CS 122 Midterm Exam - Study Guide

Hello everyone,

Here is a study guide to help you prepare for our upcoming midterm exam. The goal of this guide is not to provide all the answers, but to structure your review of the material we have covered so far. A strong understanding of these topics will prepare you well for both sections of the exam.

Exam Format Reminder:

- **Section 1: Multiple-Choice & Short Answer:** This section will test your conceptual understanding of key terms, syntax, and programming principles.
- **Section 2: Programming Challenge:** This section will assess your ability to apply these concepts to solve a practical problem by writing Java code. You will be allowed to reference the online textbook for this portion.

Use this guide to identify areas where you feel confident and topics that may require more of your attention. Good luck with your studies!

Table of Contents

---

1. Introduction to Java

This section covers the foundational concepts of the Java language and programming environment.

- **Key Concepts:**
  - High-level language
  - Java Virtual Machine (JVM) & Bytecode ("Write Once, Run Anywhere")
  - Basic anatomy of a Java program: `public class`, `main` method
  - Syntax: Semicolons, curly braces `{}`, case sensitivity
  - Comments: single-line `//` and multi-line `/* */`
  - Error Types: Compile-time, Run-time, and Logical errors

- **Guiding Questions:**
  - What is the purpose of the `main` method? What are its four components (`public`, `static`, `void`, `String[] args`) and what does each mean?
  - How does the Java compiler use bytecode? Why is this important?
  - What is the difference between a compile-time error and a run-time error? Can you give an example of each?
- **Code to Review:**
  - Be able to identify the parts of a basic "Hello, World!" program.
  - Practice writing simple `System.out.println()` statements.

2. Variables, Data Types, and Input

This section focuses on how Java stores and manipulates data.

- **Key Concepts:**
  - Variable Declaration vs. Initialization
  - Java's 8 Primitive Data Types: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`
  - `String` as a non-primitive (object) type
  - Naming conventions (camelCase for variables, ALL_CAPS for constants)
  - `Scanner` class for user input (`import java.util.Scanner;`)
  - `Scanner` methods: `.nextInt()`, `.nextDouble()`, `.nextLine()`
- **Guiding Questions:**
  - Why is Java called a "strongly-typed" language?
  - What is the difference between `int` and `double`? When would you use one over the other?
  - Explain the difference between a `char` and a `String`. How do their declarations differ (single vs. double quotes)?
  - What is a common issue when using `.nextInt()` followed by `.nextLine()` and how do you resolve it?
- **Code to Review:**
  - Practice declaring and initializing variables of all 8 primitive types.
  - Write a simple program that asks a user for their name (String), age (int), and GPA (double), and then prints the information back to them.

3. Operators and Type Casting

This section covers the symbols and techniques used for calculations and data conversion.

- **Key Concepts:**
  - Arithmetic Operators: `+`, `-`, `*`, `/`, `%` (modulo)
  - Order of Operations (PEMDAS)
  - Implicit vs. Explicit Type Casting
  - Integer division vs. Floating-point division

- Increment/Decrement operators: `++`, `--` (and the difference between `x++` and `++x`)
- Compound Assignment Operators: `+=`, `-=`, `*=`, `/=`
- The `final` keyword for creating constants

- **Guiding Questions:**
  - What is the result of `5 / 2` versus `5.0 / 2` in Java? Why are they different?
  - What is the purpose of the modulo operator (`%`)? Give a practical example of its use.
  - When is explicit type casting necessary? Provide an example. (`double average = (double) total / count;`)
  - Why is it good practice to use constants (`final`) instead of "magic numbers" in your code?

- **Code to Review:**
  - Write expressions that combine different arithmetic operators and use parentheses to control the order of operations.
  - Trace the value of a variable through a series of operations using `++`, `--`, and `+=`.

4. Strings and Core Methods

This section covers the manipulation of text data using built-in `String` functionality.

- **Key Concepts:**
  - String Immutability (Strings cannot be changed; methods return a *new* string)
  - String Concatenation with the `+` operator
  - Common String methods:
    - `.length()`
    - `.charAt(index)`
    - `.indexOf(substring)`
    - `.substring(startIndex, endIndex)`
    - `.toLowerCase()` / `.toUpperCase()`
    - `.equals(otherString)` / `.equalsIgnoreCase(otherString)`
    - `.replace(old, new)`
  - Using `printf` for formatted output (`%s`, `%d`, `%.2f`)

- **Guiding Questions:**
  - What does it mean for strings to be "immutable"? If you call `.toUpperCase()` on a string, what happens to the original string variable?
  - Why must you use `.equals()` to compare the content of two strings instead of `==`?
  - How is the `endIndex` in `.substring()` used? Is the character at `endIndex` included in the result?

- **Code to Review:**
  - Given a URL string like `"http://www.pace.edu"`, write code to extract just the domain name `"pace"` using `.indexOf()` and `.substring()`.
  - Practice using `printf` to display a `double` value formatted to a specific number of decimal places.

5. Control Flow

This section covers how programs make decisions and repeat actions.

- **Key Concepts:**
  - Conditional Statements: `if`, `else if`, `else`
  - Comparison Operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
  - Logical Operators: `&&` (AND), `||` (OR), `!` (NOT)
  - `switch` Statement: `case`, `break`, `default`
  - Loops:
    - `for` loop (when the number of iterations is known)
    - `while` loop (when the loop continues as long as a condition is true)
    - `do-while` loop (guaranteed to run at least once)
    - `foreach` loop (for iterating over all elements of an array/collection)
- **Guiding Questions:**
  - What is the difference between a series of `if` statements and an `if-else if-else` chain? When would you use each?
  - Explain the difference between `&&` and `||`.
  - In a `switch` statement, what is the purpose of the `break` keyword? What happens if you forget it?
  - When is a `while` loop a better choice than a `for` loop?
- **Code to Review:**
  - Write a program that determines if a user-provided number is positive, negative, or zero.
  - Write a `for` loop that prints all numbers from 1 to 100 that are divisible by 3.
  - Write a `while` loop for input validation (e.g., keep asking the user for a password until they enter one that is at least 8 characters long).

6. Arrays

This section covers how to store and manage collections of data.

- **Key Concepts:**
  - Array Declaration and Initialization
  - Zero-based indexing (the first element is at index 0)
  - Fixed length (`.length` property)
  - Accessing and modifying elements using an index (e.g., `myArray[i]`)
  - Iterating through arrays using a `for` loop or `foreach` loop
  - `Arrays.toString()` for printing an array's contents
  - 2D Arrays for representing grids or tables
- **Guiding Questions:**
  - What happens if you try to access an array element at an index that is out of bounds?
  - What is the difference between the `.length()` method of a `String` and the `.length` property of an array?

    - How would you declare a 2D array to represent a 3x3 tic-tac-toe board? How would you access the center square?

- **Code to Review:**

    - Write code to find the largest number in an array of integers.
    - Write code to calculate the average of all values in an array of doubles.

7. Methods

This section covers how to create reusable blocks of code.

- **Key Concepts:**

    - Method Definition (return type, name, parameters)
    - `void` methods (perform an action, but do not return a value)
    - Methods with a return type (perform a calculation and return a value using the `return` keyword)
    - Parameters (passing data into a method)
    - Method Overloading (multiple methods with the same name but different parameters)
    - Variable Scope (where a variable is accessible)

- **Guiding Questions:**

    - What is the difference between a method's *parameter* and an *argument*?
    - When you pass a primitive variable (like an `int`) to a method, can the method change the original variable's value? Why or why not?
    - Explain what method overloading is and why it is useful.

- **Code to Review:**

    - Write a `void` method called `printGreeting` that takes a `String` name as a parameter and prints a personalized greeting.
    - Write a method that returns a value, such as `calculateArea`, which takes two `double` parameters (length and width) and returns their product.

8. Introduction to Object-Oriented Programming (OOP)

This section covers the fundamental concepts of the OOP paradigm.

- **Key Concepts:**

    - Class (a blueprint or template)
    - Object (an instance of a class)
    - Properties / Fields / Instance Variables (data a class holds)
    - Methods (behaviors an object can perform)
    - Constructor (a special method for creating and initializing objects)
    - The `new` keyword to create an object
    - The `this` keyword to refer to the current object
    - Instance Methods vs. Class (`static`) Methods

- **Guiding Questions:**
  - Explain the "cookie cutter and cookie" analogy for classes and objects.
  - What are the two main purposes of a constructor? How can you identify a constructor in a class definition?
  - What is the difference between an instance method and a `static` method? When would you use `static`?
- **Code to Review:**
  - Create a simple `Student` class with properties for `name` and `gpa`.
  - Add a constructor to the `Student` class to initialize these properties.
  - Add an instance method like `displayInfo()` that prints the student's details.
  - In the `main` method, create two different `Student` objects.

9. The Four Pillars of OOP

This section details the core principles that define object-oriented programming.

- **Key Concepts:**
  - **Encapsulation:** Bundling data and methods together, and controlling access using modifiers.
    - Access Modifiers: `public`, `private`, `protected`
    - Getters and Setters
  - **Inheritance:** Creating a new class (child/subclass) from an existing class (parent/base class).
    - `extends` keyword
    - `super()` keyword to call the parent's constructor
  - **Abstraction:** Hiding complex implementation details and showing only essential features.
    - `abstract` classes and methods
  - **Polymorphism:** The ability of an object to take on many forms.
    - Method Overriding (`@Override`): A child class provides a specific implementation for a method from its parent.
- **Guiding Questions:**
  - Why is it good practice to make class fields `private` and provide `public` getters and setters?
  - What is the "is-a" relationship in inheritance? (e.g., a `Dog` is-an `Animal`).
  - What is the difference between an `abstract` class and a regular (concrete) class? Can you create an object from an abstract class?
  - What is the difference between Method Overloading and Method Overriding?
- **Code to Review:**
  - Refactor the `Student` class to have `private` fields and `public` getters/setters.
  - Create a `Vehicle` parent class and a `Car` child class that `extends Vehicle`. Practice using `super()` in the `Car` constructor.

10. Data & Organization

This final section covers how Java organizes code and bridges the gap between primitive types and objects.

- **Key Concepts:**
  - **POJO (Plain Old Java Object):** A simple class used primarily to hold data (private fields, public getters/setters).
  - **Wrapper Classes:** Provide an object representation for primitive types (e.g., `Integer` for `int`, `Double` for `double`).
  - **Packages:** A way to organize related classes (like folders).
  - The `import` statement to use classes from other packages.
  - The `java.lang` package is imported automatically.
- **Guiding Questions:**
  - Why are Wrapper classes necessary? (Hint: Think about data structures like `ArrayList` that can only hold objects).
  - What is the special significance of the `java.lang` package? Name two classes that are in it.
  - What is the naming convention for packages?
- **Code to Review:**
  - Look at examples of Wrapper classes, like `Integer myAge = 25;`. What can this `Integer` object do that a primitive `int` cannot (e.g., be `null`)?
  - Review the syntax for importing classes, like `import java.util.Scanner;` or `import java.util.Random;`.