Current Trends in Technology and Science
ISSN: 2279-0535. Volume: 3, Issue: 3

A Mathematical Modeling of Object Oriented Programming Language : a case study on Java programming language

Manoj Kumar Srivastav*, Dr.Asoke Nath
Designation : Student * ,Organization : Indira Gandhi National Open University*,
Email ID* : mksrivastav2011@rediffmail.com,
Designation : Associate Professor, Organization : Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India Email ID : asokejoy1@gmail.com

Abstract **- Srivastav et al already developed method to describe procedural language using mathematical model. As a case study the authors taken C-language as the procedural language. Using simple concept the authors tried to develop some unified mathematical model to explain procedural language. In the present work the authors made an attempt to describe Object Oriented language using simple mathematical model. The class, object defined using simple concept of optimality test of decision function. The inheritance also describe in terms of set concept.**

Keywords **- [class, object, mathematical modeling, object oriented language, Procedural language].**

1. INTRODUCTION

Any mathematical problem depends on the process of its computation. It is mainly depends on the number of steps taken to solve a problem and the time required to solve a problem. A mathematical problem can be solved using various methods/algorithms but we want to find a best possible solution out of all those solutions. One can claim that a problem can have an optimal solution if it is giving an optimal value. The optimal solution may be obtained if the number of steps to solve a problem is optimal and the time taken to solve a problem will be minimum. Now optimal solution and feasible solution can be defined in the context of any Object oriented programming language as follows:

(i) Optimal solution: values of the decision variable which satisfied the condition in lowest effort/cost.

(ii) Feasible solution: all possible solutions that include exception handling such as divide by zero, exceeding array Index upper bound ,format error , data type error etc. These type of exceptional handling generally not possible in procedural language such as in C-language.

All the programming language follows the time and space complexity of a program. The following cases can occur during the running of a program:

Table-1: Space and time requirement

Maximum space	Maximum time
Minimum space	Minimum time
Minimum space	Maximum time
Maximum space	Minimum time
Optimal space	Optimal time

The basic objective of any computer program is to get optimal solution of the given problem. The same problem can be solved in different ways even though the output of the program may exactly be the same. The same problem can be solved using different languages also. So therefore, the main point to be noted here is to get the solution of a problem in minimum time and to solve many problems in some stipulated time to make the solution optimal.

1.1 Introduction to Procedural language:

In procedural language a program comprises of several instructions. A program creates a set of instructions and the computer carries them out. So, a procedural language is divided into functions and each function has clearly defined interface to other function in the program. The idea of breaking a program into functions can be further extended by grouping a number of functions together into a larger entity called a module which is a group of components. Dividing a program into functions and modules is termed as structured programming. If the size of the program grows even larger and more complex then procedural language may produce of the following type of problems.

(a) Reuse of same code is restricted.

(b) Unrelated functions and data provide a poor model of programming.

2. RELATION AMONG TIME, COST AND AREA OF A RELATED PROBLEM

Relation between time, cost and coverage of a related problem in programming language:

Let T =total time for computation of a problem

A=total coverage area of the related problem

C_e =total expenditure cost to prepare a problem

C_v = total valuation cost a problem after preparing it.

2.1Expenditure cost on solving a problem in a programming language follows the following rule:

2.1.1Note: The relationship between Procedural language and Object Oriented language can be further explored as follows:

Procedural Language	Object –Oriented Language	Remarks
$C_e \propto T_1$	$C_e \propto T_2$ with respect to procedural language	(i)when the total coverage area of a problem is constant. (ii) $T_1 >= T_2$ where

		T_1, T_2 corresponds to computation time taken in Procedural and Object oriented programming language respectively.
$C_e \propto A_1$	$C_e \propto A_2$ With respect to	(i)when the total computation time of a problem is constant (ii) $A_1 >= A_2$ where A_1,A_2 corresponds to coverage area taken in Procedural and Object oriented programming language respectively.
$C_e \propto T_1*A_1$ Hence, $C_e =m*T_1*A_1$	$C_e \propto (T_2* A_2)$ Hence $Ce=k*(T_2*A_2)$	(i) When both time and area vary. (ii) Here, m and k are variation constant.

Conculsion: Expenditure cost in the procedural language is >= Object oriented programming language.

2.2Valuation cost of solved problem follows the following rule

Procedural Language	Object – Oriented Language	Remarks
$C_v \propto 1/T_1$	$C_v \propto T_2$	(i)when the total coverage area A_1,A_2 of a procedural language and object oriented language respectively. (ii) T_1,T_2 corresponds to valuation time of program developed using procedural language and object oriented language respectively.
$C_v \propto 1/A_1$	$C_v \propto A_2$	(i)when the total valuation time T_1, T_2 remains constant in procedural language and object oriented language respectively.
		(ii) A_1,A_2 corresponds to total coverage area of procedural language and object oriented language respectively.
$C_v \propto 1/(T_1*A_1)$ Hence, $C_v =p/(T_1*A_1)$	$C_v \propto (T_2*A_2)$ Hence $Cv=r*(T_2*A_2)$	(i)When both time and area vary. (ii)Here, p and r are variation constant.

2.3.Comparision of valuation cost in the programming language with the symbol stated above:
Let T1=T2=T, A1=A2=A

Procedural Language	Object Oriented Programming
Cv=p/(T*A)	Cv=r*(T*A)

Conclusion: From the above table it is clear that valuation cost in procedural language is less than equal to valuation cost in object Oriented Language.
Optimality Test: Let C_{e1} represents Expenditure cost in Procedural Language and C_{e2} represents Expenditure cost in Object Oriented Language.
Let C_{v1} rep represents valuation cost in Procedural Language and C_{v2} represents valuation cost in Object Oriented Language.
Since, $C_{e1}>=C_{e2}$ which implies that $-C_{e1}<=-C_{e2}$
And $C_{v1}<=C_{v2}$
Therefore $(C_{v1}-C_{e1})<= (C_{v2}-C_{e2})$
Hence, we can conclude thatObject Oriented Programming is giving more profit than procedural language.

2.4 Conclusion: We may conclude that we can get an optimal cost from object Oriented programming if the problem becomes larger and better cost in procedural language if the problem is smaller with small time of computation.

3. Mathematical Description Of Object Oriented Programming Language

Object-Oriented programming language is based on the following properties:
(i) Class and Object
(ii)Data Abstraction and Encapsulation
(iii)Inheritance
(iv)Polymorphism
Class is a collection of objects of similar type.

3.1 Class concept in Java Programming Explain with real life Example:

Consider Person as a class .Now we can have some properties associated with this class Person such as

1.Attributes of Person:

(i) Name of Person
(ii) Gender
(iii)Skin color
(iv) Hair color etc.

Now these are general properties which form the template of the class Person,and above properties are called as attributes of the class.

Now Person class may have some functionality such as

1.Talking
2.Walking
3.Eating

Thus in short class have

1.Class Name
2.Properties or Attributes
3.Common function

Now we are going to create object of the class.i.e., actual instance of the class. Let us say that we have created object such as "Aayush "," Aditay". Both will have the same attributes and functionality but have different attribute value:

Name Of Person :----------
Skin color : ----------
Gender :----------

Now this is just a template, called as 'class' and object as instance of the class.

Name of Person : Aayush
Color : white
Gender :Male

Object-P1

Name Of Person : Aditya
Skin color : White
Gender : Male

Object-P2

Set: A set is well- defined collection of distinct objects of our perception or our thought ,to be conceived as a whole.

Set of sets: we have defined a set as a collection of its elements. If the elements be set themselves, then we have a family of sets, or set of sets. For example ,the collection of all subsets of a non-empty set S is a set of sets. This set is said to be the power set of S and is denoted by P(S).

Class may be defined as follows:

In set theory and its application throughout mathematics, a class is collection of sets where each set has some common property. The members of a class is countable.

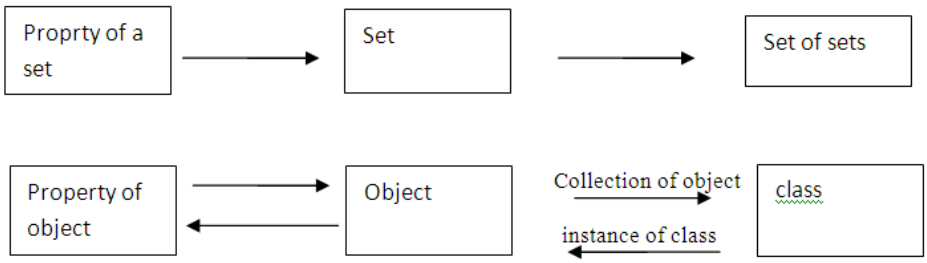
So therefore, class={ sets : where all members have some common property}

The basic form of a class in object –oriented programming is

class classname
{
parameter declaration;
Method declaration;

Object: The members of a class is called an object. Since the members of a class is a set .Therefore it should possesses some property. Objects are real world entity such as pen, chair,
tables,etc.

Similar comparision between set theory and object oriented language :



Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. Any entity that has state and behavior is known as an object. For example: chair, pen, table, etc. It can be physical and logical.

Collection of objects is called class. It is a logical entity.

Any programming problem is analyzed in terms of objects and the nature of communication between them. **An object takes up space in the memory and has an associated address like structure in C.**

Thus class is a set of object and object is a set having some common feature. We can write the class by the symbol φ , ρ , τ ……etc and object by X,Y, Z……. etc.

3.2 MEMORY SPACE :An object is in Java is essentially a block of memory that contains space to store all instance variables. i.e each object of a class contain an address in memory space.

Memory space for an object={address of the instance variables: variables are members of class}

Creating an object is also referred to as instantiating an object. Objects in Java are created using the new

operator.The new operator creates an object of the specified class and returns a reference to that object.Therefore the address of the new object is changed.

Initial object has an address	New object have new address

Therefore we define a mapping $f:(X,\tau) \rightarrow (Y,\sigma)$ such that f(address of object)=address of new object[Here X and Y are object set and are set of sets with respect to object X and σ are set of sets with respect to object Y.Actually τ represents classes τ for domain and σ represents class **for range in the above function and f represents main function.**

Progam1 : Input two numbers a, b. Calculate and print H.C.F of two numbers.

```
import java.io.*;
Class HCF
{
int hcf(int a, int b)
{
int r;
while((r=a%b)!=0)
{
a=b;
b=r;
}
return b;
}
}
class q1
{
public  static void main(String args[])throws IO Exception
{
int a, b,h;
{
Buffered Reader br= new Buffered Reader(new Input Stream
Reader(System.in));
System.out.print("Enter ist number=");
a=Integer.parseInt(br.readLine());
System.out.print("Enter 2nd number=");
b=Integer.parseInt(br.readLine());
HCF H=new HCF();
h=H.hcf(a,b);
System.out.println("\na="+a+b"="+b+" HCF="+h);
}
}
```

Let explain the above program:-

Domain		Range	
object	Class	Object	Class
An object is a software bundle that encapsulates variables and methods operating on those variables. So in domain part here hcf(int a,int b) is taken as method function and a, b ,r are variables.Therefore there is some address for the object variable.	HCF	The name of object is H. The following steps are taken in the range set:- (i) Declaring an object (ii)instantiating an object/initializing an object. So, there is new address for the new object.	Q1

Program2 :-Write a program to input an integer and display whether it is even or odd.

```
import java.io.*;
Class q2
{
public static void main(string args[]) throws IO Exception
{
int n;
BufferedReader br=new BufferedReader (new InputStreamReader(System.in));
System.out.print("\n Enter any integer=");
n=Integer.parseInt(br.readLine());
if((n%2)==0)
System.Out.println(n+"is even");
else
System.Out.println(n+"is odd");
}
}
```

Explaination :-The mapping defined in the above program is in the following way: $f:(X,\tau) \rightarrow (X,\tau)$

i.e we are taking f as an identity mapping.

Program 3: Write a program by creating constructor to show that each object has its own copy of the instance variables of its class.

```
class Student
{
int roll_numb;
int age;
void display()
{
System.out.print(roll_numb);
System.out.println(age);
}
Student(int Rn, int ag)
{
```

```
roll_numb=Rn;
age =ag;
}
public static void main(String arg[])
{
Student  shivam=new Student(1100,4);
Student subham=new Student(1105,2);
shivam.display();
subham.display();
}
}
```

Output

1100 4

1105 2

Explaination :

Domain		Range	
Object	Class	Object	Class
An object is a software bundle that encapsulates variables and methods operating on those variables. So in domain part here void display() and student (int Rn, int ag) are taken as method function and roll_numb, age,Rn,ag as variables.Therefore there is some address for the variables of the given class/object..	Student	Shivam Shubham .Each object has its own copy of the instance variables of its class. Therefore there is new address for the objects Shivam,Shubham	Student

Note:It is important to understand that each object has its own copy of the instance variables of its class.This means that any changes to the variables of one object have no effect on the variables of another. It is also possible to create two or more references to the same object.

3.3 INHERITANCE: When one object acquires all the properties and behaviors of parent object, i.e. known as inheritance. It provide code reusability. It is used to achieve runtime polymorphism. Inheritance aids in the reuse of code.A class can inherit the features of another class and add its modification. The parent class is known as the super class and the newly created child class is known as the subclass.A subclass inherits all the properties and methods of the super class, and can have additional attributes and methods. .

Collection of a class is an object. All the object of a class have unique address and its value is a real number. Since the number of object in a class is finite. Therefore we can prepared a set S having the collection of finite number of address .

Hence S={address of object : object is a member of a class}

Now Class ={collection of object which share some common feature}

Letτ=class={set of objects sharing some common property say p1}

={object :each object has common property say p1}

Let σ=class={set of objects sharing some common property say p2}

={object :each object has common property say p2 which includes property p1 also}

i.e $\sigma = \tau \cup$ {some extra property than the member of τ }

i.e.child class/subclass=parent class/superclass +some extra property

Therefore, We can say that class σ is inheritance of class τ.

If we think the above thing in the sense of memory view for the classτ we have

τ= S={address of object : object is a member of a class}

={a1,a2,a3,……an: ai$\in$ R,set of real numbers}

Since, $\sigma = \tau \cup$ {some extra property than the member of τ }

= {a1,a2,a3,……an: ai ε R,set of real numbers} $\cup$ {b1,b2,b3,……,bn for extra member of object :bi$\in$ R, set of real number}

={a1,a2,a3,……an,b1,b2,b3……..bn}

This property follows by the class is known as inheritance property.

Single level inheritance:

$\sigma = \tau \cup$ {some extra property than the member of τ }

Multilevel inheritance:

$\rho = \sigma \cup$ {some extra property than the member of σ } .

Hierarchical inheritance:

$\sigma = \tau \cup$ {some extra propertyp1 than the member of τ }

$\rho = \tau \cup$ {some extra propertyp2 than the member of τ }

$\varphi = \tau \cup$ {some extra property pn than the member of τ }

Multiple inheritance:

$\sigma = \tau_1 \cup$ {some extra property than the member of τ_1 }

$\sigma = \tau_2 \cup$ {some extra property than the member of τ_2 }

But this type of case will not follow directly because logically here left hand side is same .So the right hand side should be identical

Note :Java does not support multiple inheritance amongst classes. It can still be achieved with the help of interface..

Program: //Write a program to show inheritance in a class

```
import java.io.*;

interface Sum
{
static final int x=100;
 int sum(int n);
 void display(int s);
}
```

```
interface Product
{
final static float pi=3.1415926F;
int product(int n);
}

class SUMPRODUCT implements Sum,Product
{
int i;
        public int sum(int n)
        {
         int s;
        s=0;
        for(i=1;i<=n;i++)
        s+=i;
        return s;
        }
        public int product(int n)
        {
        int p;
        p=1;
        for(i=1;i<=n;i++)
        p*=i;
        return p;
        }
        public void display(int s)
        {
        System.out.print(s);
        }
 }

  class SUM1 extends SUMPRODUCT
  {
        int sum_odd(int n)
        {
        int s,i;
        int x=999;
        float pi=1.234F;
        s=0;
        for(i=1;i<=n;i=i+2)
        s+=i;
        System.out.print("x="+x);
        System.out.print("\nValue of Pi="+pi);
        return s;
        }
  }
  class interface1
  {
        public static void main(String args[])throws
IOException
        {
 BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
        int n,s,p,sodd;
        System.out.print("\nEnter n:");
        n=Integer.parseInt(br.readLine());
        SUM1 SP=new SUM1();
        s=SP.sum(n);
        p=SP.product(n);
        sodd=SP.sum_odd(n);
        System.out.print("\nSum=");
        SP.display(s);
        System.out.print("\nProduct=");
        SP.display(p);
        System.out.print("\nSum od odd numbers=");
        SP.display(sodd);

}
    }
```

class SUMPRODUCT implements Sum,Product	Mutiple inheritance
class SUM1 extends SUMPRODUCT	Single level inheritance

Method Overriding: {f: function f override from the superclass to subclass with same name and type signature}

Example:

```
class A
{
int i=0;
void dooverride(int k)
{
i=k;
}
}                          //subclass definition starts here
 class B extends A
{                          //Method Overriding
 void doOverride(int k)
{
i=2*k;
System.out.println("The value of i is :"+i);
}
public static void main(String args[])
{
B b=new B();   //Create an instance of classB
b.dooverride(12);   //class B  dooverride() will be called
}
}
```

Hence we can get optimal solution of a problem by taking optimal time by making an inheritance on a class.

3.4 ENCAPSULATION

Definition: the ability of an object to hide its data and methods from the rest of the world - one of the fundamental principles of OOP. Because objects encapsulate data and implementation, the user of an object can view the object as a black box that provides services. Instance variables and methods can be added, deleted, or changed, but as long as the services provided by the object remain the same, code that uses the object can continue to use it without being rewritten.

3.5 ABSTRACTION Abstraction refers to the act of representing essential features without including the background details or explanations .

Classes use the concept of abstraction and are defined as a list of abstract attributes.

Hence we can get the optimal solution of a problem from a without source knowledge of an object.

3.6 POLYMORPHISM: When one task is performed by different ways i.e.known as polymorphism. Polymorphism is the ability of an object to take on many forms. In programming languages polymorphism is the capability of an action or method to do different things based on the object that it is acting upon. This is the third basic principle of object oriented programming. The three types of polymorphism are: ad-hoc (overloading and overriding), parametric (generics) and dynamic method binding.

3.6.1 Method Overloading: Method Overloading is one way of achieving polymorphism in Java. Each method in a class is uniquely identified by its name and parameter list. What it means is that we can have two or more methods with the same name, but each with different parameter list. This is a powerful feature of the language called method overloading. Overloading allows us to perform the same action on different types of inputs. In Java whenever a method is being called, first the name of the method is matched and then , the number and types of arguments passed to the method are matched. In method overloading, two methods can have the same name but different signature, i.e. different numbers or type of parameters.
Advantages: The overloading concept is advantageous where similar activities are to be performed but with different input parameters
Program:

```
class overloadDemo
{
void max(float a,float b)
{
System.out.println(“max method with float argument invoked”);
If(a>b)
System.out.println(a+”is greater”);
else
System.out.println(b+”is greater”);
}
void max(double a,double b)
{
System.out.println(“max method with doublet argument invoked”);
If(a>b)
System.out.println(a+”is greater”);
else
System.out.println(b+”is greater”);
}
void max(long a,long b)
{
System.out.println(“max method with long argument invoked”);
If(a>b)
System.out.println(a+”is greater”);
else
System.out.println(b+”is greater”);
}
public static void main(String args[])
{
Over loadDemo O= new Over loadDemo();
O.max(23L,12L);
O.max(2,3);
O.max(54.0,35f);
O.max(43f,35f);
}
}
```

Output:

```
max method with long argument invoked
23 is greater
max method with long argument invoked
3 is greater
max method with double argument invoked
54.0 is greater
max method with float argument invoked
43.0 is greater
```

ThusOverloaded methods are methods with the same name signature but either a different number of parameters or different types in the parameter list.
In mathematical way overloading can be defined in the following way :-
S={collection of function:function have either a different number of parameters or different types in the parameter list}.
={f ε R: f have either a different number of parameters or different types in the parameter list}
={f ε R : same function f can be defined on any dimension in the Eucliedian space}
R is set of real number.
Overridden methods are methods that are redefined within an **inherited or subclass**. They have the **same** signature and the subclass definition is used.
Parametrics are **generic typing** procedures.
Dynamic (or late) method binding is the ability of a program to resolve references to subclass methods at **runtime**. For example assume that three subclasses (Cow, Dog and Snake) have been created based on the Animal abstract class, each having their own speak() method. Although each method reference is to an Animal (but no animal objects exist), the program is will resolve the correct method reference at runtime.

```
public class AnimalReference
{
 public static void main(String args[])
 Animal ref                // set up var for an Animal

Cow aCow = new Cow("Bossy"); // makes specific objects
 Dog aDog = new Dog("Rover");
 Snake aSnake = new Snake("Ernie");
 // now reference each as an Animal
 ref = aCow; ref.speak();
 ref = aDog; ref.speak();
 ref = aSnake; ref.speak();
}
```

4. Summary of Object-Oriented Concepts

I. Everything is an object.
II. Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for action, bundled with whatever arguments may be necessary to complete the tasks.
III. Each object has its own memory, which consists of other objects.
IV. Every object is an instance of a class. A class simply represents a grouping of similar objects, such as Integers or lists.
V. The class is the repository for behaviour associated with an object. That is, that all objects that are instances of the same class can perform the same actions.
VI. Classes are organized into a singly rooted tree structure, called the inheritance hierarchy. Memory and behaviour associated with instances of a class are automatically available to any class associated with a descendant in this tree structure
VII. We can get optimal solution for same type related problem

References

[1] Mathematical modeling of various statements of C-type Language, Manoj Kumar Srivastav , Asoke Nath, International Journal of Advanced Computer
[2] Research(IJACR), Vol-3,Number-1, Issue-13, Page:79-87 Dec(2013).
[2]Mathematical Description of variables, pointers, structures, Unions used in C-type language, Manoj Kumar Srivastava, Asoke Nath, Joiurnal of Global Research Computer Science, Vol-5, No-2, Page:24-29, Feb(2014)
[3] E Balaguruswamy- Programming with Java- TataMcGrawHill EductionP rivate Limited.,2011
[4] Sachin Malhotra, Sourabh Choudhary- Programming in Java- OXFORD University Press,2012
[5] Herbert Schildt, Java ™ 2: The Complete Reference, TataMcGraw-Hill Publishing Company Limited,2001
[6] Satish Jain,Vineeta Pillai,Kratika, Introduction to Object Oriented Programming through Java,BPB Publications,2011
[7] S.K.Mapa, Real Analysis , Asoke Prakasan 1998
[8] S.K.Mapa Higher Algebra, -Sarat Book Distribution,2000
Website reference:
[9] www.c4learn.com
[10] www.ctp.bilkent.edu.tr

Author's Profile

Dr. Asoke Nath is an Associate Professor in the Department Department of Computer Science. Presently he is busy with Cryptography and Network Security Steganography, Green Computing, e-learning, Mathematical Formulation of computer Language.

Mr. Manoj Kumar Srivastav has been Post graduate in Pure Mathematics from University of Calcutta in year 2004 with Special paper in Advanced functional analysis and category theory, universal algebra and lattice theory. At present he is doing MCA from IGNOU and he is working as a postgraduate teacher in Mathematics in an esteemed institution.