Programming in Java

Object, Classes, Methods and Constructors

Object and classes in java

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

An object is a real-world entity.

An object is a runtime entity.

The object is an entity which has state and behavior.

The object is an instance of a class.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

Fields

Methods

Constructors

Blocks

Nested class and interface

A class is also a data type. You can use it to declare object reference variables. An object reference variable that appears to hold an object actually contains a reference to that object. Strictly speaking, an object reference variable and an object are different, but most of the time the distinction can be ignored.

Syntax to declare a class

```
class <class_name>
{
    field;
    method;
}
```

Class is a keyword in java. <class_name> can be replaced by name of the class we want to give.

Syntax to declare an object using new keyword

Class_name object_name = new class_name();

For example:

rectangle r = new rectangle(); // for single object

rectangle r1 = new rectangle(), r2= new rectangle(); // for two objects

- In java object is just a reference variable for accessing members of the class.
- An object is an instance of a class. You use the new operator to create an object, and the dot operator (.) to access members of that object through its reference variable.

Object and Class Example: main within the class

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 int id;
 String name;
 public static void main(String args[])
{
   Student s1=new Student();//creating an object of Student
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Output

```
0
```

Object and Class Example: main outside the class

```
class Student{
 int id;
 String name;
}
//Creating another class Example which contains the main method
class Example{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Now your file name should be same as that of class name in which main method is defined.

Method in Java

Making programs modular and reusable is one of the central goals in software engineering. Java provides many powerful constructs that help to achieve this goal. Methods are one such construct.In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

Code Reusability

We will discuss methods in more detail after few slides

Ways to initialize object

There are 3 ways to initialize object in Java.

By reference variable

By method

By constructor

By reference variable

```
class Student{
 int id;
 String name;
}
class Example
{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name=“rohit";
  System.out.println(s1.id+" "+s1.name);
 }
}
```

Initialization through method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n)
{
  rollno=r;
  name=n;
 }
 void displayInformation()
{
System.out.println(rollno+" "+name);
}
}
```

```
class TestStudent4
{
public static void main(String args[])
{
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Initialization through a constructor

We will discuss the constructor seperately.

Methods

Method

- A method is a construct for grouping statements together to perform a function.
- A method that returns a value is called a value retuning method, and the method that does not return a value is called void method.
- In some other languages, methods are referred to as procedures or functions.
- A method which does not return any value is called a procedure.
- A method which returns some value is called a function.

Defining a Method

Syntax:

```
modifier returnType methodName (list of parameters)
  {
        // Body of the method(set of statements);
  }
```

Method Signature

Modifier can be static, public, private or protected.

Example:

```
int sum(int a, int b)
{
      ...
}
```

Method Header

Method Body

- Method header specifies the modifier, return type, method name and parameters of method.

```
public void display()
   {…}
```

- The variables defined in method header are called formal parameters or parameters.

```
void display(int x, int y)
  {…}
```

- When a method is invoked, a value as a parameter is passed which is known as actual parameters or arguments.

```
a.display (3, 5);
```

- Method body contains a set of statements that define the function to be performed by the method.

- A return statement using the keyword *return* is required for a value-returning method to return a result.

Calling a Method

- To use a method, we need to *call* or *invoke* it.
- There are two ways to call a method:
  - If the method returns a value, a call to method is usually treated as a value.

    int area = **rectangleArea** (4,6);

    System.out.println(**rectangleArea** (4,6));

  - If the method returns void, a call to method must be a statement.
  - For example, println method returns void

    System.out.**println**("Hello…");

```
// Defining method with in the class in which main method is defined.

class Example
{
        int max (int a,int b)
        {
                return (a>b?a:b);
        }
        public static void main(String[] args)
        {
        Example ob = new Example();
         System.out.println("Maximum of 12 and 34 is "+ ob.max(12,34));
        }
}
```

```
// Defining a static method with in the class in which main method is defined.
// When a method is defined as static method with in the same class in which
// main method is defined. We need not to create object to call that method.
class Example
{
        static int max (int a,int b)
        {
                return (a>b?a:b);
        }
        public static void main(String[] args)
        {
        System.out.println("Maximum of 12 and 34 is "+ max(12,34));
        }
}
```

Example of methods defined in the separate class

```
class Rectangle{
 int length;
 int width;
 void insert(int l, int w)// method
{
  length=l;
  width=w;
 }
 void calculateArea() // method
{
System.out.println(length*width);
}
}
```

```
class TestRectangle1{
 public static void main(String args[])
{
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```

Calling method using anonymous objects

Anonymous objects are the objects that are instantiated but are not stored in a reference variable.

- They are used for immediate method calling.
- They will be destroyed after method calling.

Example

```
class factorial
{

    double calculate(double x)
    {
        double i,f=1;
        for(i=1;i<=x;i++)
        f*=i;
        return f;
    }
}
```

```
class Example
{
    public static void main(String[] args)
    {
  double f= new factorial().calculate(5);
    System.out.println(f);
    }
}
```

Exercise

Write a program to create a class BankAccount having instance variable *balance*.Implement a method deposit(int amt) which receives the amount to be deposited as an argument and adds to the current balance.

Implement another method int withdraw() which asks the user to enter the amount to be withdrawn and updates the balance if having sufficient balance and return the new balance. Invoke both the methods from TestBankAccount class.

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.
However we can implement call by reference method if we pass object of the class rather than primitive data types.

//Example of call by value Method

```
public class Main
{
    void byValue(int x)
    {
        x=x+10;
    }
      public static void main(String[] args) {
            Main obj=new Main();
            int x=12;
            System.out.println("Before method call:"+x);
            obj.byValue(x);
            System.out.println("After method call:"+x);
      }
}
```

```
Before method call:12
After method call:12
```

In java if we pass an object of the class then we can implement call by reference method. In case of call by reference original value is changed if we made changes in the called method.

```
class Example
{
 int data=50;
 void change(Example op)
{
 op.data=op.data+100;//changes will be in the instance variable
 }
     public static void main(String args[])
 {
      Example op=new Example();
    System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}
```

Passing Array to method(1D)

When passing an array to a method, the reference of the array is passed to the method.

```
public class Main
{
   static int method(int arr[])
     {
        int sum=0;
        for(int i=0;i<arr.length;i++)
        {
           sum=sum+arr[i];
        }
        return sum;
     }
      public static void main(String[] args) {
      int a[]={1,2,3,4,5};
      System.out.println("Sum of array elements is:"+method(a));
     }
}
```

Passing anonymous array to a method(1D)[Anonymous array means without creating reference for array]

```
public class Main
{
   static int method(int arr[])
     {
        int sum=0;
        for(int i=0;i<arr.length;i++)
        {
            sum=sum+arr[i];
        }
        return sum;
     }
      public static void main(String[] args) {
      System.out.println("Sum of array elements is:"+method(new int[]{1,2,3,4,5}));
     }
}
```

Passing Array to a method(2D)

```
public class Main
{
   static int method(int arr[][])
     {
        int sum=0;
        for(int i=0;i<arr.length;i++)
        {
           for(int j=0;j<arr[i].length;j++)
           {
           sum=sum+arr[i][j];
           }
        }
        return sum;
     }
      public static void main(String[] args) {
      int a[][]={{1,2},{3,4}};
      System.out.println("Sum of array elements is:"+method(a));
     }
}
```

Returning array from a method[1D]

```
public class Main
{
  static int[] method()
   {
   int a[]={1,2,3,4,5};
   return a;
   }
    public static void main(String[] args) {
    int arr[]=method();
    for(int i=0;i<arr.length;i++)
    {
       System.out.println(arr[i]);
    }
   }
}
```

Returning array from a method[2D]

```
public class Main
{
    static int[][] method()
      {
      int a[][]={{1,2},{3,4}};
      return a;
      }
        public static void main(String[] args) {
        int arr[][]=method();
        for(int i=0;i<arr.length;i++)
        {
            for(int j=0;j<arr[i].length;j++)
            System.out.println(arr[i][j]);
        }
      }
}
```

Using varargs to pass variable number of arguments to a method

```
// Java program to demonstrate varargs
public class Main
{
   // A method that takes variable number of integer arguments.
   static void fun(int ...a)
   {
      System.out.println("Number of arguments: " + a.length);
      // using for each loop to display contents of a
      for (int i: a)
      System.out.print(i + " ");
      System.out.println();
   }
   public static void main(String args[])
   {
      // Calling the varargs method with different number
      // of parameters
      fun(100);        // one parameter
      fun(1, 2, 3, 4);  // four parameters
      fun();           // no parameter
   }
}
```

Output:

```
Number of arguments: 1
100
Number of arguments: 4
1 2 3 4
Number of arguments: 0
```

Method Overloading

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- **<u>Advantage of method overloading</u>**

➢ Method overloading *increases the readability of the program*.

➢ Different ways to overload the method

- <u>There are two ways to overload the method in java</u>

➢ By changing number of arguments

➢ By changing the data type

- **Note: Methods are never overloaded by just changing their return types**

Example 1(Changing no. of arguments)

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}
}
```

Output:

```
22
33
```

Example 2(Changing data type of arguments)

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

```
22
24.9
```

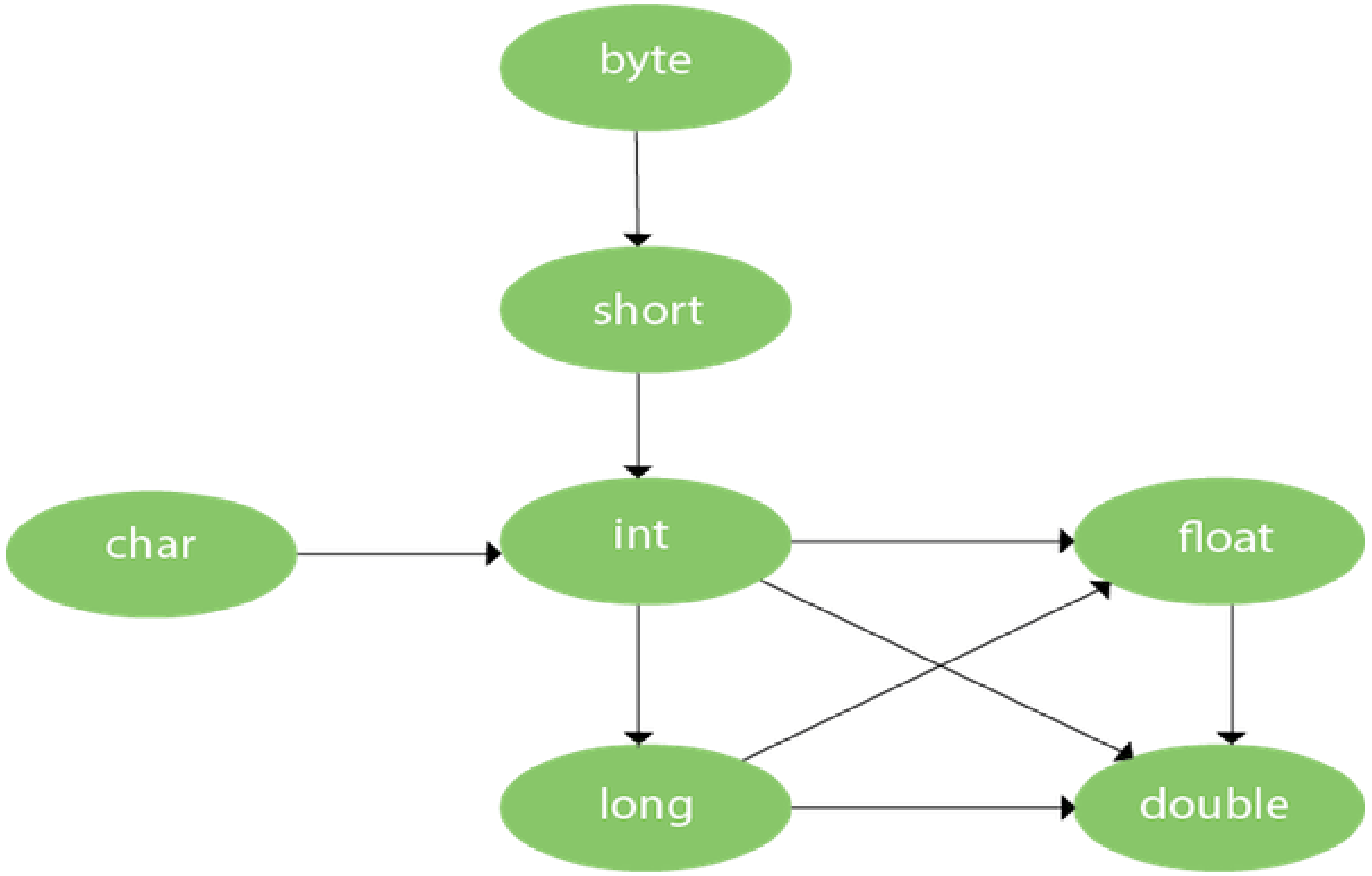
Note: In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

Output:

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

Method Overloading and Type Promotion



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int,long,float or double and so on.

Example-Type Promotion

```
class OverloadingCalculation1{
  void sum(int a,long b){System.out.println(a+b);}
  void sum(int a,int b,int c){System.out.println(a+b+c);}
  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);
  }
}
```

Output:

```
40
60
```

CONSTRUCTOR

Constructors

- A constructor is a special method that is used to initialize a newly created object.
- It is called just after the memory is allocated for the object.
- It can be used to initialize the objects with some defined values or default values at the time of object creation.
- Constructor cannot return values.
- Constructor has the same name as the class name.
- It is not mandatory for the coder to write constructor for the class.

Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- If we do not provide any constructor in the class then default constructor will automatically called to initialize the values of data members.
  - numeric data types are set to 0
  - char data types are set to null character(‘’)
  - reference variables are set to null
  - Boolean are set to false

- In order to create a Constructor observe the following rules:
  - It has the **same name** as the class
  - It should not return a value, not even ***void***

Defining a Constructor

- Like any other method

```
public class ClassName {

    // Data Fields…

    // Constructor
    public ClassName()
    {
        // Method Body Statements initialising Data Fields
    }

    //Methods to manipulate data fields
}
```

- Invoking:
  - There is NO explicit invocation statement needed: When the object is created, the constructor method will be executed automatically.

Constructors

- Constructor name is class name. A constructors must have the same name as the class its in.

- Default constructor. If you don't define a constructor for a class, a default (parameter-less) constructor is automatically created by the compiler.

- The default constructor initializes all instance variables to default value (zero for numeric types, null for object references, and false for booleans).

Key Points

- Default constructor is created only if there are no constructors.

- If you define *any constructor* for your class, no default constructor is automatically created.

- There is *no return type* given in a constructor signature (header).

- There is *no return statement* in the body of the constructor.

Key Points

- The *first line* of a constructor must either be a call on another constructor in the same class (using this), or a call on the super-class constructor (using super).

- If the first line is neither of these, the compiler automatically inserts a call to the parameter-less super class constructor.

Parameterized Constructors

A constructor which has a specific number of parameters is called a parameterized constructor.

```
class rectangle
{
    int l,b;
    rectangle(int x,int y)
    {
        l=x;
        b=y;
    }
    int area()
    {
        return l*b;
    }
}
```

```
class Example
{
public static void main(String[] args) {
      rectangle r1 = new rectangle(12,34), r2= new rectangle(34,56);
      System.out.println(r1.area());
      System.out.println(r2.area());
    }
}
```

Constructor Overloading

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of constructor overloading

```
class sum
{
    sum(int x,int y)
    {
    System.out.println("Sum of 2 integers are "+(x+y));
    }
    sum(int x,int y,int z)
    {
    System.out.println("Sum of 3 integers are "+(x+y+z));
    }
    sum(double x,double y)
    {
    System.out.println("Sum of 2 doubles are "+(x+y));
    }
}
```

```
class Example
{
    public static void main(String[] args) {
        sum s1 = new sum(12,34);
        sum s2 = new sum(12,34,67);
        sum s3 = new sum(12.56,34.78);
    }
}
```

Let’s Do Some thing

Write a program to create a class named Patient which contains:

a. Attributes patient _name, age, contact_number
b. Constructor to initialize all the patient attributes
c. A method to display all the details of any patient

Create a Patient object and display all the details of that patient.

Q1

```
class Test {
  int i;
}
public class Main {
  public static void main(String args[]) {
      Test t = new Test();
      System.out.println(t.i);
   }
}
```

A.-1

B. 0

C. Compiler error

D. Runtime error

Q2

```
class Test {
 int i;
}
pubic class Main {
  public static void main(String args[]) {
    Test t;
    System.out.println(t.i);
}
```

A. 0

B. -1

C. Compiler error

D. Runtime error

Q3

```
public class Main
{
      static void modify(int a)
      {
       a=a+12;
      }
      public static void main(String[] args) {
      int x=12;
      modify(x);
      System.out.println(x);
      }
}
```

A. 24

B. 12

C. 0

D. Compile time error

Q4

```
public class Main
{
   int a,b;
   static void modify(Main obj)
   {
      obj.a=obj.a+2;
      obj.b=obj.b+2;
   }
       public static void main(String[] args)
{
       Main o1=new Main();
       o1.a=13;
       o1.b=15;
       modify(o1);
       System.out.println(o1.a+" "+o1.b);
       }
}
```

A. 13 15

B. 15 17

C. 0   0

D. 1   1

Q5

```
class Dummy
{
    static void show()
    {
        System.out.println("Hello");
    }
}
public class Main
{
        public static void main(String[] args) {
            show();
        }
}
```

A. Hello

B. Compile time error

C. Runtime error

D. Nothing will be displayed