CS 3500 Final Notes Cheat Sheet

by _allisonwalker via cheatography.com/181463/cs/38486/

Networking

Key Networking Issues -Data Arrival

> You never know how much data will arrive at any given time Some or all may “Show up” - What does arrive will be in order Therefore: all data concatenated/stored Search “all data” for messages Protocol of “what is a message?” Newline/Period/etc.

Key Parts to Networking Object

> Callbacks -This is how the network code communicates state to the “user” program TcpClient/TcpListener objects -The main object you will be using See SimpleChat[Client/Server]TcpClient code ID -A unique identifier for the connection Settable (default → socket)

DataBase Principles - SQL and C# Interface to DB

Good Software Practices

GSPs refer to tools, techniques, processes, etc., that are used to ensure our code is:

> Easier to maintain Contains fewer defects

Good Software Practices (cont)

Examples include:

> Versioning (git/github/github projects) Design Patterns System Architecture decisions (e.g., MVC) SOLID, DRY Diagramming/UML Testing Unit Testing, Integration Testing, open/closed-box testing, C# test syntax, e.g., TestClass()]…)

Self Documenting Code

> “Self-documenting code is ostensibly written using human-readable names, typically consisting of a phrase in a human language which reflects the symbol’s meaning…”

Refactoring

> Changing code (often automatically) to be more maintainable

Refactoring Techniques 1

> Inline Temp That being said: Extract Variable (Self-Documenting Code) Decompose-conditional Split-temporary-variable Self-encapsulate-field

Refactoring Techniques 2

> Extract-interface Extract-method Consolidate-duplicate-conditional-fragments

Good Software Practices (cont)

Refactoring Techniques 3

> Hide-method Replace-exception-with-test Extract-class What SOLID is this?

Visual Studio (to the rescue!?!)

> Visual Studio support… None – Some operations will be done by hand Some – Some operations, like renaming can be done quickly. A lot – Some operations, like method extraction, can be done completely!

Visual Studio (to the rescue!?!) Continued

> Note: you cannot extract private methods to an interface… this doesn’t make sense as interfaces expose functionality. Note 2: Phase1 and Phase2 are probably poor examples of where you would create an interface.

Good Software Practices (cont)

Code Analysis

> Apply computer programs to Analyze and Improve computer programming Code Analysis Tools Complexity Analysis - e.g., how many lines of code Duplicate Code Analysis - e.g., could be refactored into methods Style Guides - e.g., put spaces around parameters Code Cleanup - e.g., remove unused using statements

Code Analysis Continued

> Code Analysis is run upon compilation Shown as warnings/messages in Error List

Code Cleanup

> Auto-Tools for “cleaning up” code base E.G., remove unnecessary usings Many of the “suggestions” you see with the underline squigglys Note: All “styles” should be agreed upon by team at start of development

Design Patterns

> Design Patterns Well understood and beneficial ways to effectively solve recurring problems Anti-Patterns Common ways of solving problems that have unintended side effects. Examples included: Iterator, and Null object

Good Software Practices (cont)

Design Pattern -Iterator

collection = new List(…) Foreach (var x in collection) collection = new Dictionary(…) Foreach (var x in collection) collection = new Tree(…) Foreach (var x in collection) They all work and collection Type can even be changed without changing the loop! We don’t care how the Data Structure iterates, just that it does Aside: How does a Dictionary iterate?

Design Pattern -Null Object

// BAD Logger logger = null; If (logger != null) logger.LogDebug(...); // GOOD Logger logger = NullLogger.Instance; logger.LogDebug(…)

Anti-Pattern: Hard Coded Constants

All constants in your code should be replaced by named constants/read-only variables What is difference between read only and const? All loops/data-structures/etc., should be able to handle “any size” of data.

Good Software Practices (cont)

Anti-Pattern: Premature Optimization

Don’t write complex code to speed up a potential bottleneck. _DO_ use well known data structures and worry about Big O (Complexity). Optimize when you have empirical evidence that what you think is the bottleneck, actually is. Next Lecture → Profiling

Rigor

Documenting Subdividing (separation of concerns) Big methods broken into multiple small methods Big tasks broken into multiple small tasks Hard Tasks “approximated” first Naming Debugging

Software Practice Principles

Understand the problem you are trying to solve Seek clarification of requirements early Approach each step in isolation Toy programs Separation of concerns Iterate - small steps Use versioning often

MAUI/GUI

Dispatcher

Dispatcher Puts requests for GUI changes into a queue When the GUI thread “has a chance”, take care of requests in queue Always runs on GUI thread Warning - More than one thread (GUI + networking(s)): LOCKING!!! If your GUI thread deals with an object that a networking thread also uses, you must protect the code using locks!

MVC in A7

Tendency is to write all of our logic together Assignment 7 Server “logics” Networking ← Model GUI ← View/Controller Connected Clients ← Model

Separation of concerns GUI server

Everything to do with storing multiple clients should be “abstracted” (i.e., put into a model class) Everything to do with Networking should be put into the Networking class Everything to do with the GUI should stay in the GUI class

MAUI/GUI (cont)

GUI/Network Multi-threading

Guis are generally multi-threaded, -unless your GUI doesnt do any processing that requires time, you must use a BG worker to do the work so the GUI remains responsive. Networking code is multi-threaded.

Invalidate

Invalidate i.e., PlaySurface.Invalidate(); Can be called anywhere (not just on GUI thread) Tell GUI Thread: At next opportunity, redraw the affected region Suggested Options Always (inside Draw Scene) → BAD :(“Burns” cycles On Message Received → Locked with Server → Okay Timer → X times per second → Good as well Especially if your client did “interpolation”!

Dispatching

All GUI updates must be done on the GUI thread (thread 1) When there are multiple threads in your program Any access to the GUI should be through the Dispatcher.

Basic Web Server ideas, e.g. HTTP/HTML/CSS

Web Server

Simply a program that listens on a port for a connection (e.g., from a web browser) and sends back web pages Using HTTP Hypertext transport protocol

HTTP Sequence

Point browser to a web server, e.g., http://www.webpage.com Browser opens socket to address Server accepts socket Browser sends text: “GET / HTTP/1.1” + stuff Server replies with text: “HTTP/1.1 200 OK …” + “<html>…</html>” Server closes socket – every HTTP request uses a new connection

Protocol

hwww.khanacademy.org hypertext-transfer-protocol-http Just a bunch of strings sent through sockets on the net… Request, e.g., GET / HTTP/1.1 Host: www.cs.utah.edu Response, e.g., HTTP/1.1 200 OK Content-Type: text/html; charset-UTF-8

Basic Web Server ideas, e.g. HTTP/HTML/CSS (cont)

HTTP Web Response

Just a string that follows the protocol, e.g., Header [New Line] Body. Example: HTTP/1.1 200 OK\n Connection: close\n Content-Type: text/html; charset=UTF-8\n \n <text or html>

HTML Body Example

<!DOCTYPE html> <!-- From w3schools –> <html> <body> <h1>My First Heading</h1> <p>My first paragraph.</p> </body> </html>

URL

Uniform Resource Locator scheme://host:port/path?query scheme → protocol (e.g. ftp, http, https) host → server name or address e.g. www.cs.utah.edu Port → 80/443 by default We will continue to use 11000

HTML Keywords

HTML very similar to XML Headers → h1, h2, h3, … Paragraph → <p> … </p> Image → <img src=”url” alt=”english”/> Format → <b> bold, <u> underline, etc. Logical Divisions → <div> Document → <doctype> Link -> <a …> Many more

Basic Web Server ideas, e.g. HTTP/HTML/CSS (cont)

URL -Path

scheme://host:port/path path → file/directory/page to access Optional Default is assumed to be file “index.html” Interpreting path is entirely up to the server www.cs.utah.edu/germain www.twitter.com/search https://www.google.com/search?q=abc&oq=abc&aqs=chrome..69i57j0l6j46.552j0j7&sourceid=chrome&ie=UTF-8

C#

Compiler Optimization in C#

In C#, we don’t have much control Optimization is done by the runtime (JIT) And partly by the CIL compiler https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net/ This is fine for most purposes C# is not the language of choice for (super) high performance…

Micro Optimization - Data Access

Accessing data in certain patterns (order) can be faster than others Caching Memory system Prefetching

C# (cont)

When to "Micro Optimize"

If you’ve picked all the right algorithms… …and profiled everything …and it’s still not fast enough Then maybe start micro-optimizing Learn more about this in CS 4400 Games and graphics need a lot of this

Profiling

First, determine where our program spends time. Every once-in-a-while, stop the program and examine where it is executing (Sampling) Accumulate samples over time – attribute them to methods “Once-in-a-while” is on the order of milliseconds

Logging

Logging Definition

Logging is the act of recording information about the what is happening during the run time of your program

Logs can be sent to one or more of the following:

Console Debug Display in Visual Studio GUI Files Databases Windows Event Log Etc.

Logging (cont)

Log Levels

Some information is more important (to certain people) than other. Too much information is often not useful. Log Levels Refers to how much and how “high a level” (or low) the information is E.g. Debug vs Critical Error

0 Trace

every little detail.. way too much information... turn this off most of the time Stakeholder: Developer - Something really odd is going on… Show me all the mouse X,Y values as the mouse traverses the screen

1 Debug

standard detail level... usually too much info to store... use this for development Stakeholder: Developer - Standard Development Cycle Show me every collision between two objects in the game

2 Information

high level information... for example "when a client connects"... use this to understand "meta-level" of application Stakeholder: Developer/Site Administrator Show me who logged into the system.

Logging (cont)

3 Warning

something is probably going wrong, but we haven't handled it yet Stakeholder: Developer/Site Administrator Show me when an unauthorized user attempts to access data

4 Error

something has gone wrong and we didn't know how to (or didn't want to spend the time) to handle it Stakeholders: Site Administrators, Project Lead, (Developer) Bank software cannot transfer to another bank

5 Critical

the application is about to die... can we do it gracefully? Stakeholders: Site Administrator, Project Lead Bank software “died” because database is not accessible

6 None

turn off messaging. Reasons to do: My code is perfect. Disk space is very expensive. (Note: neither of these are a reason to do this, nor are they true)

Logging (cont)

Where in Code to place Logging

Any “big” action Connection on the net Any “complicated” action For debugging complete message building Any “catastrophic” event Let developer manager know where to go back to look for problem

Example of levels to log messages at

Position of mouse _logger.LogTrace($ “{MousePosition}”); Partial messages coming over internet _logger.LogDebug($”Data Receive: {data}”); Information (or maybe just warning) New Chat Client Connected _logger.LogInformation($” {name} connected from {ip_address}”);

Setting Up Logger on a C# program

To (correctly) use Loggers and Dependency Injection, you must instrument the following places of your code: The Main Program The Class Constructor Class Methods

Threading

Async

Async methods should do “something” that may take a long time E.g., DB lookup, network connection, big computation

Threading (cont)

Await

await keyword tells program to “yield” cpu to other tasks while the long running operation completes Then resume at the given line when task is done WARNING: may be on a different thread

Networking and GUI → Threads

Network communications come in “Async” Handled on DIFFERENT threads GUI work MUST BE DONE on the GUI thread Solution: Dispatcher.Dispatch(( ) => { MessagesAndStatus.Text = $"Server Shutdown!" + Environment.NewLine; }

Which thread is running

Use: Environment.CurrentManagedThreadId Print this next to every log message to help you understand what is going on. If value != 1, then not the GUI thread! Also Remember: Debug Window Parallel Stacks

Threading (cont)

Race Condition

The Clients list is “shared” by all threads, thus becoming a Race Condition How do we fix this? Answer: locking! Warning for Server: Don’t “hold on to” client list while doing lots of work (e.g., sending messages to all clients) Copy client list (while locked) Then iterate over the copied list!

Race Condition Continued

GUI and Networking code use Threads to execute Any object that is referenced in methods that may be executing simultaneously in two or more threads MUST lock the data * Why is “lock the data” misleading in the statement above?

Dependency Injection

Key Ideas:	Code (objects) need other Code (objects) to work (or to be more effective) Databases, Loggers, UserManagers, etc.,
Where do those other objects come from?	Static Globals? Parameters?
Summary	Objects that need other objects to help do their job Want to be able to use different objects without modifying source code: Define functionality as an interface Pass object in at construction time (and save in class field) Important: only works for classes that the “system” builds for you E.g., you MAUI program MainPage!

JSON/Serialization

JSON Serialization -Default Option

What constructor is used BEFORE data put in: Default constructor What is serialized? Public properties What is not serialized? Fields, private properties How do we change that? Attributes (Metadata Tags)

JSON Serialization "Recursion"

Is it recursive? Yes - all objects inside public Properties are also serialized. Deep Copy (not shallow copy) “Cloning” ?? Circular References ??

JSON De/Serialization

Reflection Runtime - inspect code Deserialization Calls default constructor *(or constructor that matches field names in JSON string). “Finds” the names of the properties Reads the JSON looking for matching names Puts data directly into properties Serialization “Finds” names of properties and auto-generates json string

JSON/Serialization (cont)

JSON Attribute Tags and Options

[JsonConstructor] Specify the specific constructor to use Constructor parameters must match names of Json Fields [JsonInclude] Force property to be part of the serialization [JsonIgnore] For property to not be part of the serialization [JsonRequire] If field _not_ there then error

Parsing JSON ex

Food f = new Food(); List<Food> list = new(); list.Add(f); message = JsonSerializer.Serialize(list); JsonSerializer.DeSerialize<List<Food>>(message); DeSerialize(message, new List<Food>().Type()) {CMD_Food}[{X:50,Y:50,ID:5..},...]

JSON – Message Passing Optimization

Content of message (JSON string) is “too big” for value, i.e., we can compress the information in the JSON string into _far_ fewer bytes. What are we doing to do about it? Answer: Nothing

MVC

MVC for Agario

World status should be its own class (Model) GameClient interactions should be its own class Have important methods like: HandleMessageFromServer(…) SendMoveRequestToServer(…) Uses a World object Networking code Updates client with messages GUI code Reads client’s World information and draws it Need to make all references to client/world Thread Saf

MVC for Agario Continued

Start simple. Put most/all code in MainPage.xaml.cs As you start to see patterns, build helper classes to handle them Then in the later stages of development review code such that MainPage.xaml.cs is very short. See “refactoring” idea of extracting methods.

SQL

CS 3500

Purpose of CS 3500

What is it you are supposed to learning in CS 3500? CS 3500 is primarily about learning to: Solve complex problems with software Afterall, software rules the world And that, that software should Contains fewer defects Be more maintainable

CS 3500 (cont)

Maintainable/Testable Software

Simple (as possible) Organized/Architected (e.g., MVC) SOLID (e.g., Separation of Concerns/Single Responsibility) DRY Documented/Commented/Describable READMEs Pictures/Figures/UML Testable

Teamwork

Working with another developer Versioning, e.g., git usage Iterating Asking questions Discussing

Components software systems will need

Databases Networking (e.g., client/server architecture) Multi-threading GUI Logging

IDE

IDE support - Visual Studio Debug tools Optimization tools Database interface tools Multi-Architecture Deployment (e.g., MAUI) Intellisense Copilot?, GPT? XML documentation

CS 3500 (cont)

C# Programming

C# is one of many choice (albeit a good one) it could have been C++, or Ruby, or Python, or Java, or Go, or …. What are elements of the language that support GSPs? Long winded essay question possibility: “Describe multiple reasons we should use C# on our next project to support making maintainable software?”

Some Niceties of the C# Language

Functions as Values - Delegates, Event Handlers, etc. Lambdas, Closures Events - Design code around events “happening” Threading OOL, Library support Exceptions Garbage collection GUI Support Much much more.