CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Advanced OO Concepts
OOP-3
(Ch.3 of T1)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Content

- Constructors
- Method overloading
- Error handling
- Scope Concepts
- Multiple inheritance
- Operator overloading
- Object operations

Constructors

Constructors are not seen in non-OO programming. They look like methods, but not.

The have same name as the class, and have no return type.

Compilers will identify methods with no return type and with same name as class, as Constructor.

Purpose: To initialize java objects when they are created. They have no return type.

- Constructor is automatically called after object is created and new operator completes.
- In Java, a default constructor is supplied if user don't define his own. All instance variables will be initialized to zero or null(if it is object reference).
- If we explicitly define a constructor the default constructor is no more available.

Note: Always advisable to have our own constructor, than using default one. It is always good to have all attributes initialized.

When is constructor called?

It is called immediately after the object is created by ***new*** operator. The allows programmer to initialize instance variable as appropriate.

Ex

```
Class Box{
   int x;
    Box(int a){x=a;}
}
Class Demo
{
   public static void main(String args[])
   {
      Box b= new Box(7);
    }
}
```

Default constructor:

User Defined constructor:

Having multiple constructors (overloaded):

// class without user defined constructor

```
class Box
{
   int length, width, height;
   void printBoxDetails()
   {        System.out.println("Box length is : "+length);
            System.out.println("Box width is : "+width);
            System.out.println("Box height is : "+height);
    }
}
class Demo
{
            public static void main(String args[])
             {
                        Box b=new Box();  // using default constructor
                        b.printBoxDetails();
            }
   }
```

Note: when no user defined constructor exists, a default constructor will do the job.

// demonstrating overloaded constructors

```
class Box
{
   int length, width, height;
   void printBoxDetails()
   {        System.out.println("Box length is : "+length);
            System.out.println("Box width is : "+width);
            System.out.println("Box height is : "+height);
    }
    Box(int x, int y, int z){length=x;, width=y, height=z;}
    Box(){length=10;, width=5, height=2;}
}
class Test
{
            public static void main(String args[])
             {          Box b1=new Box();  b1.printBoxDetails();
                        Box b2=new Box(4,7,9);  b2.printBoxDetails();
            }
   }
```

Method overloading

Allows programmer to use same name over and over as along as the signature is different.

Signature: return type, name, parameter list(number, type, order)

In some languages return type is not part of signature. Ex. Java and C#

Error Handling

It is almost impossible to have programs that do not encounter abnormalities during execution in terms w.r.t., SW/HW etc.

There are three ways to handle errors if reported.

1. **Ignore**: not a good idea may lead to crash.
2. **Check for potential problems, and abort the program**. Display appropriate message to user and exit gracefully. Ok to some extent.
3. **Check for potential problems, catch the mistakes, fix the problem and proceed**. Ok but difficult to identify code that can generate error.
4. **Throwing an Exception**: Most OO languages provide feature called exception.

Exceptions are unexpected events that occur within a system.

They provide a way to detect mistakes and handle them.

Exceptions thrown from blocks where they occur.

We catch them in appropriate handlers and handle them.

```
{
    try{ // code expected to throw an exception }
    catch(Exception type1){ // exception handler for type1 }
    catch (Exception type2){ // exception handler for type2 }
}
```

```
class Exc
{ public static void main(String args[ ])
   { int d,a;
     d=0;
     a=42 / d;
     System.out.println("Divide 42 by zero.");
    }
 }
```

```
java.lang.ArithmeticException: / by zero at Exc.main(Exc.java: 4)
```

```
class ExDemo1
{    public static void main(String args[ ])
     {      int d,a;
                        try{
            d=0; a=42 / d;
            System.out.println("Divide 42 by zero.");
                        }
            catch (ArithmeticException e)
                        { System.out.println("Division by zero is not allowed"); }
             System.out.println("After catch statement.");
    }
}
```

Result

```
C:\Users\Admin\JavaPrograms>java ExDemo1 Division by zero is not allowed
After catch statement
```

Exception granularity:

We can catch exceptions at various levels of granularity.

Handle them in a generic way or in a more specific way.

The Concept of Scope

We can create multiple objects of same class.

Each object has unique identity and state.

Each object has its own memory.

We can declare certain attributes o be shared by all objects of same type.

Methods represent the behavior of an object.

Attributes and their values represent the state of an object.

Types of attributes

- Local attributes
- Instance level or Object level attributes
- Class attributes

```
class Box
{
      static int a=10;
      int b;
     Box(int x, int y){a=x; b=y; }
      void test()
      {
         int c;
          System.out.println(c );
      }
    static  void show()
      {
      System.out.println("Inside show:" );
      }

}
```

```
class Demo
{   public static void main(String args[])
    {
        System.out.println(Box.a );//ok
        System.out.println(Box.b );// illegal
        Box b1=new Box(4,6);
        System.out.println(b1.a );//ok
        System.out.println(b1.b );// ok
        System.out.println(b1.c);//ok
        b1.test();
         b1.show();
         Box.test();//illegal
          Box.show();//ok

    }
 }
```

Operator overloading

- Some OO languages allow overloading operators.
- It allows you to change the meaning of operator.

Ex. C++

Ex: Matrix a,b,c;

c=b+c;

Java supports in limited way. Only for String concatenation. It is done by system not the user.

Multiple inheritance

This allows to inherit a class from more than one class. Ex. C++

It is a powerful feature and can solve many problems elegantly.

But it leads to lots of other confusions.

- Name clashes.
- Repeated inheritance.

Name clashes: Ex:C++ resolves the name clash by attaching the class-name qualifier.

Repeated inheritance:

This occurs when two or more peer superclasses share a common superclass.

In such situation, the inheritance lattice will be a diamond, and so the question arises, does the leaf class have one copy or multiple copies of the structure of the shared superclass.

Most of the cases, multiple inheritance is misused.

Hence it is prohibited by some languages. Ex.Java

Object operations

Deep and shallow copies of objects.

Ex. Cloning in Java.

Summary

- Constructors- default, user defined, overloaded
- Method overloading
- Error handling- catching and handling errors
- Scope- local, object and class attributes
- Multiple inheritance- issues
- Operator overloading
- Object operations- deep and shallow copying