ICSI 201/IECE 141 Midterm exam study guide
Fall 2022

To prepare for the midterm exam, students should be comfortable with the following concepts:

1. Chapter 1: Introduction
    a. How hardware and software work together to support computing.
    b. What is a compiler?
    c. What is the Java Virtual Machine and how does it support code portability?
2. Chapter 2: Java fundamentals
    a. Differentiate between variables and literals. Determine if variable names are legal/illegal.
    b. What are the eight primitive data types in Java?
    c. How do primitive data types use the computer's memory?
    d. Identify the parts of a java program including class header, class body, method header, method body, variable declaration and initialization, method call vs. method declaration.
    e. Understand integer division and its implications on rounding errors.
    f. Variables scope in the context of a single method, across methods and across classes.
3. Chapter 3: Decision structures
    a. Basic operation of if/if-else statements
    b. How to use logical operators to create complex conditionals
    c. How to compare string objects (and why is this different than comparing primitive data types)
    d. How and why to use the switch statement
    e. Understand flowcharts of decision structures. Be able to draw the flowchart for a given decision structure code and vice versa: be able to write the code given the flowchart for a decision structure.
    f. Understand and follow proper syntax and indentation for decision structures.
4. Chapter 4: Loops
    a. Why are loops necessary?
    b. What types of loops exist and why do we need them? When is a while loop applicable? When is a for loop applicable? When is a do-while loop applicable?
    c. Given a loop code, anticipate what the output of the program will be and vice versa, given a desired program output, select an appropriate loop type and write down the code for it.
    d. Loop flowcharts. Flowcharts for input validation. Convert a flowchart to Java code and vice versa, convert Java code into a flowchart.
    e. Variable scope in loops.
    f. Nested loops. Why we need them and how to create flowcharts for them. Given a problem, determine whether it can/must be solved with a nested loop or not.
5. Chapter 5: Methods

a. Why do we need methods? How do methods help us “divide and conquer” code development and support “code reuse”?
b. The main() method, Java API methods and custom methods: compare and contrast.
c. Identify the components of a method: header and body.
d. Determine the scope of a method depending on the keywords in a method’s header.
e. What are the roles of method caller and callee? Identify the caller/callee in a code example.
f. Determine the sequence of code execution in snippets of code with methods.
g. Understand the difference between value-returning and non-value returning methods. How can we use the method header to find if a method is value-/non-value-returning?
h. How can we pass arguments to a method?
i. Why do we say that method arguments are passed by value?
j. Variable type compatibility for methods that take arguments as input.

6. Chapter 6: A first look at objects and classes
   a. How is object-oriented programming useful?
   b. What is the relationship between objects, classes and instances?
   c. How are objects stored in memory? Is this different than storing a primitive variable in memory? If yes, why?
   d. Identify the components of a class definition: header and body.
   e. Identify the members of a class: fields and methods. What they are and how they are different?
   f. How do keywords in the header and body determine the scope and accessibility of the class, methods, and fields?
   g. What are UML diagrams? How can they help us structure the idea of our code before we begin implementing?
   h. Accessor vs. mutator methods.
   i. What is the syntax to create an object of a given class? What is the syntax to set an object instance’s fields and call its methods?
   j. How do classes help hide data? Why is this important?
   k. How to avoid stale data when implementing and calling methods?
   l. How are instance methods different than normal methods?
   m. What are constructors? Identify/implement the constructor in a code example. How can we use a constructor to initialize instance fields? Default vs. custom constructors.
   n. Overloading methods and constructors: how and why?
   o. What is variable shadowing? Is it a good practice? Why? Why not?
   p. What are Java packages and what does the import statement do?
   q. Using our discussion of object-oriented design (i.e. finding classes and their responsibilities), consider a task specification and fill out a UML diagram to implement one class for the task.