Don Petersen

S3 Discussion Questions

1) What are the "blocks?" What is nesting? What kinds of things may be nested? How deeply should you nest at most? Why?

The textbook states that “Blocks are a collection of statements enclosed in braces.” These blocks are used during if statements that want have multiple statements execute when the condition of the if statement is met. Nesting is when you put conditional statement(s) inside another conditional statement. An example of nesting is a nested if statement:

In this example the program checks the inside statement if(num2 > num3) and then, if it passes, checks the outer statement if(num1 > num2). Many things can be nested in C#, I will probably miss something, but if, do, while statements and loops can all be nested, as well as, switch statements(although, I think this is not practiced). With nested statements it is good practice to consider the readability of the code, for this reason it is best to only nest as deeply as is necessary to achieve the objectives of the program and if you can find a way to avoid nesting deeper than you should choose that option.

Lewis, J. (2007). C# Software Solutions: Foundations of Program Design. Boston, MA: Pearson Education, Inc.

2) What repetition or iteration control structures are included in C#? How do they perform? In what ways are they similar/different from one another?

Iteration control structures are more commonly called loops, the types of loops included in C# are: the while loop, the do loop, the for loop, and the foreach loop. The while loop checks a condition and then continues to execute a block of code as long as the Boolean used evaluates as true, looping through the block until the Boolean value evaluates as false. The while loop is distinguished from other loops because if the Boolean expression is false, the statements inside the block of code are skipped. The do loop is similar to the while loop but instead of checking the condition before the loop, it checks its condition at the end of the loop. This is to guarantee that the loop execute at least once. The for loop is used when you know exactly how many times you want the statements in the loop to execute. The syntax of the for loop includes initialization, a Boolean expression, and a counter. The for loop will run until the Boolean expression evaluates as false, similar to the while loop, when a false is evaluated control is then given to the first statement following the for block. The foreach loop is used to iterate through the items in a list, the list can either be an array or collection.

Lewis, J. (2007). C# Software Solutions: Foundations of Program Design. Boston, MA: Pearson Education, Inc.

3) How are overloaded methods distinguished from one another by the compiler? Why do we use overloaded methods?

Overloaded methods have different parameters passed to them but have the same name. The complier distinguishes the different methods by the types of the parameters being passed. We use overloaded methods when we want the same method used in different ways. The return type for overloaded methods must be the same so we are just passing different parameters into the same method. For example: public int Sum(int num1, int num2) {return num1+num2;} has the same name and return type as public int Sum(int num1, int num2, int num3) {return num1+num2+num3;}. In this example the Sum method is overloaded. The reason that programmers use overloaded methods is that it gives a greater flexibility to the method and allows you to perform similar methods on different types of data without having to write a whole new method.

Lewis, J. (2007). C# Software Solutions: Foundations of Program Design. Boston, MA: Pearson Education, Inc.

4) What does this refer to? What function(s) does this perform for the programmer? Some programmers dislike using this. Do you? Why/why not?

The this keyword refers to the object that invokes the keyword. For example: this.name = name; allows the object name to refer to itself. This is mostly used in constructors to make it so the programmer doesn’t have to come up with another instance variable name that means the same as the parameters. For example: public Account (string name) {owner = name;} the programmer had to create the instance variable owner for the constructor when instead the keyword this allows the programmer to write the constructor like this: public Account(string name) {this.name = name;}. Personally, I have never used the this keyword because I am just now being introduced to it, but I imagine I will use it from now on, especially in constructors, because I dislike having to come up with multiple variable names from the same thing. It seems that the this variable will make the program easier to read and understand.

Lewis, J. (2007). C# Software Solutions: Foundations of Program Design. Boston, MA: Pearson Education, Inc.

5) What are the differences between classes and interfaces? Will you design your own interfaces in your programs? Why/why not?

I think the main difference between classes and interfaces is that classes create methods with specific statements in the body of the methods and interfaces hold names and parameters of methods that can be used elsewhere, and is considered an abstract method. Reading the textbook material is my first exposure to interfaces, so, I imagine that it would take some deliberate use of them to get me to remember that they exist. I could very easily see myself just using classes since all of the programming languages I have used so far use classes but I don’t think they deal with interfaces and it seems that whatever can be accomplished with interfaces can also be accomplished with classes.

Lewis, J. (2007). C# Software Solutions: Foundations of Program Design. Boston, MA: Pearson Education, Inc.