

# APCS Notes

Yicheng Wang

2014-2015

# Contents

<b>1</b>	<b>2014-9-8</b>	<b>3</b>
1.1	Comparison of Programming Languages . . . . .	3
<b>2</b>	<b>2014-9-10</b>	<b>4</b>
2.1	"Hello World" in Java . . . . .	4
2.2	Running Java . . . . .	4
2.3	Java technicalities . . . . .	4
<b>3</b>	<b>2014-9-11</b>	<b>5</b>
3.1	Moving into the "java way" of doing things! . . . . .	5
<b>4</b>	<b>2014-9-15</b>	<b>5</b>
4.1	Typical anatomy of Java Program . . . . .	5
<b>5</b>	<b>2014-9-17</b>	<b>7</b>
5.1	Instance vs Local variable . . . . .	7
<b>6</b>	<b>2014-9-18</b>	<b>7</b>
6.1	Setter Function . . . . .	7
6.2	Return Type . . . . .	8
6.3	Style Stuff . . . . .	8
6.4	Constructor Functions . . . . .	8
<b>7</b>	<b>2014-9-19</b>	<b>8</b>
7.1	Full Anatomy of a Java Program . . . . .	8
7.2	work! . . . . .	9
<b>8</b>	<b>2014-9-22</b>	<b>9</b>
8.1	string literal vs new function . . . . .	9
8.2	string methods . . . . .	10
8.2.1	.equals(jstring <sub>i</sub> ) method: . . . . .	10
8.2.2	.equalsIgnoreCase(jstring) method: . . . . .	10
8.2.3	.compareTo(jstring <sub>i</sub> ) method: . . . . .	10
8.2.4	.contains(jstring <sub>i</sub> ) method . . . . .	10
8.2.5	.endsWith(jstring <sub>i</sub> ) and .startsWith(jstring <sub>i</sub> ) . . . . .	10
8.2.6	.isEmpty() . . . . .	10

# 1 2014-9-8

## 1.1 Comparison of Programming Languages

### **Scheme:**

Annoying Prefix Notation

strict syntax

not object oriented

only separated by parenthesis

IDE NOT necessary

Mostly recursion + list

Functional Programming Language (Everything is a function)

### **Netlogo:**

GUI based

Shines on Interactive Modeling

Bad for input/output data

Parallel Programming Language

Not a general-purpose language – ONLY USEFUL IN NETLOGO ENVIROMENT

Netlogo IDE (Integrated Development Enviroment) NECESSARY

### **Python:**

High-level Language

Uses Indentation

infix math + prefix function

Linear processing

General-purpose language

Interpereted Language

IDE NOT necessary

### **Java:**

Object oriented

infix math + prefix function

Mid-level Language

### ***ΛT<sub>E</sub>X***

Markup Language

Compiled Language

## 2 2014-9-10

### 2.1 "Hello World" in Java

Last year, we learned how to write the "Hello World" program in python.

```
1 def hello():
2     print "Hello world!"
3
4 hello()
```

Now, we'll learn about how to do this in Java:

Java is more restrictive than python, java programs are usually in their own folders. Java's invented for portability and "amount of stupid/super-smart people." Different smart people have different ways of approaching problems. Java's designed to limit people's ways of doing things to make big project easy. Real good programmers don't like java... b/c it's restrictive. Java is designed to be industrially viable.

An object defines a specific thing within your program. Everything in java is an object. A Class = object type.

Tradition = 1 class per file, named starting with upper-case letter

Here's a simple program in Java:

```
1 /*
2     This is a null line
3     C'est un comment!
4 */
5
6 // C'est un end-of-line comment
7
8 import java.io.*;
9 import java.util.*;
10
11 public class Hello { // public = the outside world (aka other things in your
12     program) can see this
13     public static void main(String[] args) {
14         System.out.println("Hello World");
15     }
16 }
```

### 2.2 Running Java

Source code (foo.java) → Java compiler (foo.class) → JVM

Note that java doesn't compile to machine code, java compiles to javaBiteCode using JVM. This is where the portability comes in.

### 2.3 Java technicalities

method = function in python

You need a method in one of you're classes called "main"

## 3 2014-9-11

### 3.1 Moving into the "java way" of doing things!

Java is object oriented. Object oriented means that the world is made of objects. Every object has its unique attributes. Objects also have abilities, aka things they can do. Every program in java is made of objects.

Let's take the example of a simple chess program. An example of an object would be a pawn. It would have attributes like color and position. Its abilities would include moving and attacking. However, these pawns are different, White pawn 1-8 and Black pawn 1-8. They behave in the same way, but they have different positions. You don't want 16 separate definitions b/c most of them are the same. Therefore one would create a class for all of the pawns, which would define the "info about objects." We then make objects which are known as "instances of a class." Objects are made based on the definitions defined within the class.

Hello world program 2 – the java way:

```
1 // We'll use objects to do stuff
2
3 import java.io.*;
4 import java.util.*;
5
6 public class Greeter {
7     // We put the attributes here
8
9     // We put the abilities here
10    // In Java, these are called methods
11    // Methods are functions, but they belong to specific classes
12    public void greet() {
13
14        // public = can be called from outside the class
15        // void = this doesn't send anything back, like null returner in C
16
17        System.out.println("Hello world!");
18    }
19 }
```

## 4 2014-9-15

### 4.1 Typical anatomy of Java Program

A program is consisted of objects. One must tell java where to start the program – i.e. public static void main One calls that class "driver," it starts the java program.

Driver.java:

```
1
2 import java.io.*;
3 import java.util.*;
4
5 public class Driver {
6     public static void main(String[] args) {
```

```

7
8    //How to use the greeter within the driver.
9
10   Greeter g;
11   //Creates a local variable to be of type greeter
12
13   /*
14   Variable declaration , all variables must be declared
15   like global , turtles-own and patches-own variables in netlogo
16   Declaration specifies the type of the variable
17   local variable = a variable only visible/usable within a method,
created when the method is called , destroyed when the function exits
18   */
19
20   /*
21   When main is ran , it occupies some memory on the computer
22   Greeter g is a small box within main, we need to do something with it
23   or java refuses to do stuff with it
24   */
25
26   g = new Greeter();
27   /*
28   New:
29       1. Allocates enough memory to store a Greeter.
30       2. Do whatever's necessary to setup / initiates the memory to be a
Greeter.
31       3. Returns the address of the memory that was allocated.
32
33   The assignment statement stores the address in g.
34   */
35
36   System.out.println(g);
37
38   // This prints the location of the variable g within the memory
39
40   /*
41   When this file is compiled , Greeter is compiled as well
42   All methods/class called during main are compiled as well
43   */
44
45   g.greet();
46   /*
47   Accesses the greet method within the class g.
48   */
49   }
50 }

```

## 5 2014-9-17

### 5.1 Instance vs Local variable

The instance variable is defined within the class and can be called from outside (if public) as well as methods w/i the class. Note that local variable within the methods overshadow instance variables of the same name. To assign a public instance variable from an outside object is: `objectName.variable name = new Variable Type variable value;` Instance variables are like turtle variables in NetLogo.

Instance Variables: Declared in class outside the methods. Usually at the top. Each instance (object) of the class has its own copy of the instance variables.

Ex.

```
1 public class Driver {
2     public static void main(String[] args) {
3         Greeter g1 = new Greeter();
4         g1.greeting = new String("Sup!");
5         g1.greet();
6     }
7 }
```

In here, the Driver class assigns the instance variable "greeting" a new value. If greeting is set to private, the above code will not work. In java, we almost NEVER make instance variables public so you can't assign them from outside. Instead, we write public "set" methods within the class which then assigns the private instance variable. "Set" methods will be covered in detail next time.

## 6 2014-9-18

### 6.1 Setter Function

A setter function edits a private instance variable from within its own class. An example follows:

```
1 public class Greeter4 {
2     private String greeting = new String ("Hello World!");
3     public void greet() {
4         // String greeting = new String("Sup!");
5         System.out.println(greeting);
6     }
7     public void setGreeting(String s) { // This is the setter function
8         greeting = s;
9     }
10    public void ungreet() {
11        System.out.println("I'm out!");
12    }
13 }
```

## 6.2 Return Type

In java, the name after public/private specifies the return type. A method can return any type of value (String, Int, even custom classes). When the function doesn't return any value, the type is "void"

An example getter method follows:

```
1
2    \\ blah stuff above
3
4    public void setGreeting(String s) {
5        greeting = s;
6    }
7
8    \\blah stuff below
```

Note that when the getter function is called, it is tantamount to a STRING! It can be used whenever a string is used.

## 6.3 Style Stuff

Stylistically speaking, in most languages, we should usually avoid using void and print stuff. However, they should return a value and then be printed in the main function.

## 6.4 Constructor Functions

These functions are called only when new objects are declared. Constructor functions are always public, its name is the name of the class, and there's no return value (NOT VOID, simply NO return value). This will be called when you run the new statemet.

Note that once we write a constructor, we lose the default constructor. This means all assignments must have a set parameter list. We solve this problem via overloading. We'll make multiple constructors, for example, we make 2 constructors, one with a String parameter while the other one doesn't have a parameter.

# 7 2014-9-19

## 7.1 Full Anatomy of a Java Program

Super generic java code of a class:

```
1 import java.io.*; // import pacakges (iff you need them)
2
3 public class genericClass { // class name, camelTyped, same name as file
4     // instance variables
5
6     private String s; // instance variables are almost always private
7
8     private String s2 = "hello"; // Declaration and assignment can be done at
9     once
```



```

10     private String s3,s4,s5; // multiple declaration seperated by commas
11
12     // constructors <- called automatically on "new"
13
14     public genericClass(String s) { // ALWAYS PUBLIC and NO RETURN VALUE
15         // do stuff here!
16     }
17
18     public genericClass() { // Constructors can be overloaded
19         // do stuff!
20     }
21
22     // methods
23
24     public void method1(params) {
25         blah;
26     }
27
28     private String method2(params) {
29         blah2;
30     }
31 }

```

## 7.2 work!

Write a method greetPerson, which takes 1 parameter (name) and appends the name to the greeting.

Write another method called "LOUD GREET," which returns greet in AllCaps.

# 8 2014-9-22

## 8.1 string literal vs new function

Let's look at the following code:

```

1     public void test1() {
2         String s1 = "hello";
3         String s2 = "hello";
4         String s3 = new String("hello");
5         String s4 = new String("hello");
6         System.out.println("s1 == s2:" + (s1 == s2));
7         System.out.println("s1 == s3:" + (s1 == s3));
8         System.out.println("s1 == s4:" + (s1 == s4));
9         System.out.println("s2 == s3:" + (s2 == s3));
10        System.out.println("s2 == s4:" + (s2 == s4));
11        System.out.println("s3 == s4:" + (s3 == s4));
12    }

```

When run, everything except the first line returns false. This is because of the different means of variable assignment. When s1 is assigned, java creates a block of memory to store the sting "hello." When s2 is assigned, java checks for the existance of "hello" and finds the

memory block of s1. Then s2 is assigned to the same memory as s1. Therefore s2 equals to s1. However, new function doesn't check pre-existing "hello" but creates a new memory block. Therefore the memory location of s1,s3, and s4 are all different.

## 8.2 string methods

### 8.2.1 .equals(*string*) method:

It compares the literal value of the string instead of the memory location.

### 8.2.2 .equalsIgnoreCase(*string*) method:

Same as .equals, but ignores case

### 8.2.3 .compareTo(*string*) method:

It's like a dictionary ordering. Returns 0 if the original object is equal to the new string

Returns *>* 0 if the original object is greater than the new string

Returns *<* 0 if the original object is less than the new string

### 8.2.4 .contains(*string*) method

:

Returns if the new string is w/i the old string.

### 8.2.5 .endsWith(*string*) and .startsWith(*string*)

As the name suggests, ends with blah and starts with blah

### 8.2.6 .isEmpty()

Returns true if the string is empty. False otherwise.