APCS Notes

Yicheng Wang

2014-2015

Contents

1 2014-9-8

1.1 Comparison of Programming Languages

Scheme:
Annoying Prefix Notation
strict syntax
not object oriented
only seperated 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
infixed math + prefix function
Linear processing
General-purpose language
Interperted Language
IDE NOT necessary

Java:
Object oriented
infixed math + prefix function
Mid-level Language

$LaTeX$
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.

```
def hello():
    print "Hello world!"

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:

```
/*
    This is a null line
    C'est un comment!
*/

// C'est un end-of-line comment

import java.io.*;
import java.util.*;

public class Hello {  // public = the outside world (aka other things in your program) can see this
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

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 seperate 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 –¿ 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 otside 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

Stylisticly 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
      once
9
```

```
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 class genericClass {
2      public void test1() {
3          String s1 = "hello";
4          String s2 = "hello";
5          String s3 = new String("hello");
6          String s4 = new String("hello");
7          System.out.println("s1 == s2:" + (s1 == s2));
8          System.out.println("s1 == s3:" + (s1 == s3));
9          System.out.println("s1 == s4:" + (s1 == s4));
10         System.out.println("s2 == s3:" + (s2 == s3));
11         System.out.println("s2 == s4:" + (s2 == s4));
12         System.out.println("s3 == s4:" + (s3 == s4));
13     }
14 }
```

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 stirng 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 sting is empty. False otherwise.

9 2014-9-23

9.1 Tour of Primitive Stuff in Java

9.1.1 primitive data types

Java has classes of Strings and any class we write. Java also have primitive data types:

int	integers
double	double-percision integers
char	single character
boolean	true/false

Strings and primitive data types work a bit differently. Every data we make on a 32-bit computer, the memory block assigned is a 32-bit blocks. The 32-bit block is called "work bit size." Any primitive type (their actual value) is stored in the variable's own memory location. Whereas class types, the variable's own memory location only stores the link to the actual class somewhere else in the memory. This is because one does not know how large a class can be. For example, on a 16-bit computer, if strings were stored as a primitive data type, the string can be most 4 letters. Therefore, to solve the problem with strings with arbitiary length, we only put a pointer within the class variable.

However, one must be careful with the types. Java is really strict about type operations. For example, 5.0 / 2.0 returns 2.5. However, 5 / 2 returns 2. Bottom line is that one should not mix int and double

9.1.2 If Statements

Basic form of if in java:

```
1
2 public class blah {
3
4     public void blah1() {
5         if (<boolean>) {
6             <insert statements here if the boolean is true>
7         }
8         else if (<boolean2>){
9             <insert statements here if the boolean1 is false and boolean2 is
    true>
10        }
11        else {
12            <insert statements here if both booleans are true>
13        }
14    }
15
16 }
```

A list of comparison operators:
Just everything in python, except double ampersand is and and double pipe is or.

10 2014-9-30

10.1 loops

Loop means doing stuff over and over again until something are met. We technically don't need loops, because recursion can replace loop anytime.

Examples of loop in languages:

- Netlogo forever button
- Netlogo while loop
- Netlogo repeat loop

- Python for loop
- Python while loop

While loops can be thought as repeated if statements. In Java, the while loop is perfectly analogous to the python while.

```
1
2     Codes
3
4     while (<boolean>) {
5         <statements>;
6     }
7
8     More codes.
```

11 2014-10-06

11.1 Software Development

Class dungeon crawl game – Stuyablo! Example = nethack. We'll talk about our version of dungeon crawl. You have different types of entities:

1. Monsters
    - Attribute: inventory
    - Attribute: Armor
    - Attribute: Strength
2. Wizard
    - Attribute: HP
    - Attribute: XP
    - Attribute: Level
    - Attribute: Mano
    - Attribute: Intellect
3. Warriors
    - Attribute: Weapon
    - Attribute: Shield
    - Attribute: Strength
    - Attribute: Range
4. Rogue

- Attribute: HP – Medium/Low
- Attribute: Speed – High
- Attribute: Agility – High
- Attribute: Luck – High
- Attribute: Accuracy – High
- Attribute: Power – Low
- Attribute: Intellect – High
- Attribute: Charisma – Medium/Low
- Ability: Pickpocket
- Ability: Poison/Assisinate
- Alignment: 0–3

5. Cleric

    - Attribute: HP
    - Attribute: XP
    - Attribute: Mano
    - Attribute: Healing power

Each of these will have certain attributes and abilities. But some of them have common things, like health, inventory, etc. However, something like Manna are not the same for everything. Warriors, thiefs don't have Manna. We can say the same for abilities that all should have – attack, defend, eat, pick up. However, things like spell-casting, pickpocket, etc. should be character-specific.

For flexibility, java has **Class Inherentence**. Which means that one class can be based on or is an extension of another class. So we can make a class called "characters" with instance variables being the common abilities and attributes (XP, Level, attack, pick-up, etc.) Then using class inherentence we can create specific character types.

Class Inherentence Example:

```
1
2  // BaseChar.java
3  public class BaseChar {
4      private int hp = 20;
5
6      public int getHp() {
7          return hp;
8      }
9  }
10
11 // Wizard.java
12 public class extend BaseChar {
13
14 }
15
16 // Wizard will have everything character has
```

12 2014-10-7

12.1 Subclass and Superclass

Referring to yesterday's code, the BaseChar class is called the superclass of Wizard. And Wizard is called the subclass of BaseChar. Note that the subclass only has access to the public instance variables. Therefore, the inherent class can only call public functions.

overriding: When a subclass has a variable or method of the same name and signature as a method/variable in the superclass. Thus overriding the thing in the superclass.

In order to access the superclass, we use super.¡method¿ to access stuff that's been overridden.

13 2010-10-8

13.1 More Inheritence Stuff

A superclass can be referred to any of its subclasses. However, it can only call the overridden methods. It cannot get extended methods.

Casting: call extended methods from superclass. Example:

```
1
2     Basechar c = new Basechar();
3     Mage m1 = new Mage();
4     Basechar c1;
5
6     c1 = m1;
7
8     ((Mage)c1).getManna();
```

The parenthesis is important because usually the "." operator takes precedence of the casting operator.

More local stuff overwrite less local stuff. In order to specify explicitly the instance variables, you use this.¡instance var¿.

Now suppose we get an attack method. We want to attack a basechar. Since all characters are extensions of Basechar. Ex. code:

```
1
2     public void attack(Basechar other){
3         System.out.println(getName() + "attacked" other.getName());
4     }
```

14 2014-10-9

14.1 Constructors and Inheritence

In order for a subclass to work, it needs to have everything set up for it in the superclass. Whenever one makes a subclass, it first calls the superclass constructor. It will also do so automatically. So you need to write each constructor at each level. If we don't explicitly call

the super constructor, i.e. super(); as the first line of our subclass' constructor, java will call the superclass' default constructor. Therefore, use super to specify what will be done.

Example Code:

```
1
2  public class Superclass {
3
4      private String name;
5
6      public Superclass() {
7          setName("DEFAULT");
8      }
9
10     public Superclass(String n) {
11         setName(n);
12     }
13
14     public void setName(String n) {
15         name = n;
16     }
17 }
18
19 public class Subclass extends Superclass {
20
21     public Subclass() {
22         super();
23         // This will set the name as "DEFAULT"
24     }
25
26     public Subclass(String name) {
27         super(name);
28         // This will set the name as the parameter "name"
29     }
30 }
```

15 2014-10-24

15.1 Array Stuff

Basic array declaration format: ¡type¿[] ¡name¿ = new ¡type¿[¡length¿]. "Type" is the type of stuff that'll be stored within the array called ¡name¿, like "int" "String" etc. Another more straight forward way of declaring arrays is to just type it after the equal sign. For example:

```
1
2 String[] message = new String[5] // string called message with 5 places
3
4 for (i = 0 ; i < 5 ; i++) {
5     message[i] = blah;
6 }
```

16 2014-10-27

16.1 Class Stuff – Static and Final

When one tries to access an instance variable within the same class, instance variables can't be used because the class is technically not refering to itself. To use the stuff within the same class, you do have to declare a new instance of the class within itself.

Final variables are variables whose values cannot be cahnged. For example ¡array¿.length() is a final variable. These variables can be used but cannot be assigned or changed.

17 2014-11-10

17.1 Exception Handeling

In java, error are known as exceptions. When exceptions occur, they crash the program and and returns an exception debugging message. However, we can create our own exception handeling mechanism. Note that each exception is a class and can be called throughout the program, as seen below:

```
1  public EE {
2      public void etest(int i) {
3          if (i > 10) {
4
5              throw new ArithmeticException();
6          }
7      }
8  }
9
10 // New Class!
11
12 public Driver {
13
14     public static void main(String[] args) {
15         EE ee = new EE();
16     }
17
18     System.out.println("Before the test");
19
20     EE.etest(1);
21     EE.etest(11);
22
23     System.out.println("After the test");
24 }
```

Note how EE.etest(11); crashed the program with the ArithmeticException.

However, we can make the error message more useful by using a try-catch block. This construct will stop the program as soon as an error occurs and pause the program. The syntax of class Driver is as follows:

```
1  public Driver {
2
```

```
3     public static void main(String[] args) {
4         EE ee = new EE();
5     }
6
7     System.out.println("Before the test");
8
9     try {
10        ee.etest(11);
11        System.out.println("etest has been run");
12    }
13
14    catch (Exception e) {
15        System.out.println("etest crashed with Error: " + e);
16    }
17
18    System.out.println("After the test");
19 }
```

Note that catch prevents crashing of the program, instead it just send an error message. Also note that the "Exception e" parameter can be replaced by any specific error class such as ArrayIndexOutOfBoundsException or ArithmeticException, etc. It is also possible to stack a few catch phrases one after another to expect multiple types of errors.

It is important to note that try-catch construct should NOT be used to conceal errors from crashing the program. It should be used for cases in which java's compiler is being dumb, and gives error for things that can be fixed.

18 2014-12-09

18.1 Static vs Non-static

18.1.1 Static Methods

Static methods make the java class look like a C header file that can be included. Static variables and functions can be used without creating a specific instance of a class. For example, most Arrays.* functions are static methods, since whenever one uses it, one does no create an instance of said "Array," but the argument is feeded directly to the method. Static methods are useful when one creates a function that is not specific to any specific instance of said class.

One area where static methods will be useful is if we were to create a class of different sorting methods, as shown below:

```
1 public class Sort {
2     ... ...
3
4     public static <T> void isort(T[] data) {
5         for (int i = 1 ; i <= last ; i++){ // Sorting index
6             T newVal = data[i];
7             int a;
8             for (a = i ; a > 0 && newVal.compareTo(data[a - 1]) < 0 ; a--) {
9                 data[a] = data[a-1];
```

```
10              }
11              data[a] = newVal;
12          }
13      }
14
15      public static <T> void ssort(T[] data) {
16          T minValue;
17          int minIndex = 0;
18          for (int i = 0 ; i <= last ; i++) {
19              minValue = data[i];
20              minIndex = i;
21              for (int a = i ; a <= last; a++) {
22                  if (data[a].compareTo(minValue) <= 0) {
23                      minValue = data[a];
24                      minIndex = a;
25                  }
26              }
27              data[minIndex] = data[i];
28              data[i] = minValue;
29          }
30      }
31
32      public static <T> void bsort(T[] data) {
33          for (int i = 0 ; i <= last ; i++){
34              for (int j = i+1 ; j <= last ; j++)  {
35                  if (data[i].compareTo(data[j]) >= 0) {
36                      T temp = data[i];
37                      data[i] = data[j];
38                      data[j] = temp;
39                  }
40              }
41          }
42      }
43
44      ... ...
```

There are certain advantages of using static this way. Firstly it is easily accessalbe and makes sorting a lot easier.

18.1.2 Static Variables

These variables are just like static functions, and are also shared across all instances of the said class. This is not quite as useful as static methods because most variables are storage of parameters of the class. However, things like random number generator should be set to static and shared across all instances to ensure security.

19 2014-12-10

19.1 Interfaces

Java compiler is condescending and mean. Only overloading methods like .compareTo() is not enough to convince the java compiler that your class is comparable. Therefore We need to write a verbal contract saying "yo i will include .compareTo() in this class."

This contract is what we call "Interfaces." To "sign" these contracts, we use the "implements" command when declaring a class, like the following:

```
1 public class foo implements boo {
2     ...
3     stuff happens
4     ...
5 }
```

On the basic level, an interface is like an abstract class that dictate that the current class MUST HAVE certain functions. Some other functions, such as Arrays.sort(array of T) also requires the implementation of certain interfaces like Comparable.

We'll take the Comparable interface as an example:
The comparable interface requires the existance of method .compareTo(Object other), and would raise an error if that method is not found.