CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

More on classes

Ch.6 & 7 of T2.

The Complete Reference- Java, 11th Edition, Herbert Schildt, Tata McGraw Hill Publishing.

And also refer to Class notes.

Methods

General form of a method:

type methodname1(parameter-list)

{ body }

Overloading methods in Java

- In Java it is possible to define two or more methods in a class with same name.
- In such case the methods are said to be *overloaded*.
- And this process is referred to as *method overloading*.
- This is how Java implements *polymorphism-*

"one interface multiple methods"

```
class Book
{
    void test(int a)
    {
        System.out.println("a is : " + a);
    }
    void test(int a, int b)
    {
        System.out.println("a and b are : " + a+" AND "+b);
    }
}
```

Passing Objects as parameters

```
class Line
{
    int length;
    Line (int l)
    {length=l;}
    boolean compare(Line x)
    {     if(this.length==x.length) return true;
          else    return false;
    }
}
```

```
class  Module62
{
    public static void main(String args[])
    {
        Line l1=new Line(10);
        Line l2=new Line(10);
        boolean b=l1.compare(l2);
        if (b)
                System.out.println(" Both l1 and l2 are of same length");
        else
                System.out.println(" l1 and l2 are of different length");
    }
}
```

Parameter passing (Call-b-value/call-by-reference)

```
class Line
{
    int length;
    Line(int l)
    { length=l;}
    void doubleVal( int a)
    { a=a*2;}
    void doubleLength(Line l)
    { l.length=l.length*2;}
}
```

```
class Module63
{   public static void main(String args[])
    {       Line l1=new Line(2);
             System.out.println("Call by value  :");
            System.out.println("value of Line length  before doubled is: "+
                                                              l1.length);
            l1.doubleVal(l1.length);
            System.out.println("value of Line length  after doubled is: "+
                                                              l1.length);
             System.out.println("Now Call by reference :");
            System.out.println("length of the line before doubled is: "+ l1.length);
            l1.doubleLength(l1);
            System.out.println("length of the line after doubled is: "+ l1.length);
    }
}
```

Recursion

Java supports recursion.

–Recursion is the process of defining something in terms of itself.

–A method can call itself. Such methods are called recursive methods.

```
class Factorial
{
    int fact(int n)
    {
    int result;
    if(n==1) return 1;
    result=fact(n-1) * n;
    return result;
    }
}
```

```
class Recursion
{
    public static void main(String args[])
    {
        Factorial f= new Factorial();
        System.out.println("Factorial of 4 is: "+ f.fact(4));
    }
}
```

fact(n)	n	fact(n-1) * n	result
fact(4)	4	fact(3)* 4	6*4=24
fact(3)	3	fact(2) * 3	2*3=6
fact(2)	2	fact(1) * 2	1*2=2
fact(1)	1	1	1

Storage for Java Programs

The memory used by a running java program is organized into two areas, called *segments*: the *stack segment* and the *heap segment*.

The *stack* is where memory is allocated for local variables within methods.

The *heap* segment provides more stable storage of data for a program; memory allocated in the heap remains in existence for the duration of a program. Therefore static variables and objects are allocated on the heap.

- When method is called space for parameters and variables is allocated on the stack.
- The structure of the stack includes a stack frame for each active method/procedure.
- There may be several frames in the stack at once for a given method/procedure if it is recursive.

- When a method calls itself, new local variables and parameters are allocated storage on the stack, and the method code is executed with these new variables from the start.
- As each recursive call returns, the old local variables and parameters are removed from the stack, and the execution resumes at the point of call inside the method.
- Execution of recursive calls are slow because of overhead due to function calls.

- Many recursive calls to a method can cause in stack overflow.
- In such case Java run-time will throw an error. But this is a rare event.
- The main advantage of recursion is that they can be used to write clearer code for algorithms that are iterative in nature.
- While writing iterative methods we must include an IF statement somewhere to force the method to return without recursive call being executed.
- Otherwise it will never return.

Access Protection in Java

Encapsulation gives rise to access control on class members.

How a member can be accessed is determined by access specifier.

	public	private	protected	default
Same class	Y	Y	Y	Y
Same Pkg. Sub class	Y	N	Y	Y
Same Pkg. Non Sub class	Y	N	Y	Y
Diff. Pkg. Sub class	Y	N	Y	N
Diff Pkg. Non Sub class	Y	N	N	N

Static members

The *static* can be attached before instance variable or a method name while declaring the same.

static variables

- they are global to class
- all instances will share the same copy
- not per instance basis
- can be called with class name without instances

static methods

- can call only static methods
- can access only other static data
- can not refer to *this or super*
- can be called with class name without instances

Nested Class

A *Nested class* is a class defined in another class.

If class B is defined inside class A, it is known to A but not outside A.

That is B does not exist independent of A.

All members (including private) of A are accessible by B.

But, A can not access members of B.

It is also possible to declare a nested class local to a block.

Any code outside Outer class cannot create an inner class object.

```
class Outer
{
    int outer_x;
    Outer(){outer_x=10;}
    void test(){ Inner inner=new Inner(); inner.display();}
    class Inner
    {
        void display()  {System.out.println(" display outer_x =" +outer_x); }
    }
}
```

```
InnerDemo
{
    public static void main(String args[])
    {
        Outer out=new Outer();
        out.test();
    }
}
```

Classes created

Outer.class

Outer$Inner.class

```
class Outer
{
    int outer_x;
    Outer() {outer_x=10;}
    class Inner
    {    int m;
        Inner(int a) {m=a;}
    }
}
class InnerClassDemo1
{
    public static void main(String args[])
    {
      Inner in =new Inner(55);
      System.out.println(" value of Inner clas m is: "+ in.m);
    }
}
```

```
C:\Users\Admin\JavaPrograms>jav
ac InnerClassDemo1.java
InnerClassDemo1.java:19: error:
cannot find symbol
Inner in =new Inner(55);
^
symbol: class Inner
location: class InnerClassDemo1
```

```
class Box
{
    int length;
    Box(int a) { length=a;}
}
```

```
class Demo
{
   public static void main(String args[])
  {
            Box b1=new Box(10); Box b2=b1;
            System.out.println(" box b1 lenth is :"+b1.length);
            System.out.println(" box b2 lenth is :"+b2.length);
            Box b3=new Box(10);
            System.out.println(" box b3 lenth is :"+b3.length);
            if(b1==b2)      System.out.println(" box b1 and b2 are same:");
            else     System.out.println(" box b1 and b2 are not same:");
            if(b1==b3)      System.out.println(" box b1 and b3 are same:");
            else   System.out.println(" box b1 and b3 are not same:");
   }
}
```

Overriding equals()

```
class Box
{
    int length;
    Box(int a) { length=a;}
    public boolean equals(Object obj)
    {
            Box b=(Box)obj;
        if(this.length==b.length) return true;
        else return false;
    }
}
```

```
class Demo
{
    public static void main(String args[])
    {
        Box b1=new Box(10);
        Box b3=new Box(10);
        if(b1==b3) System.out.println(" box b1 and b3 are same:");
        else     System.out.println(" box b1 and b3 are not same:");
        if(b1.equals(b3))System.out.println(" box b1 and b3 are same:");
        else System.out.println(" box b1 and b3 are not same:");
    }
}
```

Summary

- Adding Methods to class
- Method Overloading
- Parameter Passing
- Recursion with examples
- Instance variables
- The *static* keyword
- Nested Classes
- *equals()* method