Chapter 9: The Pseudocode Programming Process

Summary of Steps in Building Classes and Routines

- Iterative process
- General design
- Enumerate specific routines
- Construct specific routines
- Check class construction as a whole

Steps in Creating a Class

- Create general design for the class
  - Design specific responsibilities
  - Define "secrets"
  - Determine if to derived from another class
    - Or if another class will be derived from it
  - Identify key public methods
  - Iterate through all these steps
- Construct each routine within the class
  - After determining major routines on the first step, create them
  - This usually unearths more routines that need to be made
- Review and test the class as a whole
  - Test each routine
  - Test whole class as a standalone unit

Steps in Building a Routine

- Designing routine
- Checking design
- Coding the routine
- Checking the code

Pseudocode for Pros

- Guidelines for using effectively
  - Use English like statements that precisely describe specific operations
  - Avoid syntactic elements from the target language
    - Your writing pseudocode to avoid syntactic elements lmao
  - Write pseudocode at the level of intent
    - Describe meaning of approach, not how it will be implemented
  - Write pseudocode at a low enough level that generating code from it will be nearly atomic

        - if pseudocode is at too high of a level, it can gloss over problematic details in the code
- Once the pseudocode is written, you build the code around it and the original pseudocode becomes the comments
- Benefits
    - Pseudocode makes reviews easier
        - design details can be reviewed without slogging through code
    - Pseudocode supports idea of iterative refinement
        - At high level design, can catch high level errors
        - At mid level pseudocode, can catch logic errors
        - At low level programming, can catch programming errors
        - No overlap so easy to focus
    - Pseudocode makes changes easier
        - A few lines of pseudocode is easier to change than a page of code
        - Erasing a line on a blueprint instead of tearing down an entire wall
    - Pseudocode minimizes commenting effort
    - Pseudocode is easier to maintain than other forms of design documentation

Constructing Routines by Using the PPP

1. Design the routine
2. Code the routine
3. Check the code
4. Clean up loose ends
5. Repeat as needed

Design the Routine

Preliminary

1. First, check the reqs, and make sure the routine is useful and defined
2. Define the problem the routine will solve
    a. The information the routine will hide
    b. The Inputs to the routine
    c. Outputs from the routine
    d. Preconditions that are guaranteed to be true before routine called
    e. Postconditions the routine guarantees before passing control back to the caller
3. Name the routine
4. Decide how to test the routine
5. **Research functionality available in the standard library**
    a. **Single biggest way to improve code and productivity is to reuse good code**
6. Think about error handling
7. Think about efficiency
    a. If efficiency isn't critical, make sure routines interface is well abstracted and the code is readable

    b. If efficiency is critical, design routine so resource and speed goals can be met
    c. Its usually a waste of time to work on efficiency at the level of individual routines. Big optimization comes from high level design, not the individual routines
8. Research algorithms and data types
    a. Before writing complicated code from scratch, don't, look it up
9. Write pseudocode

High Level Pseudocode

1. Start with the general and work towards something more specific
    a. Most general is the header comment of a routine describing it
        i. Trouble writing the statement is a warning that you need to understand the routines role in the program better
        ii. If its hard to summarize, assume something is wrong
2. Think about the data
    a. Good to think about the major pieces of data before logic
3. Check pseudocode
    a. Take a step back and just think about it
4. Ask someone else to look at it or have you explain it to them
    a. People are more willing to review a few lines of pseudocode than 35 lines of C yknow
5. Make sure you have an easy and comfortable understanding of routine
6. Try a few ideas in pseudocode
    a. Iterate and keep best

Code the Routine

1. Write routine declaration
2. Write the first and last statements
    a. Also note all assumptions about the interface, in and out
3. Turn pseudocode into high level comments
4. Fill in code below each comment
5. Check the code
    a. Sometimes code below each comment line will explode
        i. Option 1: Factor code into a new routine
        ii. Option 2: Apply PPP recursively
6. Clean up leftovers

Really Checking the Code

- Problems may not appear until routine is fully coded
- Mentally check routine for errors
    - Mentally executing a routine is difficult, so keep routines small
- A working routine isn't enough. If you don't know why it works, study it
- Compile/run for sure lmao

    - You should fully expect it to work in entirety instead of incrementally hacking together lines and testing ever 4s
- **The point is to rise above the cycle of hacking something together and running it to see if it works**
    - **Compiling before you're sure your program works is often a symptom of the hacker mindset**

Clean up Leftovers

- Check routines interface
    - Make sure all IO data is accounted for and parameters are used
- Check for general design quality
    - Make sure routine does one thing and does it well
    - Loosely coupled
- Check the routines variables
    - Inaccurate variable names, unused objects, etc
- Check routines statements and logic
- Check routines layout
    - Make sure you've used white space to clarify logical structure, expressions and parameter list
- Check routines documentation
    - Make sure pseudocode translated into comments is still accurate
    - Check for algorithm descriptions
    - Documentation on interface assumptions
    - Non-obvious dependencies
    - Justification of unclear coding practices
- Remove redundant comments

Repeat all as needed haha

Alternatives to PPP

- Test-first development
- Refactoring
- Design by contract
    - Each routine has specific pre and post conditions