23/Sep/2017

Advanced Programming

Objected oriented Programming
- Encapsulation
- Inheritance
- Polymorphism

* By convention, the name of the class should match the name of the file that holds the program *

Naming Convention: First letter of a Java Class be upper-cased. Using lower case won't give a syntax error

Java is case sensitive

→ public static void main(String args[])

public: a keyword which allow the programmer control the visibility of class members.

static: void: main(String args[]):

System.out.println("Hello");

out is output stream connected to console.

println: Member of System class.

System is a predefined class that provides access to the system

byte, short, int, long — String
float, double
char
boolean

→ Type conversion and casting

- Automatic Conversion
  - → Two types are compatible both integer or both float etc.
  - → destination type is larger than the source type.
- Casting Incompatible types (Narrowing conversion)

```
int a;
byte b;
b = (byte) a;
```

for int → If value larger than the range of a byte, it will be reduced modulo. (the remainder of an integer division by the) byte's range.

for float → Truncation

→ Automatic Type Promotion in Expressions

* Java automatically promotes each byte, short or char operand to int when evaluating an expression*

```
byte a, b, c;   a=40;  b=50;  c=100;
int d = a*b/c;
```

These also causes confusing compile time error

```
byte b=50;
b = b*2; //Error! Cannot assign an int to a byte.
```

byte, short, char → int

if one operand is long then all are converted to long

if one float → entire to float

" " double → entire to double

Array

→ int a[];

a = new int[5];

→ int a[] = new int[12];

→ int[] a = new int[3];

* When you allocate memory for a multidimensional array, you need only specify the memory for the first (left most) dimension. *

```
int twoD[][] = new int[4][];
twoD[0] = new int[1];
twoD[1] = new int[2];
```

→ int[] nums, nums2, nums3;

↓

this declaration form is useful when specifying an array as a return type

String str = "this is a test";

* Modulus Operator can be applied to floating-point types as well as integer types. *

int x = 42 x mod 10 = 2

double y = 42.25 y mod 10 = 2.25

→ Implemented more efficiently by the Java run-time system than their equivalent long form

```
a += 5;          a = a + 5;
```

Bitwise Operator

>>> filled with 0.

>> : vacated bits on the left are filled with original sign bit

^ : XOR

```
boolean a = true;
boolean b = false;

x = y = z = 100; // set x, y, z to 100 allocated in Java

ratio = denom == 0 ? 0 : num / denom;
```

Versions of for loop

```
int nums[] = {1, 2, 3, ... 10};
int sum = 0;

for (int x : nums) {
    System.out.println("Value is: " + x);
    sum += x;    // Can use break; if do not want all sum
}

System.out.println("Summation: " + sum);
```

```
boolean done = false;
for (int i = 1; !done; i++) {
    if (interrupted()) done = true;
}
```

→
```
for (int x : nums) {
    System.out.print(x + " ");
    x = x * 10;  // no effect on nums
}
```

```
int nums[][] = new int[3][5];
// initialised
....
for (int x[] : nums) {
for (int y : x) {
System.out.println("Value is: " + y);
    sum += y;
}
}
```

Using break as a form of goto

```
boolean t = true;
    first: {
      second: {
          third: {
          System.out.println("Before the break");
            if (t) break second;
            System.out.println("This won't execute");
          }
        System.out.println("This won't execute");
      }
      System.out.println("This is after second block.");
    }
}
```

→ break helpful in exiting outter loop

```
outer : for (int i=0 ; i<3 ; i++) {
          Sop ("Pass " + i + " : ");
          for (int j = 0; j<100; j++) {
            if (j==10) break outer; // exit both loops
            Sop (j + " ");
          }
          Sop ("This will not print");
        }
        Sop ("Loops complete");
```

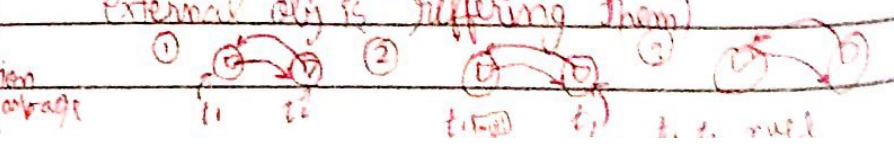
continue

```
outer: for (int i=0; i<10; i++) {
         for (int j=0; j<10; j++) {
           if (j>i) {
             System.out.println();
             continue outer;
           }
           Sop (" " + (i+j));
         }
       } Sopln();
```

Garbage Collection

```
public class Test
{ Test i;
  psvm (String[] args)
  { Test t1 = new test();
    Test t2 = new test();
    t1.i = t2;
    t2.i = t1;
    t1 = null; // No obj eligible for gc before execution
    t2 = null; // both eligible for garbage
  }
```

1. Nullifying the reference variable
2. Reassigning the reference variable
3. Object created inside method
4. Island of isolation in Java.
   (Object are reffering to each other and no external obj is reffering them)



5 Anonymous Object

new Test("t1"); → Anonymous obj without reference id

Not eligible for garbage collection

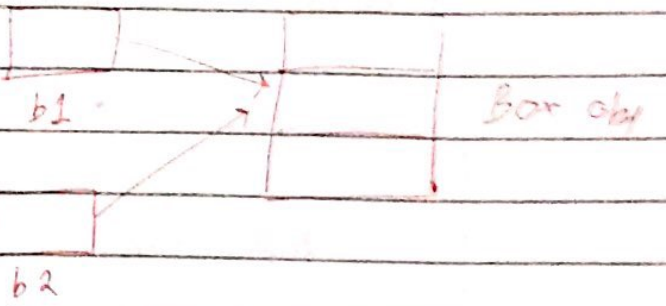
* If method returns the obj created inside it and we store this reference

declaring Objects of class

→ Box mybox = new Box();

→ Box mybox; // declare reference
mybox = new Box(); // allocate a box obj

→ Box b1 = new Box();
Box b2 = b1;



The this Keyword.

this.width

used to resolve name-space collisions

→ Overloading
→ Using Objects as parameters
→ Argument Passing
→ Returning objects

```
class Test {
    int a;
    Test(int i) {
        a = i;
    }
    Test incrByTen() { Test temp = new Test(a+10);
                       return temp; }
```

→ Recursion

→ Introducing access control
- public
- private
- protected

→ Understanding static

* When a member is declared static, it can be accessed before any of object of the class is created *

Restrictions
- → can only call other static methods
- → can only access static data
- → cannot refer to this or super in any way (even if the variable is static)

→ Introducing Final

final int file_New = 1;

doing so prevents its contents from being modified.

→ int a1[] = new int[10];

a1.length → give me size of the array that is 10.

→ Introducing Nested and Inner Classes

→ Exploring the String Class

- String myString = "this is a test";
- String myString = "I like Java";

String class contain several methods that we can use

```
boolean equals(String object)
int length()
char charAt(int index)
```

```
String Ob1 = "First String";
String Ob2 = "Second String";
String Ob3 = Ob1;

Sop("Length of Ob1: " Ob1.length());
Sop("Character at index 3: " Ob1.charAt(3));
if(Ob1.equals(Ob2))
    Sop("Ob1 == Ob2");
```

→ Variable length arguments

→ Scanner class in Java

```
import java.util.Scanner;

Scanner sc = new Scanner(System.in);
String name = sc.nextLine();
char gender = sc.next().charAt(0);
int age = sc.nextInt();
long mobileNo = sc.nextLong();
double cgpa = sc.nextDouble();
```

→ Inheritance

Subclass can acess members of ~~subclass~~ superclass (except private members and function

→ A superclass variable can reference a subclass object

```
class A
{
    int a;
    int b;

}

class B extends A
{
    int c;

}

class Test
{
    psvm(String args[])
    {
        A p = new B();
```

int l = p.a // valid
int m = p.b // valid
Int n = p.c // invalid

}

→ Using Super to call superclass Constructor

```
class A
{   int l, k, m;
    A(int a, int b, int c)
    {       l = a;
            k = b;
            m = c;
    }
}
class B extend A
{       int j;

        B(int z, int y, int r, int s)
        {   super(z, y, r); // call superclass constructo
            j = s;
        }
}
```

→ super-member

```
class A {
    int i;
}
class B extends A {
    int i // this i hides the i in A
    B(int a, int b) { super.i = a; /*i in A*/ i = b; // i in B.
```

→ Creating a multilevel Hierarchy

→ When constructors are called

```
Class A {
A() {
    Sop("A Here");
    }
    }
class B extends A {
B() {
    Sop("B Here");
    }
}
Class C extends B {
C() {
    Sop("C Here");
    }
}
class Test {
    psvm(String args[])
{
    C c = new C();
```

Output :

A Here
B Here
C Here

→ Method Overriding

Superclass and subclass have the same name then subclass hides the superclass variable

- if both have same name static function function do not override and function in A will be executed
- if both have same name one static and other is not static * ERROR *
- Same name non static the function in B executed.

A k;
B m = new B();
k = m;
B n = new B();

All func^n in A & B will be executed keeping in mind about overriding.

- Here function in B will be executed

A k = new A();

- Here function in A will be executed.

→ Abstract Class

object of this class can not be declared
Declaring reference is valid

```
abstract class Figure {
    double dim1;
    double dim2;
    Figure (double a, double b) { dim1 = a; dim2 = b; }
    abstract double area();
}
```

All the subclass of Figure must override this function.

→ Creating a multilevel Hierarchy
→ When constructors are called.

```
class A {
  A() {
    Sop("A Here");
  }
}
class B extends A {
  B() {
    Sop("B Here");
  }
}
class C extends B {
  C() {
    Sop("C Here");
  }
}
class Test {
  psvm(String args[])
  {
    C c = new C();
```

Output:

A Here
B Here
C Here

→ Method Overriding

Superclass and subclass have the same name then subclass hides the superclass variable

→ Miscellaneous

- `for(int i=0; 1; i++)` ← Error

  Should be boolean type.

- Main is a static function so non static functions can not be called in it.

```
class Test {
ps vm { fun(); }
void fun()
{ sop("Hi");
}
}
```

← Error (will work if we make the function fun of static method)

- Static local variables are not allowed in Java in function.

- If we create parametrized constructor then Java compiler does not creat default contructor.

- In Java we can acess grandparent's members only through the parent class.

- When an obj of class is created variables are declared before ~~the error~~ running of constructor.

- Default ~~acc~~ access is more restrictive. When derived class ~~is~~ overrides base ~~to~~ class function, more restrictive acess can't be given to the overridden function.

```
class Base {
    protected void foo() {}
}                                   //Error
class Derived extends Base {
    void foo() {}
}
```

- In method Overloading, most specific method is chosen at compile time

```
class Main {
    public static void gfg(String s)
    {   Sop("String");
    }
    public static void gfg(Object o)
    { Sop("Obj"); }
    psvm(String args[]) { gfg(null); }
}
```