Dylan Johnson

Final Part 2

CSC 330

Summer 2023 Part 2

1. What are the three pillars of object-oriented programming?
   1. Encapsulation
      1. Which is the idea of “hiding” information. Essentially you break apart code in such a way that it does not show all the details, but the main file still explains what the program is doing.
   2. Inheritance
      1. This is where classes come into play. Superclass and subclasses that form an is-a relationship. For example, you could have a tree super class that allows for palm trees to fall into a subclass category and inherit all important details about trees without having to add it all again.
   3. Polymorphism
      1. This is similar to inheritance but also very different. Polymorphism is when you have a base superclass, say Shape, that has a predefined function to find the area of a shape. However, not every shape calculates area the same way. So then when you create subclasses for different shapes, say a circle, it inherits the need for to calculate the area, which the developer can then define on a case-by-case basis if needed.
2. What is the purpose of a garbage collector?
   1. Garbage collection was created to help free up memory in computing when memory was not as easily accessible. Essentially the program itself would free up memory space by removing references to parts that were no longer reachable by the program.
3. Define and identify a modular programming language.
   1. Modular programming is essentially thinking as big picture as you can. Do not write a very specific program, write functions that can be used in a variety of cases, and then build that into your code. That way if you ever needed something like that in a later project you can pull that chunk of code and "plug it in" to another program with minimal effort. Essentially, it's like building libraries of code. The point being is it makes code easier to maintain, and easier to reuse.
4. Write an example of a loop and its recursive equivalent.
   1. Code screenshot:
      1. A computer screen shot of a program

         Description automatically generated
   2. Copy/Paste code:

public class CalculateFactorial {

public static void main(String[] args) {

int number = 5;

int factorial = 1;

for (int i = 1; i <= number; i++) {

factorial \*= i;

}

System.out.println("Factorial with a loop:");

System.out.println("Factorial of " + number + " is: " + factorial);

number = 6;

factorial = calculateFactorial(number);

System.out.println("\n\nFactorial using recursion:");

System.out.println("Factorial of " + number + " is: " + factorial);

}

public static int calculateFactorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* calculateFactorial(n - 1);

}

}

}

1. List pros and cons of the GOTO statement.
   1. While I really liked GOTO when I first started writing batch programs, I grew to have a distaste for them pretty quickly.
   2. Pros:
      1. Very simple to use and pretty self-explanatory.
      2. Offers access to any chunk of code at any point
   3. Cons:
      1. Any line changes to a program and every reference to a line below from GOTO is broken.
         1. Which makes maintaining the code rather difficult/inconvenient.
      2. When going back through the code you end up jumping all over to find what you are looking for.
         1. Readability of the code is, again, difficult/inconvenient.
   4. There’s really just no good long term use case for this tool unless you can make a program that never breaks.
2. Write an EBNF for fully parenthesized infix mathematical expressions, for example, (1 + 2).
   1. [This](https://bnfplayground.pauliankline.com/?bnf=%3Cexpression%3E%20%3A%3A%3D%20%22(%22%20%3Cexpression%3E%20%22%20%22%20%3Coperator%3E%20%22%20%22%20%3Cexpression%3E%20%22)%22%20%7C%20%3Cnumber%3E%0A%3Coperator%3E%20%3A%3A%3D%20%22%2B%22%20%7C%20%22-%22%20%7C%20%22*%22%20%7C%20%22%2F%22%0A%3Cnumber%3E%20%3A%3A%3D%20%3Cdigit%3E%20%3Cdigit%3E*%0A%3Cdigit%3E%20%3A%3A%3D%20%220%22%20%7C%20%221%22%20%7C%20%222%22%20%7C%20%223%22%20%7C%20%224%22%20%7C%20%225%22%20%7C%20%226%22%20%7C%20%227%22%20%7C%20%228%22%20%7C%20%229%22&name=) is a link to the below screenshot.
   2. A screen shot of a computer

      Description automatically generated
3. Describe the difference between lexical analysis and parsing in a compiler.
   1. Lexical analysis seems a lot like modular programming. Basically, modular programming is where you take the individual pieces of the code and work piece by piece. When putting that into lexical analysis terms, you are breaking code down into these "tokens" which are basically words or sentences describing what the code is doing. Whereas parsing is where you take all those pieces at once and fit it all together to get the program to run. Another way to think of parsing is the main file of a program. Lexical analysis is all the different methods, classes, and whatever else being used. Then Parsing is the main file pulling it all together and calling what's needed and when.
4. Describe the difference between weak and strong typing within a programming language.
   1. Weak typing is where the program makes curtain assumptions for you allowing for easier to interact with code. The drawback here is the possibility of errors or the handling of information in ways you did not intend. Whereas strong typing takes those assumptions away and will throw errors forcing the developer to explicitly make those changes that were assumed.
5. What is tail recursion, and why is it important to optimize tail recursion when it is used?
   1. Tail recursion, like recursion, is the idea of calling a function from within the function. The difference being the recursive call to itself is the last part of the function. It is important to optimize this because, if not properly used, it can completely kill a program, especially in larger scale use cases.
6. Describe syntactic sugar and give an example of code that is sugar for another choice of syntax.
   1. Syntactical sugar is the idea of completing a task in a shorter amount of code without losing any functionality.
   2. Here’s a basic java example of this:
      1. A computer screen shot of numbers and letters

         Description automatically generated
      2. In this example we replace an excess variable that we are counting down to and only iterating through what is there (visually). Both examples are doing the same thing, one is more “behind the scenes”/shorthand while the other is easier to understand for a beginner.