COP 3330 – Object Oriented Programming in Java Introduction

Necessary Background Information

Imperative Programming vs. Object Oriented Programming

Languages such as C and Python, as taught in introductory classes, are essentially imperative languages. These languages have variables of various types, decision and looping constructs as well as regular functions. They support sequential execution of commands given some entry point (in C it's main but in Python the programmer selects this). In an imperative language, data and functions are independent. All non-global data that a function needs to perform its task must be explicitly passed to the function.

In an object-oriented language, the programmer is not only allowed to define her own data type, but also associate functions with that data type, binding those functions specifically to objects (variables) of that data type only. In practice, this means that the programmer **does not explicitly** pass a variable of the user-defined data type into any of the associated functions. Rather, implicitly, each of these associated functions is assumed to have access to the components of the variable of the user-defined data type. Philosophically, this creates a greater abstraction for the programmer: not only the ability to create a new type, but also to specify specific behaviors for that type. In addition, it allows for data encapsulation or data abstraction: other programmers can use the new data type, without knowing **HOW** the data is stored. This is advantageous because a programmer can benefit from the power of a newly defined data type and associated functions (called a class) without spending as much time to understand the nitty gritty details of how it works. For all new students to object oriented programming, it's likely this paragraph doesn't make a whole lot of sense. I urge you to come back to it about 8 or 9 weeks into the course and I promise you it will take on new meaning.

In an object-oriented language, we'll put more focus on what data and functions (called methods for object-oriented languages) are accessible where (called visibility). We'll also pay more attention to overall application design and think more about not only how to develop code for a class, but also what functionality we would like to provide an outside programmer who will use the class (but not develop it).

Java - Strict Type Rules

Java has rules that, for the most part, are strictly followed. It's a very strictly typed language, compared to other languages. This means that the programmer has to be aware of and properly define the type of each variable and in expressions they must take extra care to make sure that statements are consistent with respect to the types of expressions that are part of the statement. In practice, this means that statements that aren't strictly following type rules will not compile in Java, even though in other languages equivalent statements would compile. While this sounds onerous to have more strict rules, it turns out to be helpful and often save the programmer time. In C, many non-sensical statements compile even when the types of the expressions in the statement don't actually make sense. As a consequence, quite a few C programs compile, but then crash or behave unpredictably. In Java, these equivalent statements wouldn't compile and the compiler gives better error messages explaining what syntax rule has been broken. Then, the programmer is alerted to the area with the error and can usually figure out what they meant to write and fix the code.

Compiled vs. Interpreted Languages

C is a compiled language while Java and Python are interpreted languages. In a compiled language, when you compile your code successfully, the compiler produces an actual executable file (think hello.exe). This executable file (complete nonsense to a human) can directly be run by the computer.

In an interpreted language, the compiler does not produce an executable. Instead, to run an interpreted language, you need an interpreter, which, as you run a program, goes back and forth between converting code the machine doesn't understand to code it can execute. For Java, the JRE (Java Run Time Environment) is what allows for the interpretation of Java Byte Code. When a Java program is compiled successfully, it produces a class file. So successful compilation of Hello.java will create the file Hello.class, which is not an executable file. (You can't directly run it like you can run hello.exe.) To compile on the command line in Java do this:

```
>javac Hello.java
```

To interpret and run a Java program do this:

```
>java Hello
```

Thus, the program that is executing is "java", and its job is to interpret and execute the commands stored in the file Hello.class, which stores Java Byte Code.

Java was invented in 1995 with the primary purpose of being more portable, which is why it's interpreted. The byte code is uniform and can be interpreted on any machine that has the JRE.

For our purposes, this won't create a huge difference. Interpreted languages tend to run a bit more slowly when they execute than compiled ones (very computationally intensive real time code tends to be written in executable languages and not interpreted ones), but with the speed of today's computers, the regular introductory student will not notice any difference.

Development Environment

There are many development environments for Java, but no fancy IDE (Integrated Development Environment) is necessary. Regardless of the IDE, in order to write Java programs, one must download the JDK (Java Development Kit). Here is the site to do so (as of January 12, 2026):

https://www.oracle.com/java/technologies/downloads/

Note that this is different than the JRE (Java Runtime Environment).

Once you do this download, you'll have programs with the names "javac" (the Java compiler) and "java" (Java interpreter). If you add the location of these executable programs to your computer's path, then you'll be able execute from the command line as shown above without typing in the full path of where the programs java and javac are located.

Here is a list of some popular Java IDEs: Eclipse, NetBeans, BlueJ, IntelliJ IDEA, VS Code, JCreator, DrJava, jGRASP.

For the purposes of this class, you may use any IDE that you see fit. Alternatively, you can just download the JDK and use Notepad++ or any other suitable text editor. The teaching assistants will use JDK 25 for grading purposes.

Please learn how to download the compiler and either compile/interpret on the command line or through an IDE. Do NOT USE an online compiler for this class.

First Java Program: Hello

Hello.java

Let's take a look at a Java program that prints out a greeting:

```
// Arup Guha
// 7/7 03
// My first java program

public class Hello {

    // Prints out a greeting.
    public static void main(String[] args) {

        System.out.println("Hello (Insert Your Name Here)!");
    }
}
```

Comments

The first three lines are comments. These are ignored by the compiler. The style of comments are the same as C comments. For a single line comment, use two forward slashes (//). For a multi-line comment, start the comment with forward slash-star (/*) and end the comment with star-forward slash (*/). Since the audience for this class has programmed before, no further details need to be included. Everything you learned about commenting in your first programming class applies here. My class examples illustrate the amount I would like for you to comment. Every program should have a header comment with your name, the date and a description of the program. Internal comments should be for each logical piece of code.

Required Components of All Java Programs

All Java code must reside in classes. For the first few weeks, our programs will consist of a single Java class stored in a single Java file. As the class progresses, our programs may have multiple classes in either one file or multiple files. The rules for these will be discussed as necessary. For these notes, we'll only describe how to write a single Java Program in a single class (with a single method, main).

Each Java file must have a name of the form CLASSNAME.java, where CLASSNAME is the name of the public class defined in the file. Thus, the program shown above **MUST BE** stored in a file called Hello.java. This isn't true in many other languages.

Within that file, since we are going to make the class public, we must define the class as follows:

```
public class Hello {

    // Class Contents go here.
}
```

For now, the entirety of your programs will be contained inside the curly braces shown above.

Main Method

In Java the term that is called "functions" for other languages is called "methods." The starting point for execution in a Java program is the main method, which must be defined as follows:

```
    public static void main(String[] args) {

        // Code for main method here.
    }
```

For now, just copy this. But here's a quick breakdown of what each term means:

public – this is the visibility modifier. We will explicitly state, for most methods, whether they are public or private, and this visibility modifier goes first. The visibility modifier indicates from which classes the method can be called.

static – all methods in Java are either static methods or instance methods. If a method is static, you must write the word static here in the method. If a method is an instance method, you simply omit the word static. In short, static means "belongs to the class" and instance means "belongs to the object." This will make more sense later, once we learn how to define our own objects.

void – this is the return type. The main method does not return anything. The return types for Java are similar to other languages. A function can either return void or return any valid type in the language, including user defined types (which are really classes).

main – this you should be familiar with; it's the name of the method that gets automatically executed.

We'll discuss String[] args a bit later.

Method for Printing

One method provided to print output to the screen is println, which resides in System.out. Thus, to print out a string, just do:

```
System.out.println(s);
```

Where s is the string to be printed. At first, we'll just print string literals. A string literal in Java is denoted by two double quotes. This prints the string given to the method and then advances the cursor to the new line. In C, the cursor doesn't get advanced to the new line and in Python you can use single quotes to denote a string literal, but not in Java.

Program Interpretation
When we interpret (from now on I'll just use the term run) our program, we get the following output:

```
Hello (Insert Your Name Here)!
```

and the cursor will advance to the next line before the program stops.

Printing Variations and Escape Sequences
There is a second method for printing, which is the print function. This is almost the same as the println function, except it doesn't advance the cursor to the next line.

Essentially, the two statements below are identical:

```
System.out.println(s);
System.out.print(s+"\n");
```

Note that the plus sign operator, when used with strings is string concatenation. More on that later.

The escape sequences for Java are the same as the escape sequences for C. I'll typically use: '\n' (newline), '\t' (tab), '\"' (double quote) and "\\" (backslash). We'll come back and look at printing with String concatenation in a bit.

Now, we can take a look at the second sample program which utilizes both print and println as well as three of the escape sequences:

```
// Arup Guha
// 7/7/03
// My second java program: Shows the difference between print and println.

public class Hello2 {

    // Prints out a meaningless message.
    public static void main(String[] args) {

        // The text describes what I am trying to teach!
        System.out.print("Hello again. ");
        System.out.println("Notice the difference between print and println.");
        System.out.println("This text is on the line below.");
        System.out.print("But I can get to the next line this way\nalso.");
        System.out.println(" Or I can make a tab\t also.");
        System.out.println("Sally says, \"she sells sea shells by the sea
shore.\"");
      }
}
```

Essentially, after prints, the output continues on the same line, and each of the escape sequences prints the desired output instead of what's literally in the double quotes.

String Concatenation

The plus sign in Java is overloaded. What this means is that it has two different meanings, depending on context. Specifically, if BOTH operands to the plus sign are numeric, then the plus sign means addition. But, if either operand to a plus sign is a string, then string concatenation is performed. (Concatenation is the act of putting two strings next to each other to form one larger string.) Expressions are evaluated left to right of course. Here is an example to illustrate string concatenation in an expression with both strings and integers:

```
// Arup Guha
// 1/12/2026
// My third java Program to show string concatenation.

public class Hello3 {

    public static void main(String[] args) {

        // Check out the output of each version!
        System.out.println("3 + 7 = "+(3+7));
        System.out.println("3 + 7 = "+3+7);

        // Other versions.
        System.out.println("3"+" + 7"+" = "+(3+7));
        System.out.println(3+" + 7"+" = "+(3+7));
        System.out.println("3 + "+7+" = "+(3+7));
    }
}
```

The only issue with any of these five lines that causes an "error" so to speak is the second example where there are no parentheses around 3 + 7. As we evaluate left to right in this example, we first encounter this expression:

```
"3 + 7 = "+3
```

Because the first item being added is a string, the plus sign is designated as string concatenation. Java automatically converts the number 3 to the string "3" and produces the following result for the expression:

```
"3 + 7 = 3"
```

Next, this string is then concatenated to 7:

```
"3 + 7 = 3"+7
```

and naturally, this evaluates to:

```
"3 + 7 = 37"
```

In the last few examples we see that no matter if the string is first or second, Java will still do string concatenation. It only does addition if BOTH operators to the plus sign are numeric (as occurs when we put parentheses to force the evaluation first of 3 + 7.)

Command Line Arguments

Going back to the definition of main we see String[] args. So what does this mean?

String[] args – we won't use this in our class, but these are for command line arguments. If you were to run the program on the command line as previously shown, but you wanted to provide actual parameters to the main method, you can do that via the command line. In short, the main method takes in an array of Strings, for which the formal parameter name is args.

Consider the following program that does use command line arguments in a basic way:

```
// Arup Guha
// 1/12/2026
// My fourth java Program to show the use of command line arguments and string
concatenation.

public class Hello4 {

    // Prints out a greeting to the person in the command line argument.
    public static void main(String[] args) {

        // The text describes what I am trying to teach!
        System.out.println("Hello "+args[0]+" "+args[1]);
    }
}
```

For example, if we did:

```
>java Hello4 Alex Cartwright
```

The output produced would be

```
Hello Alex Cartwright
```

Essentially, args is an array, and args[0] is the first string on the command line after java Hello4 and args[1] is the second string on the command line after java Hello4. In our print,