High-Level Learning Outcomes

After this course, students should be able to:

I. Define and apply definitions of key terminology related to programming and computing.
II. Read, write, and apply elements of a general O.O. programming language (e.g., Java)
    A. Write declarative statements and use operators, conditional statements, and loops to solve specific tasks
    B. Write (Java) methods and classes
    C. Use (Java) primitive types and use their encodings
    D. Learn about collections, including arrays and Java's ArrayList class
    E. Master basic exception handling (both throwing and handling them)
    F. Use appropriate tools (e.g., Eclipse) to enter and run programs
III. Solve and document problems of complexity illustrated by example projects
    A. Design algorithms to accomplish tasks described in natural lanugage.
    B. Learn to apply "systematic program design" to go from a problem or concept to an actual implementation.
        1. Object-Oriented approach
        2. Identifying design recipes from problem statements
        3. Design test case scenarios that can be used to verify the accuracy of executable code
    C. Write code to implement an algorithm described in natural language or pseudo-code, implementing appropriate methods and classes, as necessary.
    D. Regularly write informative comments that tie code back to description of a problem or algorithm
    E. Identify and solve problems where a recursive approach works well.
IV. Trace, test, and debug a program
    A. Recognize syntactic errors in a block of text that is supposed to represent a valid Java program
    B. Find and fix semantic software bugs using tools available in their programming environment
    C. Trace given sections of code by hand to predict the output
    D. Trace given sections of code and draw detailed memory diagrams including the call stack and heap
    E. Explain the importance of unit testing, and incorporate it as an essential component of software development
        1. Develop working familiarity with JUnit
        2. Learn to write concise tests for general cases
        3. Learn to identify and test corner cases
        4. Explain the limitations of unit testing