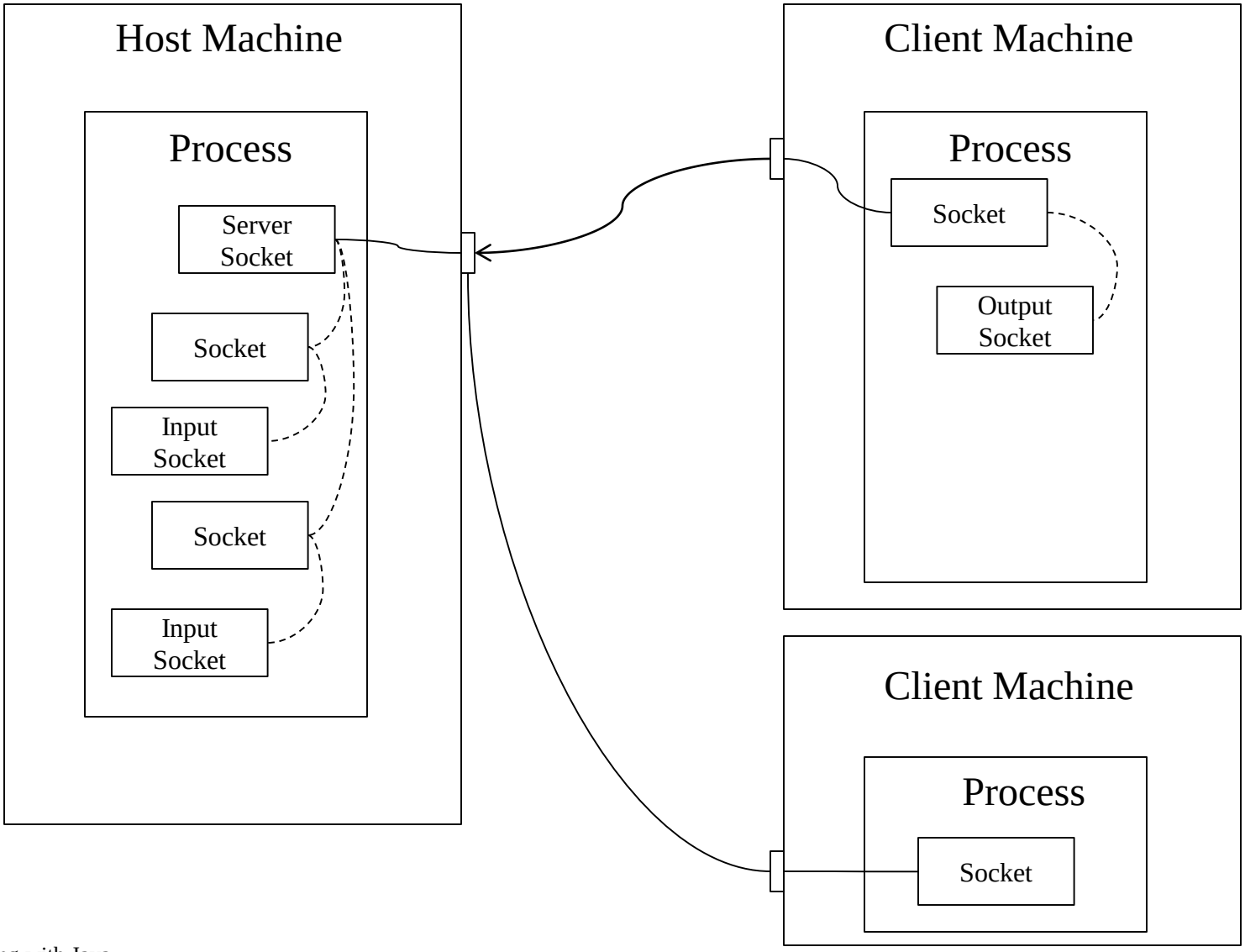
Outline

- Introduction to Networking Concepts
- Networking in Java
  - Sockets
  - Streams
  - Decorating Streams
- Summary

Sockets in Java

- Found in `java.net` package
- `java.net.ServerSocket`
  - Accepts new incoming connections
  - Creates new `ServerSocket` for each connection
- `java.net.Socket`
  - Connects to an existing `ServerSocket`, through the network

Sockets in Java



Two new types

- We'll be using two new types
  - `java.net.ServerSocket`
  - `java.net.Socket`
    - You can write to and read from a Socket's input and output streams with readObject and writeObject messages
      - which makes networked programs easier to develop

java.net.ServerSocket

- `public ServerSocket(int port)`
  - Throws `IOException`
  - Creates a `ServerSocket` to accept new connections at the specified port
- `public Socket accept()`
  - Throws `IOException`
  - Waits for an incoming connection, establishes the new connection, and returns a socket for that connection
  - Multiple applications can connect to the same ServerSocket
- `public void close()`
  - Throws `IOException`
  - Closes the server socket.
  - Does *not* close open sockets.

java.net.Socket

- `public Socket(String host, int port)`
  - Throws `IOException`, `UnknownHostException`
  - Connects to a server socket at the provided address (host) on the provided port
- `public InputStream getInputStream()`
  - Throws `IOException`
  - Returns the input stream from the socket
- `public OutputStream getOutputStream()`
  - Throws `IOException`
  - Returns the output stream from the socket
- `public void close()`
  - Throws `IOException`
  - Closes the connection

Building a Network app

- This app will have one server and only one client (no Threads needed)
- Build the server first
  - Need a new ServerSocket (int port)
  - The accept message to ServerSocket waits for a connection that knows where the server is and what port it is listening to

```
int port = 4000; // A port available on lectura soince 2003
ServerSocket socket = new ServerSocket(port);
Socket connection = socket.accept();
```

Get the connection's streams

- Let the server communicate with the connection

```
ObjectOutputStream output
    = new ObjectOutputStream(connection.getOutputStream());

ObjectInputStream input
       = new ObjectInputStream(connection.getInputStream());
```

Let the server read and write in loop

- Let the server communicate with the connection

```
// Take money from the client's account
BankAccount theClientsAccount = null;
BankAccount theServersAccount = new BankAccount("Greedy", 0.00);
while (true) {
   double amount = ((Double) input.readObject()).doubleValue();
   if (amount <= 0.0)
      break;
   theClientsAccount = (BankAccount) input.readObject();
   if (theClientsAccount.withdraw(amount))
      theServersAccount.deposit(amount);
   // Send back the modified object
  output.writeObject(theClientsAccount);
}
connection.close();
```

Write the Client

- Get the input and output stream to and friom the server we are connecting to

```
ObjectOutputStream output
    = new ObjectOutputStream(server.getOutputStream());

ObjectInputStream input
    = new ObjectInputStream(server.getInputStream());
```

Write the Client

- Request a socket connection with the running sever

```
// This IPAddress is for the machine where this code will run
// We'll run the server and the client on the same machine for now
String IPAddress = "localhost"; // There's no place like home
int port = 4000;
Socket server = new Socket(IPAddress, port);
```

Let the client read and write in loop

- Let the client communicate with the server

```
// Give money to the server without knowing it
BankAccount myAccount = new BankAccount("Sucker", 5000.00);
boolean done = false;
while (!done) {
  String amountAsString = JOptionPane.showInputDialog(
         null,
         "You've won! Enter desired amount" + " you have "
            + myAccount.getBalance());
  double amount = Double.parseDouble(amountAsString);
  output.writeObject(new Double(amount));
  if (amount > 0) {
    output.writeObject(myAccount);
    myAccount = (BankAccount) input.readObject();
  } else
    // The user figured out how to quit
    done = true;
}
server.close();
```