Differences in Java versus C++ cheatsheet

If I had to compare C++ and Java, I would say that they fit these pictures pretty well

C++

Java

C++ really just lets you do whatever you want, which is exciting, but can really mess you up. Java on the other hand is much more protective. You still get to be creative and expressive, but it makes sure you to take away any obvious ways to hurt yourself (you just have to be more creative).

The languages are very similar syntax wise! If you have any doubts you should just go about how you would do things in C++ for most of the basic elements. I'm going to highlight a couple of differences.

Bools & Arrays

Want a C++ **`bool`** variable? Declare it as a **`boolean`** instead (ie **`boolean isEasy = true;`**).

To declare an array in Java it must be written in this way. You can declare the type and size however you like.

```
int[] arr = new int[5];
```

To get the number of elements in the array, just say **`arr.length`**

Java Strings

Java strings do work differently than C++. Remember how you know the internals of C++ strings as being arrays of characters, and you could access a C++ string like an array. In Java that is not possible.

Instead if you want to get an underlying character, you can use **`charAt.`**

```
String s = "Hello";

char ch = s.charAt(1); //ch now has 'e'
```

To find something in a string you can use the method **`indexOf,`** which takes in either a string or a character and returns the integer position of where it is located. If it's not found, indexOf returns -1.

```
Int pos = s.indexOf("lo") //pos would return 3, the start of lo
```

To extract part of a string like C++ java has a substring function. Unlike C++, Java's substring works slightly differently:

```
String chopped = s.substring(1, 3) //"el"
```

Here, the numbers for substring stand for the range of characters, where you include the first one but not the 3rd. In this case, you can think of substring as saying include all the characters that start at position one and go up to (but not including) position 3. If you omit the second number, Java will give you the entire string from the starting character given up until the end.

Java has two meta-types, primitives and objects

Primitives are the basic types you know and love from C++, things like **`int`**, **`char`**, **`double`**, and **`boolean`**. You can immediately identify them based on case. Primitives are all in lower case while other types in Java, like **`String`**, are in uppercase. Anything that is in uppercase is just like a class/object from C++, it was written in a class file, which are java files. There are not separate .h and .cpp files in Java, the declaration and definition of a class is contained in the same file. Like in 53, the code that is in these java files I will refer to as a **CLASS** (the blueprints), while an actual instantiation aka instance aka something that is created to follow the rules of that class will be called an **OBJECT**. You can have many objects that follow the same class blueprints.

In Java there is no pointer notation.

Remember this?

```
int x = 5;

int *ptr = &x;
```

Yeah, well that's all gone, you won't have any code where the * means something other than multiplication. There's also no way to talk about the address of another variable. However, you'll often hear that java has no pointers. That's a misnomer. Any object that you create will always be holding a pointer to that object. When I create a Fraction object based on the Fraction.java class writing something like this:

```
Fraction f1;
```

Java will create a variable named **`f1`**, but that variable holds a pointer to **`f1`**. Saying immediately something like

```
f1.setDenominator(2);
```

Will trigger an error. You have to think of this as if just like in C++, you created a variable that holds a pointer only, but you didn't create space for the Fraction object itself. By calling **`new`**.

All Objects you create in Java will be created on the Heap.

If you really want to create an object, you will have to create it using the new operator. That will, like in C++, create the object that you want on the heap where you can then store information in it and call its values. So easiest thing would be to just get in the habit of always initializing all of your variables.

```
Fraction f1 = new Fraction(1, 1);  //Remember to call new!
```

```
f1.setDenominator(2); //now this will make the fraction 1/2
```

Java will take care of deleting objects for you.

Since it's on the heap, you might start feeling weird about having to free your own memory or call delete on objects you don't need anymore. Well, java does that for you now. And yes to the "l33t h4x0rs" this may be more inefficient and causes java to be slower, but the seconds you lose in running the program will outweigh the hours you spend trying to figure out freed memory exceptions.

Output/Input stuff has changed

I'm not gonna lie, cout, cin was pretty easy to use. Java has recognized though that there are some potential dangers in getting input that isn't in the right type for example, so they have made getting input a little bit more difficult. Printing out to a console has been replaced by doing something like this.

```
cout << "Hello, my " << age << "year-old friend!" << endl; //C++

System.out.println("Hello, my " + age + "year-old friend!"); //Java
```

Notice that Java doesn't have the **`endl`** at the end of the statement. It has been replaced with the **`println`** statement, if you don't want to go to the next line, call **`print`** instead of **`println`**.

Java has programmed many things for you

In addition to things like ArrayLists, which is an infinitely expandable array that will shift elements around for you, Java has implemented many useful classes that you can leverage immediately, without having to download any libraries. All you will need to do is import the library or collection of classes that you will need, similar to the #include in C++. Except here there are many more libraries and useful classes at your disposal. Ever wanted to have a class that stores a Date correctly? Done. Want to make an http request? Done. Want to sort an array? Done. To see, just go to this link.

http://docs.oracle.com/javase/8/docs/api/

Those are all java files that have been implemented, are stable and get leveraged by folks who work in Java so that they don't reinvent the wheel.

There is much to learn still

I can't tell you all the nooks and crannies or differences. I just wanted to highlight some major ones. You will undoubtedly have to learn a new language and IDE to work along that language when you go off to the real world, and you won't have much help, so take advantage of the jump start I gave you and start exploring and learning a new language without having complete direction from me. One good place to start is just to google "Java for C++ programmers", and see if anything interesting comes up. I'll leave you with one last thing to worry about here.

Now with your new found knowledge on Java, what will this print out?

```
Fraction f1 = new Fraction(5, 2);
Fraction f2 = new Fraction(2, 5);
Fraction f3;
System.out.println("f1:" + f1 + ", f2:" + f2 + ", f3:" + f3);
f3 = f1 = f2;
f1.setDenominator(10);
```

```
System.out.println("f1:" + f1 + ", f2:" + f2 + ", f3:" + f3);
```