CS 580 Client-Server Programming
Spring Semester, 2005
Doc 1 Introduction

Contents

Course Introduction ..................................................... 3
Introduction to Client-Server ........................................... 7
  What is Client-Server? ............................................... 7
  What Client-Server Requires of a Programmer ................. 9
Programming Issues ..................................................... 10
  Naming Convention for Classes, Variables & Methods ....... 10
  Names ......................................................................... 11
  Comments ..................................................................... 12
  Commenting Data Declarations .................................... 17

Copyright ©, All rights reserved. 2004 SDSU & Roger Whitney,
5500 Campanile Drive, San Diego, CA 92182-7700 USA.
OpenContent (http://www.opencontent.org/opl.shtml) license
defines the copyright on this document
References

*Code Complete* by Steve McConnell
Course Introduction
Course Outline

Introduction
Testing
Source Version Control
Network Basics
GUI
Client Development Issues
Concurrency
Server Types & Structure
Client-Server Protocols
Logging
Databases
Security
Web based Applications
  CGI, Fast-CGI, Servlets
Web Services
Advanced topics

This outline will be changed during the semester.
Main Points of Class

Common design issues & solutions for building a server

Issues in designing a client-server network protocol

Handling Concurrency

Accessing databases

Programming issues dealing with working on client-server code
Programming languages for the Course

Java

We will be using parts of JDK 1.5

Smalltalk – VisualWorks 7.3

C#

Not supported
What does it mean to “Know” Java, C# or Smalltalk

Basic syntax of the language

Core API

No one knows the entire API of either language

You should have good grasp of the common or core API

Collections, Files, Exceptions, Streams

Language culture - Ways of doing things in each language

- Java Doc
- Searching the API
- Compiling/running code
- Using Smalltalk browsers
- Naming conventions

Object-oriented programming
Introduction to Client-Server
What is Client-Server?

Client
Application that initiates peer-to-peer communication
Translate user requests into requests for data from server via protocol
GUI often used to interact with user

Server
Any program that waits for incoming communication requests from a client
Extracts requested information from data and return to client

Common Issues
- Authentication
- Authorization
- Data Security
- Privacy
- Protection
- Concurrency
What this Course is not

An advanced (or beginning) Networking course

### OSI Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Process Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
</tr>
<tr>
<td>6</td>
<td>Presentation</td>
</tr>
<tr>
<td>5</td>
<td>Session</td>
</tr>
<tr>
<td>4</td>
<td>Transport</td>
</tr>
<tr>
<td>3</td>
<td>Network</td>
</tr>
<tr>
<td>2</td>
<td>Data Link</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
</tr>
</tbody>
</table>

How to use a client builder application/system

Powerbuilder

What this Course covers

Skills & knowledge required to build client-server applications
What Client-Server Requires of a Programmer

- Designing robust protocols
- Network programming
- Designing usable computer-human interfaces
- Good documentation skills
- Good debugging skills
- Understand the information flow of the company/customer
- Mastery of concurrency
- Multi-platform development
- Database programming
- Security
Programming Issues
Naming Convention for Classes, Variables & Methods

• Use full words – avoid abrvtns

Pascal Casing
Capitalize the first character of each word

SomeClassName

Camel Casing
Capitalize the first character of each word except the first word

someVariableName

<table>
<thead>
<tr>
<th>Item</th>
<th>Java</th>
<th>Smalltalk</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>PascalCase</td>
<td>PascalCase</td>
<td>PascalCase</td>
</tr>
<tr>
<td>Method</td>
<td>camelCase</td>
<td>camelCase</td>
<td>PascalCase</td>
</tr>
<tr>
<td>Field</td>
<td>camelCase</td>
<td>camelCase</td>
<td>CamelCase</td>
</tr>
<tr>
<td>Parameter</td>
<td>camelCase</td>
<td>camelCase</td>
<td>camelCase</td>
</tr>
<tr>
<td>Local Variable</td>
<td>camelCase</td>
<td>camelCase</td>
<td>camelCase</td>
</tr>
</tbody>
</table>
Names

"Finding good names is the hardest part of OO Programming"

"Names should fully and accurately describe the entity the variable represents"

What role does the variable play in the program?

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Role, function</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputRec</td>
<td>EmployeeData</td>
</tr>
<tr>
<td>BitFlag</td>
<td>PrinterReady</td>
</tr>
</tbody>
</table>

**Some Examples of Names, Good and Bad**

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Role, function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrainVelocity</td>
<td>Velt, V, X, Train</td>
</tr>
<tr>
<td>CurrentDate</td>
<td>CD, Current, C, X, Date</td>
</tr>
<tr>
<td>LinesPerPage</td>
<td>LPP, Lines, L, X</td>
</tr>
</tbody>
</table>

Names should be as short as possible and still convey meaning to the reader
Comments

"Comments are easier to write poorly than well, and comments can be more damaging than helpful"

What does this do?

```pascal
for i := 1 to Num do
    MeetsCriteria[ i ] := True;
for i := 1 to Num / 2 do begin
    j := i + i;
    while ( j <= Num ) do begin
        MeetsCriteria[ j ] := False;
        j := j + i;
    end;
for i := 1 to Mun do
    if MeetsCriteria[ i ] then
        writeln( i, ' meets criteria' );
```
How many comments does this need?

```pascal
for PrimeCandidate:= 1 to Num do
  IsPrime[ PrimeCandidate ] := True;

for Factor:= 1 to Num / 2 do begin
  FactorableNumber := Factor + Factor;
  while ( FactorableNumber <= Num ) do begin
    IsPrime[ FactorableNumber ] := False;
    FactorableNumber := FactorableNumber + Factor;
  end;
end;

for PrimeCandidate:= 1 to Num do
  if IsPrime[ PrimeCandidate ] then
    writeln( PrimeCandidate, ' is Prime ' );
```

**Good Programming Style is the Foundation of Well Commented Program**
Commenting Paragraphs of Code

Write comments at the level of the code's intent

Comment the why rather than the how

Make every comment count

Document surprises

Avoid abbreviations

**How verses Why**

**How**

/* if allocation flag is zero */

if ( AllocFlag == 0 ) ... 

**Why**

/* if allocating a new member */

if ( AllocFlag == 0 ) ... 

**Even Better**

if ( AllocFlag == NEW_MEMBER ) ...
Summary comment on How

{ check each character in "InputStr" until a dollar sign is found or all characters have been checked }

Done := false;
MaxPos := Length( InputStr );
i := 1;
while ( (not Done) and (i <= MaxLen) ) begin
  if ( InputStr[ i ] = '$' ) then
    Done := True
  else
    i := i + 1
end;

Summary comment on Intent

{ find the command-word terminator }

Done := false;
MaxPos := Length( InputStr );
i := 1;

while ( (not Done) and (i <= MaxPos ) ) begin
  if ( InputStr[ i ] = '$' ) then
    Done := True
  else
    i := i + 1
end;
Summary comment on Intent with Better Style

{ find the command-word terminator }

FoundTheEnd := false;
MaxCommandLength := Length( InputStr );
Index := 1;

while ((not FoundTheEnd) and
       (Index <= MaxCommandLength)) begin

  if ( InputStr[ Index ] = '$' ) then
    FoundTheEnd := True;
  else
    Index := Index + 1;
end;
Commenting Data Declarations

Comment the units of numeric data

Comment the range of allowable numeric values

Comment coded meanings

```haskell
var
    CursorX: 1..MaxCols;  { horizontal screen position of cursor }
    CursorY: 1..MaxRows;  { vertical position of cursor on screen }
    AntennaLength: Real;   { length of antenna in meters: >= 2 }
    SignalStrength: Integer; { strength of signal in kilowatts: >= 1 }
    CharCode: 0..255;     { ASCII character code }
    CharAttrib: Integer;  { 0=Plain; 1=Italic; 2=Bold }
    CharSize: 4..127;     { size of character in points }
```

Comment limitations on input data

Document flags to the bit level