Problem Solving
Object Oriented Design
Systems Analysis

LEARN TO WRITE BEAUTIFUL CODE BY HAVING BEAUTIFUL ARCHITECTURE
DR. WILLIAM L HONIG

We will look at a quick 4 step process to learn the basics....
Each of these steps might be a whole course

Some materials originally from Robert V. Stumpf and Lavette C. Teague,
Object-Oriented Systems Analysis and Design with UML
Pearson Prentice Hall (2005)
Course Summary
Key Things and Process

Entry Criteria:
Good Programmer
Know the Basics

Know the concepts
Understand basic OO
Able to write small to medium programs

Read
Write
Discuss
Improve

Tools
Data Structures
Recursion

Technique
Design
Algorithms
UML
Testing

Beautiful Code Producer

Which do you want to be:
Hacker: Write lots of code, get it to run, test it until you get reasonable outputs for some inputs.
Alternative: Write concise correct code, constructed so that it is likely correct, uses best practices and looks good.

Problem Solving…Beautiful Code
Use UML to Plan an Architecture to Get There

A QUICK AND USEFUL ANALYSIS AND DESIGN PROCESS FOR “THINK BEFORE CODE”

1. Understand the requirements
2. Identify the key system events (and associated actors, systems, …) or use cases
3. Create a UML class diagram for the system with details of key class members and associations between classes
4. Create a UML sequence diagram for key events or use cases

LIKELY AN ONGOING AND ITERATIVE PROCESS
1. Hard work! Expect to repeat and retry
2. Not done until you can “see” the code!
Humans cope with complexity by thinking in terms of systems. A system organizes its components into a structure and is separated from its environment by a system boundary.

What Is a System?

A **system** is an interrelated set of components which are viewed as a whole.

It has:

- **Components** – its basic parts
- **Structure** – how the components are organized
- **Function** – what the system does
- **Objectives** – the human purposes served by the system
Step 1 and 2
Understand the requirements and the key system events or use cases
Event analysis identifies the events to which the system is expected to respond, names the inputs and outputs, and identifies the actors – those who interact with the system by providing inputs and receiving outputs.

Event-Driven Systems

Event analysis takes a stimulus-response perspective –

- The system does nothing until it is triggered by an event.
- When an event occurs, the system responds as completely as possible.
- After the response is complete, the system waits until another event occurs.
Event Definition

- **ACTOR**
  - Provide inputs and receive outputs
  - Person, System, something in the environment
    - May "cause" an event

- **INPUT**
  - Information entering the system
    - "data flow" events

- **OUTPUT**
  - Information leaving the system
    - Usually back to some actor
    - Not all events have output

Identify the Business Events

Event List for the Public University Registration System

- **External**
  - 1. Department submits class schedule

- **Temporal**
  - 2. Time to produce university class schedule

- **External**
  - 3. Student registers for classes

- **Temporal**
  - 4. Time to produce class roster
Identify the Actors, Inputs, and Outputs

Who supplies system inputs?

- **Department** submits a Department Class Schedule.
- **Student** supplies a list of desired classes (a Registration Request).

Identify the Actors, Inputs, and Outputs (continued)

Who receives system outputs?

- **Departments, Professors, and Students** receive the University Class Schedule.
- **Student** receives a Class List.
- **Professors** receive Class Rosters.
### Event Table

**FIGURE 3.6**

<table>
<thead>
<tr>
<th>EVENT NUMBER</th>
<th>EVENT DESCRIPTION</th>
<th>SYSTEM PROVIDING INPUT</th>
<th>ACTOR RECEIVING OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Department submits class schedule.</td>
<td>Department Class Schedule</td>
<td>Department</td>
</tr>
<tr>
<td>2</td>
<td>Time to produce class schedule</td>
<td>University Class Schedule</td>
<td>Student Department Professor</td>
</tr>
<tr>
<td>3</td>
<td>Student registers for classes</td>
<td>Registration Request</td>
<td>Student Student Class List</td>
</tr>
<tr>
<td>4</td>
<td>Time to produce class roster</td>
<td>Student Class List</td>
<td>Student Class Roster \ Professor</td>
</tr>
</tbody>
</table>
Components of a Design Class Diagram

A design class diagram follows the same UML graphic conventions as a domain model.

A design class diagram shows:
• Classes and class hierarchies
• Attributes
• Operations
• Whole-to-part associations
• Qualified associations

Qualified Associations

A qualified association associates two objects using a qualifier to select objects at the other end of the association.

A qualifier is an attribute or set of attributes which has a unique value for each object in the class.
A pattern is a named statement of a design problem together with its solution and guidance for applying the pattern. Patterns include:

- Façade
- Creator
- Expert
- Singleton
Thinking OO and Architecture
Use some common method names

A GOOD DESIGN PATTERN = REUSE COMMON METHOD NAMES FOR TYPICAL FUNCTIONS

1. To check the validity of a name or identifier and probably convert to a handle for the related object. Usually in a container for that Something.
   someObject verifySomething( suitable parameters…. )

2. To construct an object, use either the Java technique of a possibly over loaded class name or
   someObject createSomething( suitable parameters…. )

3. To set up a new something in a container, usually in the Creator pattern class
   boolean makeSomething( suitable parameters…. )

4. To fill in one end of an association; usually in an Expert for the thing being associated with
   boolean associateSomething( object being linked )

EXAMPLES FROM A UNIVERSITY REGISTRATION SYSTEM

Registration System
   Student verifyStudent( int studentIdNumber )

Student
   createStudent( int studentIdNumber, String givenNames, String familyName, LocalDateTime dob… )
   Student( int studentIdNumber, String givenNames, String familyName, LocalDateTime dob… )

Semester
   boolean makeSection( Course course, Room meetingPlace, … )

Section
   boolean associateProfessor( Professor instructor )

Step 4
Sequence Diagram showing who calls whom with what information, what happens, and how results get made

Do for one event or use case
Probably should do them all...but at least the hardest ones!
Visibility: For an object (the **client**) to send a message to another object (the **server**), the receiving object must be **visible** to the sending object. (That is, it must know the server’s identity).

There are four ways to obtain visibility in an object-oriented system:

1. **Reference visibility (Navigability):** The client object has a pointer or reference to the server object.
2. **Parameter visibility:** An object is provided by a message as a parameter.
3. **Local visibility:** An object obtains visibility to another object by declaring it inside one of its methods.
4. **Global visibility:** An object is obtained from a class by an object requiring visibility to it.

NEVER ask someone to confirm their own identity.
How do you know you are done?
And ready to code?

Some one much dumber than you could write the code given this information and detail.

All they have to do is follow the footsteps I have clearly set out.
Anyone could follow this path without getting lost.