Brief Introduction to Java

In this section we briefly discuss the Java programming language. We assume that the reader has already been exposed to basic programming in C++, including the object-oriented programming features of C++.

Reading

This is a list of various books for learning Java. Use them as reference

- Head First Java[1].
- Effective Java[2].
- Core Java I[3] and II[4]. Comprehensive, good for reference. Highly recommended as a book to add to your bookshelf.
- Java Tutorials online[5].
- A crash course from C++ to Java[6].
- Java 8 API[7].

Java Language Basics

Syntax

Java has borrowed a lot of syntax from C++, aiming to make the transition easier C++ programmers:

- We use semicolons to indicate the end of statements, and curly braces { ... } to group statements together into blocks.
- Conditional if–then–else expressions, for, while and do–while blocks and switch statements are all similar to C.
- There are eight *primitive* types: four integer number types (byte, short, int, long), each with fixed sizes, two floating point number types (float and double), a single character type char, and a boolean type with values true and false.

[1] http://shop.oreilly.com/product/9780596009205.do
[2] https://www.amazon.com/dp/0321356683
[3] https://www.amazon.com/Core-Java-I-Fundamentals-10th/dp/0134177304
[4] https://www.amazon.com/Core-Java-II-Advanced-Features-10th
[5] https://docs.oracle.com/javase/tutorial/index.html
[6] http://www.horstmann.com/ccc/c_to_java.pdf
[7] https://docs.oracle.com/javase/8/docs/api/

- Standard arithmetic operators[8] are similar in Java as in C.
- There is a standard math library[9]
- The language is statically typed, and every variable must have a type that is specified when the variable is declared. In particular, a variable cannot be assigned before it has been declared, and it cannot be used before it has been assigned a value.
- There is a `null` value that is analogous to the null pointer in C++.
- We can form arrays of values of specific length, and a shorthand notation can be used to initialize such an array with values: `int[] a = { 1, 4, 5 };`
- Everything else is an object of some class. Classes are capitalized: `Item`, `LinkedList`, `Vertex`, `Person`. Class constructors are used to create new objects: `Person aPerson = new Person("Peter");`
- Java has exceptions that we can use to indicate that something went terribly wrong with our code.
- Code is organized via functions (though they are always associated with a class as we will see later), and the types of the function arguments determine which code will be executed (i.e. two functions can have the same name but different parameter types).
- In Java, like in C++, we can have *static* class methods and properties, indicated via the keyword `static`, and we can classify all object and class methods and properties as *public* or *private* (or also *protected*).
- You can specify a property to be *constant* by using the keyword `final`. Constant properties are typically written in all capital letters.

There are also a number of differences:

- *Strings* in Java are not just arrays of characters. They are objects that you handle like other objects. They are *immutable*, so you cannot simply change a part of a string. You can however create a new string by concatenating together two other strings. Also, a powerful `StringBuilder` class is used when you want to incrementally build a string from smaller components.
- Java offers a second, more convenient, form of the `for` loop, often called the "foreach" loop, where you iterate over the elements of an array or list, rather than the indices: `for (Item item : listOfItems) { ... do something with item ... }`. This is a safer iteration approach, as you don't have to worry about running outside the array bounds, and it also works for other iterable structures that are not technically arrays. It is in many ways similar to Python's `for–in` loop.

[8] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html
[9] https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html?is-external=true

- Java has no pointers. In most instances where you would have used a pointer, you just don't, and Java handles it all behind the scenes for you. Everything that is not a primitive type is in effect a reference to an object. Equality between objects (==) tests quite literally if the objects occupy the same place in memory. Most classes also implement an `equals` method, that determines a more meaningful equality test (based for example on the properties of the object).

- Memory in Java is *managed*. An elaborate *garbage collection* mechanism is responsible for allocating and releasing the memory when needed. You never have to worry about memory leaks.

- All functions in Java are in fact *methods* of an object, or static class methods. There are no free-standing functions.

- Java uses *interfaces*, which are somewhat similar to the `.h` files in C++. An interface specifies a set of methods that a class should *implement* in order to claim that it obeys this interface. Interfaces are used to enforce certain behaviors: For instance any class that implements the `Stack` interface must provide certain methods for *popping* from the stack, *pushing* onto the stack etc.

- Java uses *generics*, which are somewhat analogous to C++ templates, to allow collection classes to be implemented without regard to the particular type of their contents (i.e. we can have a list that contains `Person` objects, or a list that contains `Vertex` objects, and we only have to write the code once). So for example the `ArrayList` class is usually written as `ArrayList<E>` where `E` is some other class indicating the contents of the list. A list of strings for example would be `ArrayList<String>`.

- Java code is not compiled into machine code directly. It is compiled into a low level language called *Java bytecode*, and it is then interpreted by the *Java Virtual Machine* (JVM). This makes the code much more portable: Compiled Java code can run on any system that has the JVM.