CSCI E-97
Software Design:
Principles, Models, and Patterns
Lecture 1

September 1st, 2020

Outline

- Faculty
- Themes for the Course
- Course Information
- Use of Java
- Course Grading System
- From Programming to Architecture
- Design Principles
- The First Assignment

Faculty

Instructor:
Eric Gieseke

Teaching Assistants:
Hannah Riggs
Annie Kamlang-ek
Anindita Mahapatra
Joseph Carey

Themes for This Course

- We will be very much interested in defining:
  - **models** for our applications,
  - along the way, applying design **principles** and
  - design **patterns** as a means to refine our understanding of our application domain.
- Along with these comes practice in **documenting our designs.**
- And some **process**

Why Models?

- Historically, building architects and engineers have used models to represent their designs and understand the characteristics of their proposed solution, whether it be for a building, a bridge, an airplane, or some other thing.
- Mathematicians and scientists use mathematical models as an abstraction of real phenomena, and through manipulation of these models derive properties or make predictions of the behavior of the phenomenon under consideration (e.g., predicting the path and strenth of a Hurricane).
- Software engineers tend to use models much less formally.

Course Information

- This class is part of the Distance Learning program and will be available on the Internet within two days of the lecture. See the website http://extension.harvard.edu/distanceed/ for more details.
- There will be a weekly section, Thursday's at 7:20pm. This will be broadcast live and also recorded. Attendance is required.
- The first in class section will be Thursday, September 3rd.
- There are no exams for this course. The grades in this class will be based on assignments – the first few involve design and programming, and the last project requires writing a modular design document without doing the implementation
- Some quizzes and surveys to gauge progress

Where Things Are

- The course website URL is: https://canvas.harvard.edu/courses/79540/assignments/syllabus
- All material will be available on that site, except for Piazza discussion board.
- The course syllabus is available on the course website.
- Piazza site for class discussions is here: https://piazza.com/extension.harvard/fall2020/cscie97/home
- Links to live sessions and recordings can be found on the course web site.
- The course website and the videos will require a login after the second week. At that point, you will need to be registered in the class and you will have to activate your Harvard PIN.
- Course chat room: https://canvas.harvard.edu/courses/79540/external_tools/1

The Course Texts

- The textbooks for this class are:
  - *UML Distilled, 3rd Edition*, by Martin Fowler. The publisher is Addison-Wesley. This provides good coverage of UML 2.x
  - *Head First Design Patterns*, by Eric Freeman & Elisabeth Robson, The publisher is O'Reilly. Will help you understand design patterns and how they help to reinforce design principles.
- The rest of the course material will include the class notes as well as some links to other sources for information.

Students

Total Students: 78
Please update your bio on the course web site and introduce yourself to the class.
You will be participating in design reviews with your classmates.

What You Should Know About Java

- Core syntax
- Java packages java.lang, java.util, java.io (or java.nio)
- Exception handling
  - This includes catching exceptions at the right place in your program, generating exceptions specific to the application you're writing, and actually handling them in a meaningful way
- Using package and import statements – setting up CLASSPATH as needed to find classes
  - This is an important point for this course. We will always use a package structure that is at least two levels deep (e.g., com.cscie97.ledger), which requires that you can manage the appropriate directory structure and submit assignments that compile 'out of the box'
- Creating and using JAR files

Development Environments and Tools

- We will do all programming assignments in Java, using the Java JDK 14.
- If you use an IDE (Eclipse, Intellij) for your assignments, you must test that your program works with the command line of the basic JDK 14, which is what we will use to compile and run your program. Sometimes you have to recompile outside of the IDE in order to be sure.
- There are a host of UML tools. For this class, we recommend that you use Astah. There is a free student academic license available. With your harvard email address, request a student license here.

  http://astah.net/student-license-request

General Grading Criteria for CSCI E97

- All assignments with programming will be graded according to two characteristics:
  - Design and documentation
  - Implementation of the design and execution of code
- Design and documentation addresses how well you structure your programs and how well you describe them. Learning how to create quality designs to produce high quality software is the focus of the course.
- Implementation of the design and execution of code addresses how well your program runs per instructions, given and how it delivers the functional features required in the assignment. Javadoc style comments are mandatory for all classes and methods.
- Your solution will be given a point grade based on a grading sheet that will be shared with the assignment.

Grading Example

- For each assignment, we will provide a grade sheet that captures specific criteria (for instance, if we are demonstrating the use of the Abstract Factory Pattern, we'd expect to see that reflected in the design document).
- Suppose your solution runs according to the problem specification, has all required features, and some nice 'extras'. However, while reading the code we find that you do a poor job of structuring your code and packages, so that the code has many extraneous lines (e.g., because you could have factored some common code into a parent class rather than copying it into many similar classes) and your exception handling code is not complete.
- In reading the design documentation, we find that your solution is poorly designed, has only cursory descriptions of classes, and the models are not complete.

The Writing Component

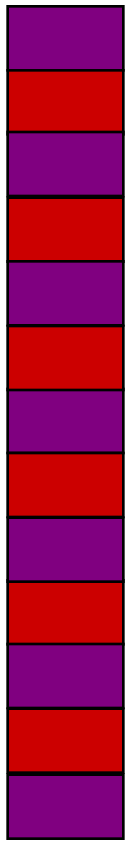
- Communication can be as important a skill as design and programming.
- Helping others understand and visualize your ideas is a valuable skill.
- Software engineers are asked to present (and defend) their designs and to write design documents that others can follow through implementation.
- Design reviews are an important component of delivering software with high quality. Students will participate in peer design reviews for each of the design assignments.

Why is it that We Build Applications?

The main reason to build applications is to implement a business process

A corollary is
Know Thy Business Requirements

Two Sides of Defining a Solution



Requirements

User Stories

Use Cases

Non Functional Requirements

Business Plan

Conceptual Model

Business Process Definition

Prototypes

Key Performance Indicators (KPI)

Technology Choices

- DBMS
- App Server
- Language
- Big Data?
- Cloud?
- Microservices
- Containers...

Tools

Implementation

Functional Testing

Performance Testing

Deployment

Process* Definition

* – The Process on this side of the wall is the software engineering method

From Programming to Architecture

- What differentiates programming from design and design from architecture?
- The *conceptual model* that's common these days when discussing software engineering includes requirements analysis and moves through multiple phases to user acceptance test, then to deployment and maintenance
- Somewhere along the way, some architecture, design, and even some programming happens
- The waterfall model treats each of the above as a phase in a software engineering project
- There are also spiral or iterative development methods
- In this class we will use a “Design First Agile” process

The Conceptual Concerns in Programming

- The input to the programming process is some kind of design for a set of modules (classes or packages) that usually require understanding of the intent of the system being built
- The conceptual concerns of someone writing code
  - Language syntax and semantics
  - Choosing the right data structures and algorithms to implement the intent of a given class
  - Writing and testing modules (classes or packages) that satisfy specific requirements
  - Source control and release management
- The output of the programming process is code that compiles and works with the code produced by others and meets the requirements specified

The Limits of Programming Languages

- At the programming language level, there are a few major choices for object oriented development (out of hundreds of languages)
  - C#, C++, Java, …
- Programming languages are a very poor way to describe *what a program is supposed to do*– you should never try to explain to a customer what your application does by showing them the code
- Methods on a 'business object' tell you what an instance of a class will do for you, but it's difficult to glean from the methods how this class contributes to the implementation of one or more business processes
- Data representation is a huge task (which collection classes are appropriate, how to control access to data, in what scope should data objects exist?)
- Despite the existence of garbage collection, there are still memory management issues to deal with, not to mention threading, asynchronous invocations, and callbacks

How Many Languages Does it Take to Build a System?

- There are plenty of other languages programmers get involved with as well
  - HTML
  - SQL
  - JavaScript
  - Python
  - XML, JSON, Yaml
  - DSL scripting languages
- Plus there are lots of tools used in development, including, but not limited to Jira, Maven, Grail, Ant, Make, Nmake, Git, configuration tools, Grep, Awk, Liquibase, IDEs, …
- Each of these play supporting roles in development. If you think of yourself as a Java programmer you still need to let several of these other tools or languages claim some of your mind-share

What is Software Design?

- Software design is commonly described as a process of making a plan for the software elements required for implementing a system
- The IEEE Computer Society defines "design" as follows:

> **design: 1.** The process of defining the software architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements. 2. The results of the design process.

What Does Software Architecture Provide?

"… it is necessary to separate the architecture, *the definition of the product as perceivable by the user, from its implementation*. Architecture versus implementation defines a clean boundary between parts of the design task, and there is plenty of work on each side of it"

Frederick Brooks,*The Mythical Man-Month*,Anniversary Edition, Addison Wesley, 1996 pg. 256

Differentiating Design from Programming

- The input to the design process is some set of requirements and perhaps an architecture that lays out how the system's components will be built to meet the requirements
- The conceptual concerns of the design process are
  - What are the functional requirements for the system, component, or module to be built
  - What are the technical (non functional) requirements for the system, component, or module to be built (performance, stability, availability, security, extensibility, manageability, resource constraints)
  - What functional components are appropriate to meet these requirements?
  - On which infrastructure elements will this be built?
  - What set of tools and languages are appropriate for construction?
- The output of the design process is a description of a solution (with diagrams and text) and occasionally some working prototypes

How Are We Going to Approach Design?

- There are many methods that prescribe how to behave like software engineers
  - We will think about design independently of the software engineering method
  - However, we will apply software design within an agile context
- There are many ways to implement, in code, a working system
  - We will think about design independently of programming language; in fact, the last assignment requires you to design an application without writing any code
- There are several techniques for defining a software solution to a problem
  - We will focus on Object-Oriented Design (as opposed to focusing on business process models , data flow models, or entity-relationship models)

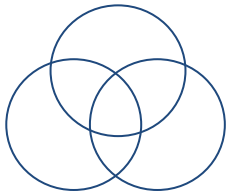
If Programming has Languages, What Does Design Have?

- Think about the semantics first: What is it one is trying to express in an Object-Oriented Design? The answer to this is given as an "abstract object model" – it answers the question of what 'things' we can put into our model
- The much, much easier part is how we express that abstract object model syntactically
- In this course we will use the Unified Modeling Language (UML), which is a standard of the Object Management Group (see www.omg.org)
- Application of design principles
- We'll also discuss Design Patterns, which provide a vocabulary for discussing proven designs that solve well known problems and apply design principles
- Collaborative brainstorming, design reviews, and functional reviews
- A key criteria is that you should be able to express your design in an implementation-independent fashion

Multiple Scopes for Design

- 'Scope' is not an official piece of the software glossary. I use it here to help identify the size of the problem we're trying to solve in our design.
- Enterprise scope covers multiple applications and components that typically execute in heterogeneous environments. Distributed Computing, Business Process Management (BPM), Service-Oriented Architectures (SOA), Microservices, and Enterprise Application Integration (EAI) are hot topics here
- Application (or component) design looks at breaking down applications or components into implementation modules
- Detailed design is typically about mapping application or component designs into working code
- Note: The design principles we will learn work at all levels!

Three Enabling Design Principles



- ***Abstraction*** is about finding 'the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries relative to the perspective of the viewer' [Booch, Object Oriented Analysis and Design with Applications, 1994 – see also Buschmann et al, Pattern-Oriented Software Architecture, 1995]
- ***Divide and Conquer*** is about breaking a large complex problem into smaller problems that are more easily addressed
- ***Separation of Concerns*** is about separating different responsibilities among different types. For instance, presentation is not a responsibility of a domain object; to be more specific, a Customer object does not know how to present itself – that responsibility is left to a presentation object. Similarly, a Customer object does not know how to read and write itself to a persistent store (a file or a database)

Two Principles for Reuse

- **Program to an interface, not to an implementation**
  - In C++ and Java this means using only the methods declared as public, restricting the use of data members to read/write access methods, and limiting the use of protected methods and (in C++) 'friend' functionality
- **Favor composition over inheritance**
  - This means to look for opportunities to supply an object at run-time with the desired functionality rather than constraining the choice of object to use to compile-time inheritance
- **Big Note:** The use of the term 'interface' is overloaded. The term has been around for a long time, pre-dating object-speak. When used in a general setting, it refers to a collection of functions or operations that a module provides

‘Interface’ is an Overloaded Term

- In the fully-qualified form, there are three variations of importance to us:
  - *Programming Interface*, also known as an API. This was what the ‘interface’ principle meant on the preceding slide
  - *User Interface*. This is an interface that allows a user to interact with a program. A Graphical User Interface (GUI) is a special case
  - *Java Interface*. This is a type that is specific to the Java language

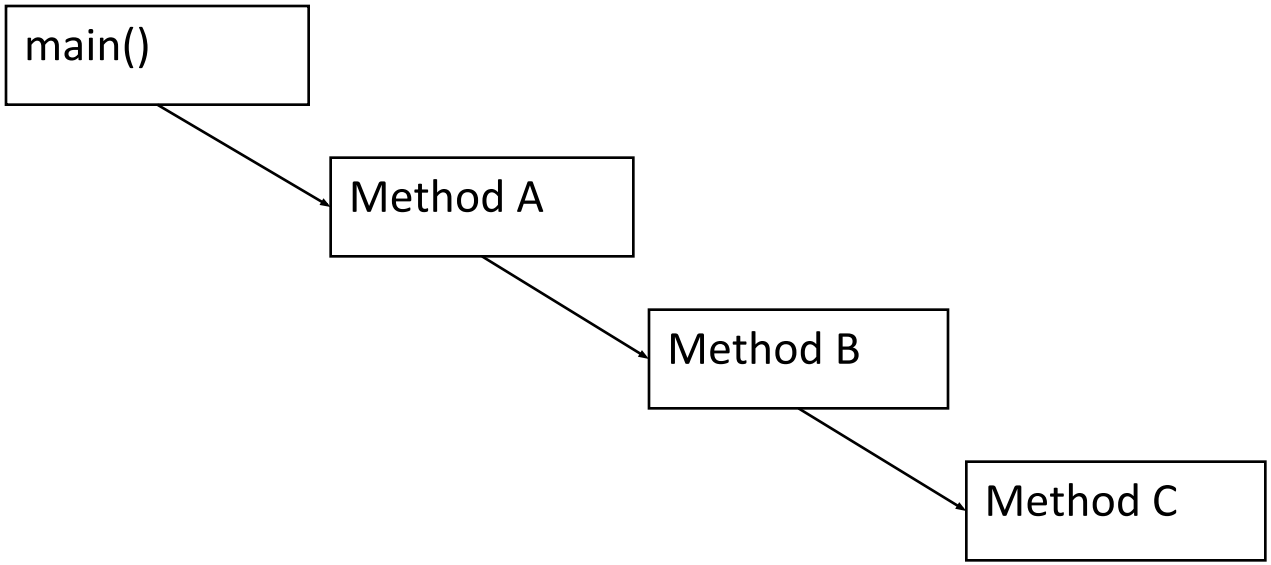
Exceptions and Exception Handling

- There's a distinction between writing code that implements the 'one true path' and code that works in the presence of various kinds of failures
- Java, C++, and C# provide exceptions as a means of helping programmers deal with problem conditions
- Here's James Gosling talking about the programming cultural problems of dealing with failure conditions

It really is a culture thing. When you go through college and you're doing assignments, they just ask you to code up the one true path. I certainly never experienced a college course where error handling was at all discussed. You come out of college and the only stuff you've had to deal with is the one true path. You get a job working in some IT department's data center, and they're using the software in production runs. If it doesn't work, it's a real problem. All of a sudden there's all this painful stuff that you don't like to do, and you're feeling grumpy about it because this isn't as much fun as just writing clean code.
There's really nothing a hacker likes more than having an empty editor buffer. You can just start to write code. That's the way university assignments are: here's a problem, just write the code. The real world is much more complicated.

Exceptions, Abstraction, and Separation of Concerns

- Exceptions have data about a condition that prevents a method from satisfying its contract with a client



- What happens if Method C encounters a FileNotFoundException? Toy examples might do a simple println() to report the result…

Designing Exception Handling

- Here a few questions that need to be answered to do a good design
- What kinds of things can go wrong that prevent the application from doing what it's supposed to do?
- What is the right set of data to pass along when an exception occurs
- What class is responsible for
  - catching the exception
  - for trying to correct the problem if that’s possible
  - for logging the occurrence of the problem
  - for showing the end user a description of the problem in a meaningful way
- So, you have to come up with the right abstractions for what the exceptions in your application domain are
- You also want to ensure that exceptions allow separation of the data about the exception from the presentation – in particular this means *don't lump everything into one big string, instead add attributes to the exception class to capture the details.*

Exception Handling Guidelines

- Define a specific subclass of java.lang.Exception that is relevant for your application domain; then define appropriate subclasses. This assumes you have done some careful error analysis and abstraction
- The base exception class and subclasses should have discrete data members that can be used or ignored in generating an error message
- Each package can have exception classes relevant for methods in the package's classes
- Exception handlers are typically most appropriate at higher levels of the food chain – don't have catch blocks at the lower levels that are empty, print a message to the console, or worst of all, exit
- In some cases, a lower-level method might be able to recover from an error, such as a timeout in a network call, and there's no need to notify the end user
- Decide how you want to deal with unchecked exceptions (e.g., NullPointerException)
- Do not rely on a one-size-fits all catch block; e.g.,

  catch (Exception e) {…}

Notes on Writing Technical Documents

- Why do we write design documents?
  - to communicate
  - to improve our own understanding
  - to streamline implementation
  - to improve the quality of our software
- The three questions we have to ask ourselves are
  - What is the intended audience?
  - What's the topic?
  - What is it that I know that I can say?
- Once you have these squared away, you can begin to **design** your document
- Fortunately, there are principles and patterns that can help with your writing
- Be explicit and provide details. Don't leave it up to the imagination of the reader.

Elementary Principles of Composition – 1

- These follow the advice from Chapter II of Strunk & White (2000), *The Elements of Style*, 4th edition, Longman

12. Choose a suitable design and hold to it.

- It is amazing how closely the process of constructing a system matches the process of constructing a document to describe that system
- In a free-form world, you can take the approach that you'll answer the three questions on the previous slide and then design your document. In reality we are often required to follow a 'standard' template.
- The authors say "...in most cases, planning must be a *deliberate* prelude to writing." and refer to this rule as the first principle of composition
- Mind maps can help organize thoughts and create an outline for a design. See https://www.mindmeister.com/

Elementary Principles of Composition – 2

13. Make the paragraph the unit of composition.

- "As a rule, begin each paragraph either with a sentence that suggests the topic or with a sentence that helps the transition. If a paragraph forms part of a larger composition, its relation to what precedes, or its function as a part of the whole, may need to be expressed."
- Apply this recursively; that is, apply this to a chapter by having an initial paragraph introduce the topic or topics of the chapter, and apply this to an entire document by having an introductory chapter that introduces the topics to be covered and describes how the rest of the document is structured
- Add descriptive text to explain diagrams.

Elementary Principles of Composition – 3

- 14, 15, and 16 require us to make a commitment to being clear about the details.

14. Use the active voice.

- It's much clearer, and more accurate, to write

"When the method detects an error, it throws an XYZ exception that the client uses to display a message to the user."

than to write

"When an error is detected a message is shown to the user."

15. Put statements in a positive form.

- In our kind of writing, use simple declarative sentences.

16. Use definite, specific, concrete language.

Avoid vague expressions. Imagine a use case where the writer begs the question of what happens by writing

"The user creates a profile"

with no details about what goes into the profile

For the Next Class (9/8)

- Read Chapters 1-3 in Martin Fowler's UML Distilled book. Chapters 1 and 2 provide background on UML and the development process
- Set up your Java 14 environment
- Start working on assignment 1
- Fill in some notes about yourself in the People page of the Course website: https://canvas.harvard.edu/courses/63155/users
- The assignments of this class will require most of the time provided, so get an early start! They also build on each other.

Assignment 1

- Assignment 1 is available on the website
- It is intended as a warm-up exercise and litmus test for your Java programming skills
- Written as a software design document for you to implement. You will be writing your own design documents in the following assignments.
- We'll be discussing this tonight.