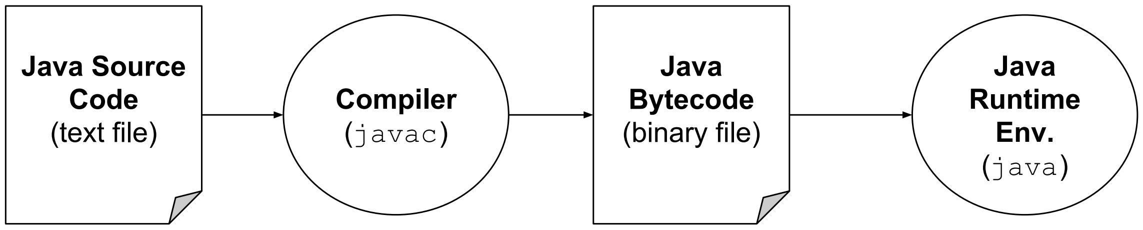
Java is Compiled

Unlike Python, which is an interpreted langauge, Java code is *compiled.* In Java, a *compiler* reads in a Java source file (the code that we write), and it translates that code into **bytecode**. Bytecode is an intermediate file format that represents low-level Java operations in a machine-independent way. The Java virtual machine then reads the bytecode and executes our program.



Practically, this means we write some code in our favorite text editor, run the `javac` compiler on that code to create a `.class` file, then run that file in the Java runtime environment using the `java` command.

```
$ atom HelloWorld.java
$ ls
HelloWorld.java
$ javac HelloWorld.java
$ ls
HelloWorld.class  HelloWorld.java
$ java HelloWorld
<output of HelloWorld.java program>
```

This also means that there is no REPL in Java. We must write an entire program, compile that program, and run that program. For every change we make, we must recompile. This limits our ability to rapidly prototype our code.

Punctuation

In Python, whitespace matters: indentation defines the scope of our code, and most control structures end with a colon (:). Java syntax relies much more heavily on punctuation. Every line must end in a semicolon (;). As we will see later, we use curly brace pairs to define scope ({}), and parentheses in most of our control structures.

Comments are also defined differently. Python uses the hash (#), but Java actually has two types of comments:

- The double backslash defines a single-line comment: //
- Multi-line comments are enclosed in dot-slash pairs: /* */

Standard Output and Printing

At this point, we are very familiar with the Python3 `print()` function: `print()` is a very useful way to debug our code by writing values to standard output. In Java, we have the `System.out.print()` and `System.out.println()` functions instead. `System.out.println()` adds a newline at the end of each print.

```
System.out.println("Hello World");
```

Java and Types

Java has builtin types that are similar to builtin Python types, but numeric Java types have a fixed size. This means that we must know the largest value that we plan to represent, and use a large-enough type to store that value.

- `byte`: 8-bit integer
- `short`: 16-bit integer
- `int`: 32-bit integer
- `long`: 64-big integer
- `float`: 32-bit real number
- `double`: 64-bit real number
- `boolean`: `false` and `true`
- `char`: single character (declared with single quotes)

In addition to these builtin types, the Java `String` and `Vector` classes are similar to Python's `str` and `list`. However, Java `String` literals must be defined with double quotes (""), and every element in a `Vector` must be of the same type.

`String` is part of the standard Java API, but the `Vector` class must be imported. To import a class, you use the `import` keyword:

```
import java.util.Vector;
```

This imports the `Vector` class from the `java.util` package.

Java is a Statically Typed Langauge

Python gives us a lot of flexibility as programmers, but as we have seen during the semester, subtle bugs often go unnoticed until our programs throw a `TypeError` exception at runtime. In Java, the compiler checks the types of our variables, verifies that they are compatible, and outputs errors *before* we are able to run our code.

Because of static typing, we must *declare* the type of every variable *before* assigning it a value, and once we declare a variable's type, its type cannot change.

```
int a;
String s = "text";
```

We often declare our variables at the top of a scope and then assign values later, but this is just a throwback to how things are done in C. The only requirement is that we declare a variable somewhere before we use it.

For certain types, we can do what is called *casting*. We cast a value from one type to another by specifying the new type in parentheses before the value. This is legal in many cases, but sometimes casting will cause our programs to raise a `ClassCastException`. The compiler cannot know when this will happen beforehand; `ClassCastExceptions` are runtime errors.

```
int a = 3;
long b = (long) a;
Object c = "Everything is a subclass of type Object";
String s = (String) c;
```

Booleans, Conditionals, and Comparisions

Python uses if/elif/else clauses. Java instead uses if/else if/else. In Java, the condition must be enclosed in Parentheses, and the conditional statement does not end in a colon. However, if your code block contains more than one line, you must enclose the entire block in braces.

```
if (a == true) {
    i = i + 1;
    System.out.prinln("This multiline block required braces");
} else if (i < j) {
    i = 0;
} else {
    return i;
}
```

Python boolean expressions are combined using `and` and `or`, and negation is done using `not`. Java instead uses symbols: && (logical and), || (logical or), and ! (logical not). Again, this is an instance of Java being similar to C.

Comparision of == and != is different in Java as well. In Python, this checked if two objects had the same *value*. In Java these check if two objects are the same object.

Loops

Java loops are logically similar, but quite different in syntax.

For

There are actually multiple ways to construct a for loop in Java. Here is a code snippet uses two of those ways. It fills a Vector of integers with the numbers 0-10 and then prints those integers to standard output:

```
Vector<Integer> v = new Vector<Integer>();
// for (initialization; stopping condition; update)
for (int i = 0; i < 10; i++) {
    v.add(i);
}

// for (type element : iterable_collection)
for (int i : v) {
    System.out.println(i);
}
```

The first for loop shows the classic way to iterate over indices (although it uses the Vector class's add() method to append to the collection instead of indexing into the Vector using bracket ([]) notation). Everything before the first semicolon is executed exactly once, immediately before beginning the first loop iteration. This is a good place to declare and initialize local loop variables. The boolean expression between the next two semicolons is executed before each loop iteration. If the expression evaluates to `false`, the loop body is skipped. Everything that follows the last semicolon is executed after each iteration; this is a good place to update any loop variables, often by incrementing/decrementing.

The second for loop is similar to the "for thing in things" syntax in Python. It declares a local variable of the type that is stored in the iterable collection (here, Vector `v` stores Integers, so we declare a local Integer `i`), and then it executes the loop body once for each object in that collection.

While

Java has two while loops: the while and the do-while. The difference is when the loop condition is checked. In a while loop, the check happens first.

```
while (expression) {
    statement(s);
}
```

In a do-while loop, the check happens last.

```
do {
    statement(s);
} while (expression);
```

Thus, a do-while loop is guaranteed to execute at least once.

Classes

Unlike Python, which supports Object Oriented Programming but does not require it, everything in Java must be inside a class body. Classes are defined with the `class` keyword, and class names start with a capital letter and use the CapWords style. The class body is contained with in curly braces ({}).

```
class MyClass {
    <class body goes here>
}
```

Classes can have member variables associated with them to represent their internal state. In Java, member variables are declared in the top-level class body. To refer to a class variable within a function, you can use the `this` keyword (you can think of `this` like the `self` convention in Python, except `this` is part of the language), but `this` is not required. You can also declare member variables as public, private, or protected.

```
import java.util.Vector;

class MyClass {
    public int i;
    private Vector<String> v;
    protected char c = 'x';

    <rest of class body goes here>
}
```

Functions

A function specifies its visibility (public, private, protected), its return type, and a formal parameter list (each parameter must also specify a type). Here is an example of a simple function to add two numbers:

```
public int addNums(int a, int b) {
    return a + b;
}
```

Java functions must always appear inside a Java class. A function is called by specifying an object of the class and using a dot (.). If the function does not depend on the internal state of an object, the function can be declared as a static function. To call a static function, use the name of the class followed by a dot (.).

Unlike Python, we can define mulitple Java functions with the same name as long as they have different arguments (types or number). The arguments are used to decide which function to call. This is called *overloading*.

The main method

For a Java program to be executable, we must define a `main` method. The `main` method must be public and static, and it takes an array of Strings as its argument. The String array is how command line arguments are passed into the program.

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```