CMSC 131: Chapter 31
Final Review: What you learned this semester

The “Big Picture”

Object Oriented Programming: In this course we began an introduction to programming
from an object-oriented approach.

- Java was the primary vehicle we used for programming.
- This will continue in CMSC 132 where you will learn more of the finer points of object-orientated design.

Major Topics:

Basic Elements: types, variables, constants, operators, I/O
Control Flow: if-else, switch, while, do-while, for
Classes:
  • object instances and references
  • instance and class data
  • methods
Arrays: 1-dim, multi-dim, ragged arrays, programming
Inheritance: overriding, late-binding, interfaces

Basic Elements (of Java)

Basic Elements:

- Basic file structure: File name, class structure, comments
- Identifiers: Naming rules and naming conventions.
- Primitive types:
  • boolean
  • Integral types: byte, (char), short, int, long
  • Floating point types: float, double
- Strings: and common String operators/methods.
- Specifying constants: true, 'x', 12, 3124L, 3.14159e-10
- Operators: Their meaning and precedence rules,

Possible Questions:

- Are these valid identifier names? $_Pasta_Fazool_22
- Operator precedence: x * 2 + y --
- Is a cast needed? float y = 2.0;
Control Flow

Control flow:
- Conditionals: if-else, switch
- Loops: while, do-while, for
- Jumps: break and continue

Possible Questions:
- Trace the loop: for (int i = 4; i < 10; i += 3) { ... }
- Combining loops and conditionals: Is a string a palindrome?
- Combining loops with arrays: Insert an item into a sorted array.
- Nested conditionals: Given x, y, and z, return the smallest.
- Nested loops: Selection, Insertion, Bubble Sort.

Avoid Pitfalls:
- Don’t forget breaks in switch statements.
- Be sure you understand the problem before coding. (If not, ask)
- Test on a example before and after coding.

Classes

Class Objects and References:
- Creating object instances with new
- The null reference
- Java’s class library
- The heap and garbage collection

Basic Class Concepts:
- Instance (non-static) and class (static) variables
- Access modifiers (public, private, protected, default (package))
- Constructors (initializers)

Methods:
- Static and non-static
- Arguments, call-return, and local variables
- Return and memory leaks
Classes

Possible Questions:

- **Draw a memory map**

- **You be the compiler:**
  - We declare methods and create some objects, and
  - ... ask you which overloaded method is called, or
  - ... ask you whether access is valid/invalid

- **Class design:**
  - We specify the objects and functionality
  - You design the class: instance variables, constructors, methods

Avoid Pitfalls:

- **Call the right method**: Determine which overloaded function to call, then check its accessibility.
- **Read your old programs**: Eclipse fills in many things for you automatically. Be sure you can do it on your own.
- **Deep/Shallow copy**: For a deep copy, create new object instances.

**Memory Map Example**

```java
public class Person {
    private String name;
    private int age;

    public Person(String n, int a) {
        name = n;
        age = a;
    }

    static void f(String b) {
        int x = b.length();
        /* STOP HERE */
    }

    public static void main(String[] args) {
        String[] n = {"Bob", "Alice"};
        Person p = new Person(n[1], 43);
        f(n[0]);
    }
}
```
Arrays

Arrays:

- Creating, indexing
- Arrays of primitives and object references
- 2-dimensional arrays and ragged arrays (array of arrays)

Possible Questions:

- Array manipulation:
  - Inserting/Removal/Searching in partially filled arrays
  - Appending/Merging arrays
  - Shifting, rotating, permuting elements in an array

- 2-dim array manipulation:
  - Appending arrays
  - Transposing arrays (inverting rows and columns)

Avoid Pitfalls:

- Allocation: Remember to explicitly allocate array/element storage
- Simplify: Break complex operations into smaller/simpler pieces
- Inside-Out: Design nested loops from the inside-out
Inheritance

Inheritance:

- Terminology: base class (superclass), derived class (subclass), super (parent class)
- Method overriding (vs. overloading), late-binding, and final
- Abstract methods and abstract classes (derived class implements) and polymorphism
- up- and down-casting, get Class and instanceof
- Multiple inheritance and interfaces

Possible Questions:

- Be the compiler: We give code. Which method is called?
- Design: We give the specs, you design the classes and methods.

Avoid Pitfalls:

- Overriding only occurs when prototypes are identical. Start at the declared class, and go to the derived class if overriding applies.

Miscellaneous

Miscellaneous Topics:

- Javadoc: @tags and their usage: @author, @param, @returns, ...
- Java Class Library: Math, DecimalFormat, ArrayList, ...
- Packages:
  - Organizes Java files into groups (directories)
  - Packages may be subdivided into subpackages
  - You have access to names in your own package or packages you import.
- Exceptions:
  - try-catch blocks and exception propagation

Possible Questions:

- Know common methods for String and ArrayList. For others know the basic concepts (Stack: push, pop, peek) but not details.
- Package: What is accessible with/without importing?
- Exception: Design_TRACE try-catch blocks,
What's Coming Up?

**CMSC 132**: You will learn more about object oriented programming in greater depth, still in Java.
- More on software testing (JUnit) and debugging
- I/O and exceptions
- Recursive functions
- Linked data structures
- Networking

**CMSC 212**: Low-level programming concepts
- C programming
- Memory allocation and deallocation
- Internal representations of arrays and data structures

**CMSC 250, 351**: Discrete math and algorithm design

**CMSC 311**: Computer organization (“what's inside”)

**CMSC 330**: Programming languages