An Introduction to Competitive Programming and Java

CS Team Captains

September 17th, 2020

1 Introduction

Welcome to NCSSM's CS Team, one of the best clubs on campus! Our main focus as a club is on programming competitions. We'll meet weekly throughout the year and learn about algorithms and data structures together. We hope to prep everyone for the United States of America Computing Olympiad (USACO), but also participate in a number of other competitions throughout the year. Our sponsor is the wonderful Dr. John Morrison who teaches a multitude of programming classes here at NCSSM. Your captains for the year will be:

- Siddhant Dubey
- Sreyas Adiraju
- Jonathan Xu
- Evan Zhang

2 Schedule

We meet every week on Thursday from 8-9 PM. Every meeting will consist of a lecture delivered by a few of the captains on Computer Science topics. Once in a while, we'll also have in-school contests for members of the club. If any changes occur in the schedule, you'll hear about it in the Facebook group.

If you ever miss a meeting, you can find the materials for it on our website: `https://ncssmcsclub.github.io/`.

3 Contests

3.1 USACO

USACO, the flagship programming contest of the United States of America, is an online contest that occurs once a month starting in December and ending with the US Open in March. The contest lasts for around four days each

month starting on a Friday and ending on a Monday evening. There are four divisions: Bronze, Silver, Gold, and Platinum that get progressively harder in difficulty. When you create your account, you start off in the Bronze division and can promote to higher divisions by meeting the cutoff score for each contest. The best programmers of the season are selected for the USA's IOI (International Olympiad of Informatics) training camp. Our material is geared towards preparing for the USACO so you should definitely try out at least one contest.

3.2 Other Contests

- **Virginia Tech's High School Programming Contest**: This is an online contest typically held in December. There are 13 difficult problems that you will try to solve in 5 hours. There's no limit on the number of teams that can participate from a school, so we don't hold tryouts. The max size for a team is 3 people. It is unknown if this contest will happen this year.
- **UVA's HSPC**: This is an in-person contest at the University of Virginia that occurs in the spring. We can take two teams of four and will hold tryouts sometime in the second semester.
- **Various Online Judges**: These are websites like Codeforces and AtCoder. They hold contests weekly and have an ELO system to rank you in comparison to other competitive programmers. They're pretty fun to participate in and don't last that long, so participate when you can, they're good practice.

4 What is Competitive Programming?

Competitive programming is an art form. It's creative problem-solving at its finest, a combination of hard analytical thinking and creativity. Competitive programmers use their knowledge of algorithms and data structures and logical reasoning skills to solve challenging algorithmic problems in a limited time frame.

The preferred language of Competitive Programmers is C++ with a large minority choosing Java. This is due to their relative run-time efficiency compared to other languages like Python. Of course, you should use whatever you feel comfortable with, but these lectures will be done in C++. The lecture notes will also contain Java code.

5 Introduction to Java

Java is a largely object oriented programming language that's popular in applications today. It's syntax is fairly similar to C++ and shares the same concepts of classes and functions. Although not as fast as C++, it's known to be more readable and is standardized across all operating systems.

6 Format of a Java Program

Java programs begin executing from the main() method. The program can have various helper methods and global variables (you'll end up making a lot of arrays this way). Using global variables lets your code run quicker and it is good practice. In general, global variables are the easiest to work with and pass through different functions, instead of passing a local variable in the main function. Every Java program should start like this

```
public class ClassName {
   /* The slash and star indicate a multi line comment, which is merely
       for marking up code
    * This is the main method
    * the keywords public, static, and void will be explained later
        along with String[] args
    */
   public static void main(String[] args) {
      //executable code, the two slashes mean this line is a comment
          and will be ignored at runtime
   }

}
```

6.1 Your first Java Program!

```
public class HelloWorld {

    //the program will execute this main function, which contains our
        output
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }

}
```

Let's walk through what this code is doing. Every Java program needs a class as the entry point, which we'll talk about more later. The class name after **public class** must match the file name of the Java program for the program to begin executing. The program will then see the **main(String[] args)** function and begin executing the code there line by line. We'll get into libraries and classes later, but essentially what **System.out.println()** does is use a built in Java library to print the string you enter to the console/terminal. A string in Java is represented between double quotes, so the string in this case is "Hello World!". At the end of every line of code, we need to add a semi-colon, otherwise the compiler won't know that this line of code is over.

7 Variables, Data Types, and Operators

The data types in Java are very similar to ones from other programming languages, so if you've coded before this should all seem familiar. If not, no problem, you're about to learn a lot. This section will cover primitive data types like **ints, chars, booleans, floats, and doubles** and the **String**. We'll also be covering the all powerful arrays.

7.1 Numerical Data Types

7.1.1 The Integer

The **int** data type requires 4 bytes of memory space and goes from -2147483648 to 2147483647.

7.1.2 The Float

The **float** data type is used to store single precision decimal values. This has pretty much no use in Java, just use a double.

7.1.3 The Double

The **double** data type is used to store double precision decimal values.

7.1.4 The Long

The **long** (long long int) data type has a larger capacity than the int data type and can go from -2^{63} to $2^{63}-1$. Use this when you think you're going to be dealing with large numbers that an int can't hold.

7.1.5 The Boolean

The **boolean** data type can contain one of two values: true or false. You can also use 1 (true) and 0 (false). These are very important when you're writing programs that require **conditional logic**.

7.1.6 Code example!

Remember Java is statically typed, which means that you have to declare the type of every variable you create. To declare variable in Java, firt type the data type and then the variable name. Then, to assign a value, you can optionally add a value to assign it to. Every line of code in Java ends with a semi-colon, so don't forget that. Here's an example of how you would do that with the numerical data types that you were just exposed to.

```
public class Test {
```

```
  public static void main(String[] args) {
      int a; //declaring variables without assignment
     double b;
     boolean c;

     int x=10; //declaring variables with assignment
     double y=1.01;
     long z = 200000000001; //the l at the end (lowercase L) indicates
         that the literal number is a long
     boolean sreyas_is_cool=true;
  }

}
```

7.2 Arithmetic and Assignment Operators

In addition to all the standard binary operators(+, -, *. /) and the simple assignment operator (=), Java also supports the modulo operator(%), compound assignment(+=, -=, *=, /=, %=), bitwise operators and unary operators (++, –, !). The modulo operator gives the remainder after division, for example: 4 % 3 = 1. In Java, dividing an integer by another integer will perform integer division only. Integer division is truncating, which means that it doesn't round and just takes the whole number that is present. For example, the line $1/2 = 0$, whereas $1.0/2 = 0.5$. In the second example, Java will interpret 1.0 as a double, and thus perform regular division. When you're working with floating points, you should expect precision errors. As for the order in which these operators work, Java does have operator precedence that closely matches PEMDAS. Just use parentheses and you'll be fine.

7.3 Relational Operators

In addition to arithmetic and assignment operators, Java also features relational operators used to perform conditional tests. These operators return booleans and include:

- **Equivalence tests: == and !=**: == is checking for equality while != checks for inequality.
- **Comparisons: <, >, <= and >=**

Additionally, Java allows for the compounding of conditions with logical operators: **logical and** (&&) and **logical or** (||) . Java uses short-circuit evaluation for these operators: if the first condition is enough, the second condition will not be evaluated.

7.4 Strings and Characters

Dealing with Strings and Characters is essential for any program, because all programs need some level of human interaction.

- The **char** (character) data type, holds a single character, for example: 'a'.
- The **string** data type is what you would use to hold any sort of textual data that is longer than one character. It's just multiple characters stringed (whoa!) together.
- It is important to remember that **chars** are in fact treated as numbers. Java uses an encoding (ASCII) to map numbers 1-255 to printable characters.

Here's a code example that shows you how to create strings and characters in your own programs.

```
public class Test {

   public static void main(String[] args) {
      char a='a';
      String word="cool";
   }

}
```

You may have noticed that single quotes are used for characters and double quotes are reserved for strings. This may shock the Python users among you, but it is a harsh reality of programming in Java.

7.5 Arrays

Arrays in Java are collections of items. Whereas Python lists can contain different data types, all the items in a Java array must be the same data type. These could be any of the primitive data types we've already discussed. There are multiple ways to initialize arrays and they're shown in the code block below. The commonality in between them is that you always need to put the data type the array contains before the name of the array.

```
//Note: This code is within a main() method.
//In the future I will omit public class and public main lines
//Array Declaration by specifying a size:
int[] arr=new int[10];
//NOTE THAT int arr[]=new int[10]; works the exact same, the placement
    of brackets is optional
//I will use the first way of declaring arrays

//You can also initialize an array with a size that is a variable:
```

```
int size = 100;
int[] arr2=new int[size];

//Array declaration by initializing elements:

int[] arr3 = {1,2,3,5}
```

When you initialize an array without specifying any elements in the array, you make an empty array where every space in the array is empty. In Java, number arrays will initialize to 0. Arrays are extremely valuable tools you'll be using a lot in your Competitive Programming journey.

7.5.1 Indexing Arrays

Oftentimes you want to be able to retrieve an element in a particular location in the array. In Java, arrays are 0 indexed, which means the first element in the array is at position 0, the second element is at position 1, and so on.

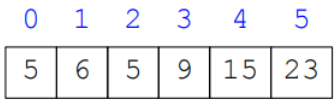

Figure 1: An Array

To output the *nth* element of an array *arr*, you would write the following line of code:

```
    System.out.println(arr[n-1]); //arr[n-1] retrieves the nth element
        of arr
```

8 Conditional Logic

Conditional statements enable you to create control flow in your programs, and are essential in most algorithimic tasks.

8.1 If/Else Statements

If-else statements are an essential tool for any programmer and they take the following form in Java

```
    //Note: This code is within a main() method.
if(/*boolean condition*/){
    //Code to execute if true
}
else{
```

```
    //Code to execute if false
}
```

The curly braces that you see in the code are used to define a block of code, establishing a scope for variables defined within it. However, if you only have one line of code to execute in a statement, you can omit the curly braces.

8.2 Else If and Nested If statements

Sometimes you just have a lot of conditions to check, and in that case you can chain them together in an **if ... else if ... else** structure.

```
int x = 3;
if(x < 2){
    x = 2;
}
else if(x == 2){
    x += 2;
}
else{
    System.out.println("SUCCESS!")
}
```

The code above checks if some integer x is less than 2, which x is not, then checks to see if it is equal to 2, and if it is neither of those two things the code outputs "success" to the terminal.

Other times, you have different things to check based on the first conditional statement. In that case you would use **nested ifs** which place one condition within another conditional statement. For example:

```
if(x <= 17 && y > 0){
    if(x <= 2 && y > 3)
        System.out.println("cool");
}
else{
    System.out.println("not cool");
}
```

Huzzah! You now know how to use conditional logic in Java and you're one step closer to being a Competitive Programmer god.

9 Loops

Loops are an essential part of any Competitive Programmer's toolbox. They allow you to execute similar code an arbitrary number of times and are invaluable in competitive programming. They have an infinitude of uses and it is of the utmost importance that you are comfortable with using loops.

9.1 For Loops

For loops are used to do something **for** a specified number of times. The for loop header consists of three parts:

- An **initialization**, often used to declare and assign an iterator, typically an **int i** or **j**.
- A **condition** used to determine whether the loop runs.
- An update step, typically used to increment the iterator.

The loop body contains the code that you want to run in the for loop. Here's an example of a for loop that is outputting the first n elements in an array.

```
for (int i = 0; i < n; i++){
    System.out.println(arr[i]);
}
```

In the for loop above our iterator is an integer i which is initialized with a value of 0. Our condition is that i must be less than n, and our update step adds 1 to i every time the for loop body executes. The body outputs the ith element of the array starting with the 0th element.

9.2 Enhanced For Loop

The **enhanced for-loop** allows you to more conveniently iterate through array-like types:

```
int[] arr = {2, 3, 4, 5, 6, 7};

for(int element : arr){
    System.out.println(element);
}
```

This type of for loop is also known as a *for each* loop on account of it going through each element of an array.

9.3 While Loops

While loops execute until the supplied **termination condition** returns **false**.

The **while** loop has a few important uses (in graph algorithms):

```
while(/*Condition*/){
    //execute code here
}

//Fake for loop
int i = 0;
```

```
while(i < n){
    System.out.println(arr[i]);
    i++;
}
```

10 Functions

A **function** in Java is a code block that is only run when it is called on. Functions have different data types that they return which include all the primitive data types we have already discussed and others like the **void** type. **void** simply means there is no value, so it is used for function types that don't return any sort of value.

```
//hello(), add(), and main() are all functions.
//Just like variables you have to specify the type of data a function
    will return.
static void hello(){
    System.out.println("hello");
}

static int add(int a, int b){
    return a + b
}

public static void main(String[] args){
    hello(); //Outputs "Hello"
    System.out.println(add(1, 2)); //prints 3
}
```

You'll often see Competitive Programmers using functions to make their code look neater as well as modularize it so that it becomes easier to test.

11 Basic Input/Output in Java

Contest problems will first and foremost require you to read some input and return an output in a file. Java will not immediately have the packages required to do this, but to be able to use these packages, we can add import statements to the top of our Java program to let the compiler know we'll need to use these packages. Here's an example program with both input and output.

```
import java.io.*; //imports the io class from java

public class Test {

   public static void main(String[] args) throws IOException{
      File file = new File("filename.in"); //new file
```

```
      BufferedReader reader = new BufferedReader(new FileReader(file));
          //initialize reader
      PrintWriter writer = new PrintWriter(new BufferedWriter(new
          FileWriter("filename.out"))); //initializing a writer

      String line=reader.readLine(); //we're using our reader to read a
          line in the file

      writer.write("This is a test"); //using a writer to output some
          text
      writer.close();
   }

}
```

Okay, that's a lot of code which might be very confusing. In general, you'll just copy paste this into the start of every one of your programs and alter the filename depending on the question. In Java, we use the BufferedReader class and PrintWriter class for quick input. Remember, every declaration of a variable is in the format [type] [variablename]. That means the first line is simply grabbing a File, which we later pass into the BufferedReader. The BufferedReader is declared next, and when declaring a BufferedReader, we need to pass in a FileReader which opens the file. The PrintWriter does the same thing. This is probably very confusing, but the important thing here is that we're declaring a variable **reader** and a variable **writer** to do our input output. To get a raw input line from reader, I assigned a new variable **line** to reader.readLine(), which gets a newline from the **reader**. The example on the bottom demonstrates how we use a writer to write a string into a file. Remember to add **writer.close()** at the end of the program to close the stream. Also, note the **throws IOException** after the main(String[] args), which basically just tells the program to be ready for an error from io in case the file's corrupted, doesn't exist, etc.

Note that this is getting an input from a file. If we aren't going to be reading from a file and instead a stream like websites such as Codeforces and CSES, we need to use a slightly modified code. Here it is:

```
import java.io.*; //imports the io class from java

public class Test {

   public static void main(String[] args) throws IOException{
      BufferedReader reader = new BufferedReader(new
          InputStreamReader(System.in)); //initialize reader

      String line=reader.readLine(); //we're using our reader to read a
          line in the stream

       System.out.println("This is a test"); //using a writer to output
           some text
```

```
  }

}
```

When we're working with the stream, we no longer need a PrintWriter and can instead just output to the System stream.

11.1 Casting Variables from String to Numerical Values

Usually we read integers from these files, so in order to parse the raw string input into a integer or number, we need to use the Integer.parseInt(/*some string*/) function. As you might guess, this applies to Double and and Long. Here it is in action:

```
String strX="123";
String strY="456";
System.out.println(strX+strY); //prints "123456" instead of adding
    numerical values
int x=Integer.parseInt(strX);
int y=Integer.parseInt(strY);
System.out.println(x+y); //prints 579 because this is now addition of
    numbers

String strZ="123.123"; //we can do this for doubles too
int z=Double.parseDouble(strZ);
```

As you might imagine, this will be useful for processing inputs. "What if there are multiple integers on a line in the input?" you may ask. That's where the StringTokenizer comes in.

11.2 The StringTokenizer

Suppose the input is "1 2 3". How do we read these into separate variables? We use a StringTokenizer to parse this string input by some divider (usually a space). How does it know where to separate the individual numbers? By default the StringTokenizer separates the input by a space, but you can set it to a comma or something else for example. Consider the following

```
import java.io.*; //imports the io class from java
import java.util.*; //NOTE THIS IMPORT STATEMENT, StringTokenizer is
    part of java.util
public class Test {

  public static void main(String[] args){
    String line="1 2 3"; //here I'm setting the line, but this could
        be a reader input
```

```
        StringTokenizer st=new StringTokenizer(line); //by default inputs
            are parsed by spaces

        int a=Integer.parseInt(st.nextToken()); //use nextToken to get the
            next value
        int b=Integer.parseInt(st.nextToken());
          int c=Integer.parseInt(st.nextToken());
          //a=1, b=2, c=3

          String line="1,2,3"; //here I'm setting the line, but this could
              be a reader input

        StringTokenizer st=new StringTokenizer(line, ","); //separates by
            comma

        int x=Integer.parseInt(st.nextToken()); //use nextToken to get the
            next value
        int y=Integer.parseInt(st.nextToken());
          int z=Integer.parseInt(st.nextToken());
          //x=1, y=2, z=3
    }

}
```

There's many ways to read inputs, but in general I use the BufferedReader and StringTokenizer for my inputs as it is the fastest way to do so.

12 Conclusion

This should serve as a good quick introduction to Java, but you won't really know Java until you start writing a lot of code. A good place to get practice when starting out is CodingBat. For more challenging problems, check out CSES and Codeforces.

For "homework" attempt the first 5 problems on CSES's site: *Weird Algorithm, Missing Number, Repetitions, Increasing Array, and Permutations.* You'll come out feeling more confident in your Java skill and a warmed-up brain ready to tackle more Competitive Programming problems as they come. However, and I cannot stress this enough, **DO NOT LOOK UP THE ANSWERS!** If you are having difficulty, reach out to one of us captains or another club member, but don't look up a solution unless you've struggled for an hour or two. Contests like the USACO give you 4-5 hours for 3-4 problems and most people send the entire time period getting 1 or 2 problems done. It is natural to struggle when you're starting with something but it is essential that you embrace that struggle rather than run away from it. Happy Coding!