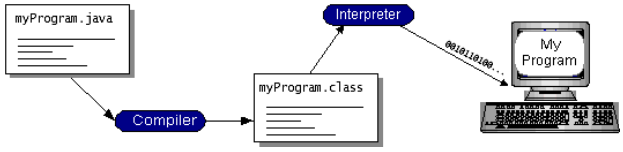
Introduction to Java (tentative)

Jinzhong Niu
Sunday, February 08, 2004

How to Program in Java

- Text editor to edit Java source code files
  - Notepad, UltraEdit, etc.
- Sun's in JDK (Java Development Kit)
  - Java compiler - *javac*
  - JVM (Java Virtual Machine) - *java*

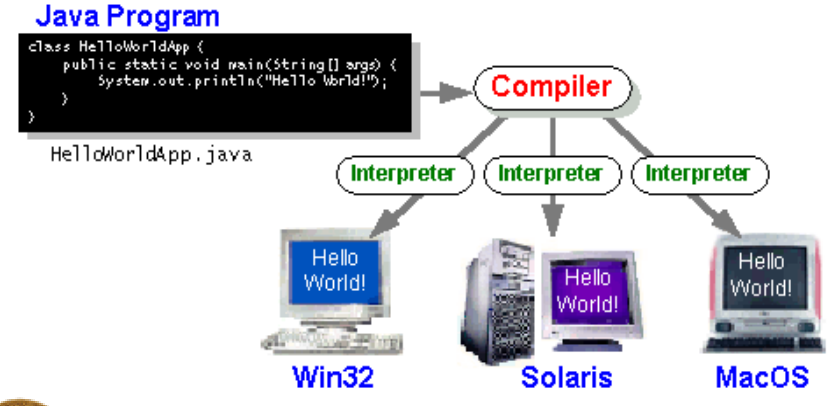


Example: Hello Java World!

- *HelloWorldApp.java*

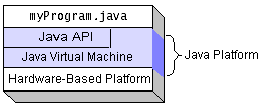
```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello Java World!");
    }
}
```

Write Once, Run Anywhere



The Java Platform

- Platform
  - Hardware or software environment in which a program runs
  - Combination of the operating system and hardware
- The Java platform
  - Software-only platform that runs on top of other hardware-based platforms
  - Slow Java execution on such a platform



Object Orientation (OO)

- Java Application
  - a set of related classes
  - main class with *public static void main(String[] args)* method defined
- Class
  - Constructors, Methods, variables (*public*, *private*, …), …
  - Descendant of *Object*
  - Coded in a separate file with the same name

OO – Using Objects

- Here, code in one class creates an instance of another class and does something with it …

```
Fruit plum=new Fruit();
int cals;
cals = plum.total_calories();
```

- ***Dot operator*** allows you to access (public) data/methods inside *Fruit* class

OO – *Public* and *Private*

- Methods/variables may be declared ***public*** or ***private*** meaning they may or may not be accessed by code in other classes …
- Good practice:
  - keep data private
  - keep most methods private
- Well-defined interface between classes - helps to eliminate errors

OO – Creating objects

- Following code creates an instance of the *Fruit* class

  Fruit plum = new Fruit();

- The initial data of an object may be passed as parameters.
- Several different type of constructor with different argument lists, e.g. *Fruit()*, *Fruit(a)* ...

Usage of JDK Tools

- Compilation, generating *HelloWorldApp.class*

  javac HelloWorldApp.java

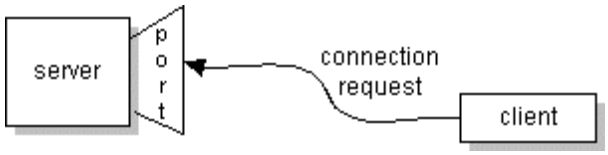
  Execution

  java HelloWorldApp

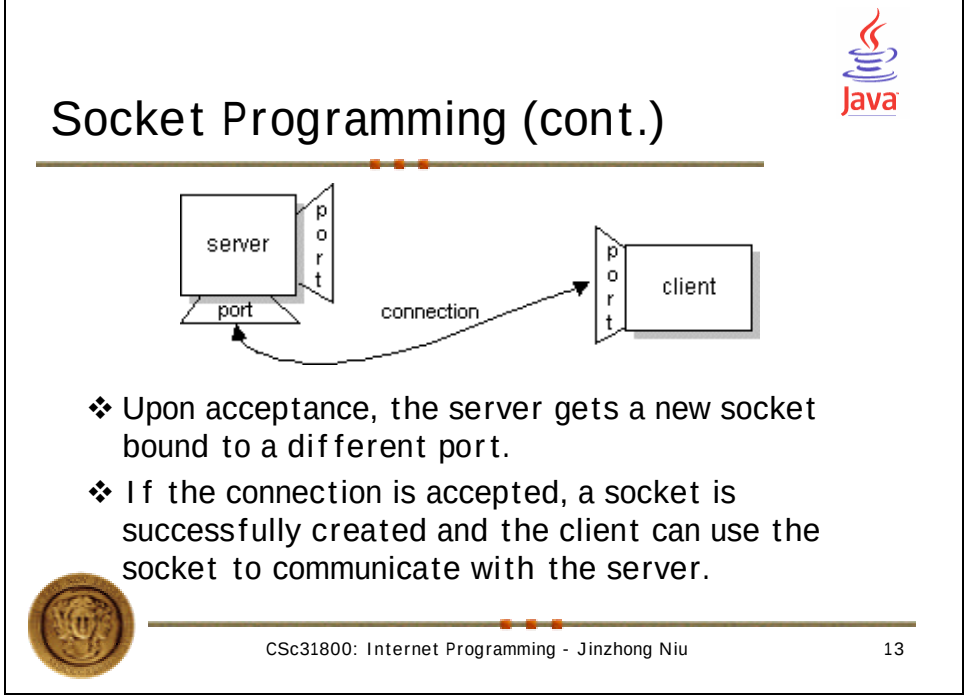
Socket Programming

- What is Socket?
  - A socket is a communication endpoint — an object through which an application sends or receives packets of data across a network.
- The purpose is to abstract away the underlining network.

Socket Programming (cont.)

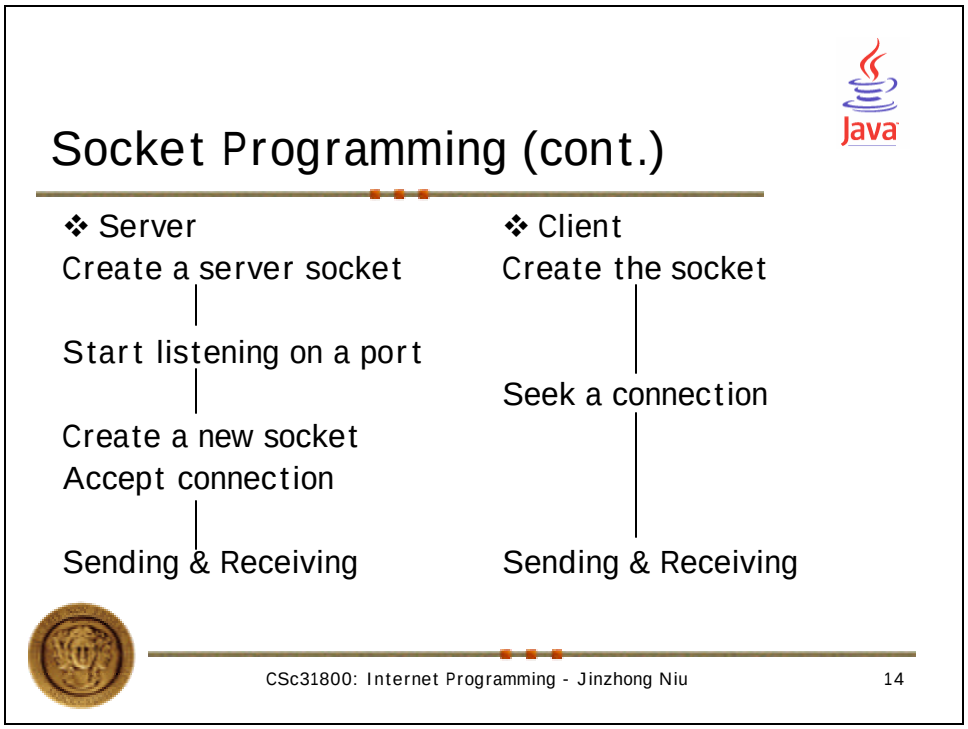


- A server runs on a specific computer and has a socket that is bound to a specific port number.
- A client makes a connection request based on the host name of the server and the port number.



Socket Programming (cont.)

- Upon acceptance, the server gets a new socket bound to a different port.
- If the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.



Socket Programming (cont.)

- Server
  - Create a server socket
  - Start listening on a port
  - Create a new socket
  - Accept connection
  - Sending & Receiving
- Client
  - Create the socket
  - Seek a connection
  - Sending & Receiving

Socket Programming (cont.)

- Client side
  - Socket
    - This class implements client sockets
    - A socket is an endpoint for communication between two machines
- Server side
  - ServerSocket
    - This class implements server sockets.
    - A server socket waits for requests to come in over the network.
    - It performs some operation based on that request, and then possibly returns a result to the requester.

Client Side – Socket

- Common methods
  - *getInetAddress()* - Returns the address to which the socket is connected
  - *getInputStream()* - Returns an input stream for this socket
  - *getOutputStream()* - Returns an output stream for this socket.
  - *getPort()* - Returns the remote port to which this socket is connected
  - *close()* – close this socket

Client Connection

- PrintWriter
  - Print formatted representations of objects to a text-output stream.
  - It does not contain methods for writing raw bytes, for which a program should use unicoded byte streams.
- BufferReader
  - Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines

Client Side – Operation

- Client flows
  1. Open a socket.
  2. Open an input stream and output stream to the socket.
  3. Read from and write to the stream according to the server's protocol.
  4. Close the streams.
  5. Close the socket.

Server Side

- Server flows
  1. create a *ServerSocket*

     No need to bind and listen, the constructor has already done that.
  2. accept a client call
  3. Send/Receive data
  4. close socket