CORE JAVA

(As per the New Syllabus 2017-18 of Mumbai University for S.Y.B.Sc. (Computer Science), Semester III)

Prof. Kiran Gurbani
B.E., MCA, M.Phil.,
Head of Computer Science and Information Technology Department,
R.K. Talreja College of Arts, Science and Commerce,
Ulhasnagar (West).

Dr. Ashwin Mehta
M.Sc., Ph.D.,
Ismail Yusuf College,
Jogeshwari.

Prof. Sandeep Vishwakarma
B.Sc., MCA,
B.Sc. (IT) Co-ordinator
Chandrabhan Sharma College of Arts, Science and Commerce, Powai,
Mumbai.

Himalaya Publishing House
ISO 9001:2008 CERTIFIED
Acknowledgements

“Behind every young child who believe in herself is a parent who believes first.” So, I would like to dedicate this book to my mother and father Kavita S. Bajaj and Sahijram Bajaj. I would like to thank my son Chirag Gurbani for being my stress buster and a motivating force for putting a better performance every time.

I am also thankful to all my friends and colleagues; with their support only, this book is possible.

My sincerest big thanks for a lifetime to Mr. S.K. Srivastava of Himalaya Publishing House Pvt. Ltd., for giving me best writing advice and motivation and whose belief in this book has kept me motivated, inspired and encouraged on even the darkest of days.

Kiran Gurbani

Dedicated to my wife Bhavna, my daughter Binkal Lalchandani and my son Kunjal.

I wish to remember my parents who always encouraged me to ‘Study’.

No book can be completed without support of excellent work of publishers. My sincere thanks to Mr. S.K. Srivastava and Ms. Archana to complete work on schedule.

Ashwin Mehta

I would like to dedicate this book to my mother and father Mrs. Savitri Devi and (Late) Shri Jai Prakash Vishwakarma. I would like to thank my wife Mrs. Archana for motivating me to write the book. I would like to thank all my family members for their support.

I would like to thank Mr. S.K. Srivastava (Sales Manager, Himalaya Publishing House Pvt. Ltd.).

I am also grateful to my college principal Dr. Chitra Natarajan for motivation and encouragement.

I would like to thank my vice principal Mrs. Pratima Singh for her constant support and guidance.

Special thanks to all my colleagues from Chandrabhan Sharma college who helped me while writing this book.

Sandeep Vishwakarma
PREFACE

It gives us immense pleasure to present this First Edition of “Core JAVA” to the teachers and students of Bachelor of Computer Science, Second Year, Third Semester. This book has been written as per the syllabus prescribed by the University with effect from academic year 2017-18.

When Java appeared in 1995, immediately its potential as a language for beginning programmers it was noticed. Java satisfies today’s need for early instruction in an object-oriented language, while avoiding the complexities of C++. At the same time, Java is similar enough to C++ to serve as a stepping stone to that language. (At my institution, Java is the first language introduced, immediately followed by C++.) With no pointers to cause problems, Java programs are immune to those frustrating crashes that are so common in C++. GUI of Java is introduced in this book by introducing Applets and AWT.

Chapter 1 describes features of Java, if, for statements, tokens, arrays, strings and string buffer reader, Chapter 2 Inheritance and Interfaces, Chapter 3 Strings, Chapter 4 Packages, Chapter 5 Exception Handling, Chapter 6 Multithreading, Chapter 7 I/O streams and File stream, Chapter 8 Networking, Chapter 9 Wrapper Classes, Chapter 10 Collection Framework, Chapter 11 Inner Classes, Chapter 12 Applets and Chapter 13 AWT.

The concept and theory of each topic is followed by the theoretical explanation and some self-assessment questions, all practical solutions of java is provided.

We have tried our level best to give everything in this book in proper manner. Still few points from book may be explained in better way than presently are there in the book. We hope that this edition will meet all the requirements of S.Y.B.Sc CS and IT students in their regular study and examination preparation.

Constructive suggestions and comments from the users will be sincerely appreciated. We would appreciate hearing from you about any textual errors or omissions which you identify. If you would like to suggest improvements or to contribute in any way, we would be glad to hear from you.

Please send correspondence to kiranktcollege@gmail.com, mehtaashwin2007@yahoo.com and sandeepvcbs@gmail.com.

Last but not the least, we would like to give big lifetime thanks to Mr. S.K. Srivastava (Mob. 7498213322), Himalaya Publishing House Pvt. Ltd. for providing an environment which stimulates new thinking and innovations and his support, motivation, guidance, cooperation and encouragement to write this book. We are very grateful for his support and thank him for his true blessings.

Kiran Gurbani
kiranktcollege@gmail.com 9637128628/7769979964

Ashwin Mehta
mehtaashwin2007@yahoo.com 9820681349

Sandeep Vishwakarma
sandeepvcbs@gmail.com 9819543049
## SYLLABUS

**Course:** USCS302  
**TOPICS (Credits: 02 Lectures/Week: 03)**  
*Core Java*

**Objectives:**
The objective of this course is to teach the learner how to use Object Oriented paradigm to develop code and understand the concepts of Core Java and to cover up with the pre-requisites of Core Java.

**Expected Learning Outcomes:**
1. Object oriented programming concepts using Java.
2. Knowledge of input, its processing and getting suitable output.
3. Understand, design, implement and evaluate classes and applets.
4. Knowledge and implementation of AWT package.

<table>
<thead>
<tr>
<th>Details</th>
<th>Lectures</th>
</tr>
</thead>
</table>
| **Unit I**
The **Java Language:** Features of Java, Java Programming Format, Java Tokens, Java Statements, Java Data Types, Typecasting, Arrays  
**OOPS:** Introduction, Class, Object, Static Keywords, Constructors, this Keyword, Inheritance, Super Keyword, Polymorphism (Overloading and Overriding), Abstraction, Encapsulation, Abstract Classes, Interfaces  
**String Manipulations:** String, String Buffer, StringTokenizer  
**Packages:** Introduction to Pre-defined Packages (java.lang, java.util, java.io, java.sql, java.swing), User Defined Packages, Access Specifiers | 15L |
| **Unit II**
**Exception Handling:** Introduction, Pre-defined Exceptions, Try-Catch-Finally, Throws, Throw, User Defined Exception Examples  
**Multithreading:** Thread Creations, Thread Life Cycle, Life Cycle Methods, Synchronization, wait(), notify(), notifyAll() methods  
**I/O Streams:** Introduction, Byte-oriented Streams, Character-oriented Streams, File, Random Access File, Serialization  
**Networking:** Introduction, Socket, Server socket, Client Server Communication | 15L |
| **Unit III**
**Wrapper Classes:** Introduction, Byte, Short, Integer, Long, Float, Double, Character, Boolean Classes  
**Collection Framework:** Introduction, Util Package Interfaces, List, Set, Map, List Interface and its Classes, Set Interface and its Classes, Map Interface and its Classes  
**Inner Classes:** Introduction, Member Inner Class, Static Inner Class, Local Inner Class, Anonymous Inner Class  
**AWT:** Introduction, Components, Event-Delegation-Model, Listeners, Layouts, Individual Components Label, Button, CheckBox, Radio Button, Choice, List, Menu, Text Field, Text Area | 15L |
# List of Practicals

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accept integer values for a, b and c which are coefficients of quadratic equation. Find the solution of quadratic equation.</td>
</tr>
<tr>
<td>2</td>
<td>Accept two ( n \times m ) matrices. Write a Java program to find addition of these matrices.</td>
</tr>
<tr>
<td>3</td>
<td>Accept ( n ) strings. Sort names in ascending order.</td>
</tr>
<tr>
<td>4</td>
<td>Create a package: Animals. In package animals, create interface Animal with suitable behaviors. Implement the interface Animal in the same package animals.</td>
</tr>
<tr>
<td>5</td>
<td>Demonstrate Java inheritance using extends keyword.</td>
</tr>
<tr>
<td>6</td>
<td>Demonstrate method overloading and method overriding in Java.</td>
</tr>
<tr>
<td>7</td>
<td>Demonstrate creating your own exception in Java.</td>
</tr>
<tr>
<td>8</td>
<td>Using various swing components design Java application to accept a student’s resume (Design form).</td>
</tr>
<tr>
<td>9</td>
<td>Write a Java List example and demonstrate methods of Java List Interface.</td>
</tr>
<tr>
<td>10</td>
<td>Design simple calculator GUI application using AWT components.</td>
</tr>
<tr>
<td>Unit</td>
<td>Particulars</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>UNIT I</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Java Language</td>
</tr>
<tr>
<td>2.</td>
<td>OOPS</td>
</tr>
<tr>
<td>3.</td>
<td>String Manipulations</td>
</tr>
<tr>
<td>4.</td>
<td>Packages</td>
</tr>
<tr>
<td>UNIT II</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Exception Handling</td>
</tr>
<tr>
<td>6.</td>
<td>Multithreading</td>
</tr>
<tr>
<td>7.</td>
<td>IO Streams</td>
</tr>
<tr>
<td>UNIT III</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Wrapper Classes</td>
</tr>
<tr>
<td>10.</td>
<td>Collection Framework</td>
</tr>
<tr>
<td>11.</td>
<td>Inner Classes</td>
</tr>
<tr>
<td>12.</td>
<td>Applets</td>
</tr>
<tr>
<td>13.</td>
<td>AWT</td>
</tr>
<tr>
<td></td>
<td>Final java Practical</td>
</tr>
</tbody>
</table>
Chapter 2

OOPS

“Constructor is a special method used to create and initialization of object with memory allocation”

STRUCTURE

2.1 Introduction to OOPS
2.2 Class
2.3 Object and accessing class members
2.4 Static Keywords
2.5 Constructors
2.6 this Keyword
2.7 Inheritance
2.8 Super Keyword
2.9 Polymorphism
  2.9.1 Compile time polymorphism
  2.9.2 Run time polymorphism
2.10 Abstraction
2.11 Encapsulation
2.12 Abstract Classes
2.13 Interfaces

1.1 Introduction to OOPS

Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.
OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

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

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.
Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don’t know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

1. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
2. OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
3. OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real world problem if we are using the Object-Oriented Programming language.

Java Naming conventions

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

<table>
<thead>
<tr>
<th>Name</th>
<th>Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>class name</td>
<td>should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.</td>
</tr>
</tbody>
</table>
**CamelCase in Java naming conventions**

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

### 2.2 CLASS

**Specifying (Defining) a Class:**

1. A class is a predefined blank template which can be reused number of times.
2. A class is a way to bind the data and its associated functions encapsulated in one unit.
3. It allows the data hiding of data and its functions.
4. When defining a class, we are creating a new abstract data type, that can be treated like any other built-in-data type.
5. A class can be reused many number of times by crating object of it.
6. Objects contains data encode to manipulate the data and the class contain different data members and method s.
7. Wrapping of data and functions in one unit is called class.
8. Class specification has two parts:
   (a) Class declaration.
   (b) Class function definition.

(a) **Class declaration**

**Syntax:**

```java
Class class-name
{
    Private:
    variable declarations;
    function declarations;
    
    Public:
    variable declarations;
    function declarations;
}
```

Class declaration is similar to the structure declaration. The keyword class specifies that what follows is an abstract data of type class-name.
The body of a class is enclosed within braces and terminated by a semicolon. The class body contains the declaration of variables and functions.

These variables and functions are collectively called class members.
These variables and functions are grouped under two sections:
(1) Private (2) Public if needed then protected.

The private and public are known as Scope Specifier or Visibility labels OR Access Specifiers.

These two visibility modes are followed by colon sign (:).
The class members which are declared as private can be accessed only from within the class.
Public members can be accessed from outside the class also.

The no visibility mode is defined within a class, then all class members are by default private.
Such a class is completely hidden form outside world.

The variables declared inside the class are known as data members. They are used to serve the data being a member of a class. Functions declared inside the class are called methods. They are used to perform specific function or task being a member of a class.

Example:

```
Class Product
{
    Private:
        int productid;
        float price;
        void stock();
    Public:
        float amount;
        void getitem();
        void calculate();
        void display();
}
```
In this data member’s productid and price can be accessed by stock, getitem, calculate and display methods.

i.e. private data members can be accessed by private methods and public methods functions of a class and data member amount in public can be used outside the class.

i.e. public variables (properties) can be used by methods of a class as well as functions outside the class.

2.3 Object and Accessing class members

Object is the physical as well as logical entity whereas class is the logical entity only.

Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:
- **state**: represents data (value) of an object.
- **behavior**: represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **oidentity**: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

**Object is an instance of a class.** Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

**Object Definitions:**
- Object is a *real world entity*.
- Object is a *run time entity*.
- Object is *an entity which has state and behavior*.
- Object is an *instance of a class*.
Creating Objects

Object is a instance variable or class variable. Once the class has been declared, we can create object of the class i.e. we can create variables of that type by using the class name.

Example:

product obj; It creates a null object obj of product class.

product is a class, obj is the object or instance of type product, obj is a variable of class type we can declare more than one object of a class.

product x, y, z;

This statement create three null objects x, y, z. The single memory space is allocated to an object.

TO create a object of a class

product obj = new product();

product is a class, obj is object of a class, new operator is used to call a constructor product() to allocate the memory for the object.

If constructor is parameterized the object creation statement is as follows:

product obj = new product(2,3);

Here product is a class, obj is object created when new operator calls the constructor with two parameters 2 and 3.

Accessing Class Members

(1) Private data of the class can not be accessed outside the class. It can be accessed only by the method s of that class.

(2) The public variables and public methods can be accessed outside the class by referring objectname.functionname(list of arguments);

obj.getdata();

obj is the object and getdata() is the method accessed with the help of object. If method is parameterised then while calling parameters are to be passed.

Example:

obj.calculate(2,3,5);

obj is the object, calculate is a method. 2,3,5 are the value of the arguments of the function.

(3) If x is private data member & to access in main function if you have written.

obj.x=10; // It is not allowed

It gives error because private data members cannot be accessed outside the class.

If it is public data member then it can be accessed outside the class.

It x is public variable then we can write.

obj.x=100;

Hence x is assigned as 100.

A class can have both public and private variables and methods. The default is public. private double width; private variables are to be initialized in a constructor; or using a public method.

Constructors and methods can be overloaded for different parameters. Parameters passed to a method are always by value. If an object is passed as parameters it will be by reference.
public class Rectangle
{
    public static void main(String args[])
    {
        System.out.println("how are you");
    }
}

//you can some variables declared and initialized.
public class Rectangle
{
    public static void main(String args[])
    {
        int length, breath=100;
        System.out.println("length=" + length + ", breath=" +breath);
    }
}

//variables can be given in class
public class Rectangle
{
    int length,breath=100;
    public static void main(String args[])
    {
        System.out.println("length=" + length + ", breath=" +breath);
    }
}

//this gives an error you can initialize variable only with declaration or in a function
public class Rectangle
{
    int length=50,breath=100;
    public static void main(String args[])
    {
        System.out.println("length=" + length + ", breath=" +breath);
    }
}

//this again gives an error saying width and height are not static only static variables can be referenced from a static function.
public class Rectangle
{
    static int length=50,breath=100;
    public static void main(String args[])
{  
    System.out.println("length="+length+" breath="+breath);
}

//this gives no error is class variables are to used in main they have to be static or public, if it is public you can access only after instantiating an object of the class

class Rectangle
{
    int length=50,breath=100;
    public static void main(String args[])  
    {  
        Rectangle bd =new Rectangle();
        System.out.println("length="+bd.length+" breath="+bd.breath);
    }
}

// by default variables declared are public, variables of a class should be private and are initialized using a constructor and displayed using a function

class Rectangle
{
    int length,breath;
    public Rectangle()
    {  
        length=50;
        breath=100;
    }
    public void show()
    {  
        System.out.println("length="+length+" breath="+breath);
    }
    public void area()
    {  
        System.out.println("The area of Rectangle is "+(length * breath));
    }
    public static void main(String args[])
    {  
        Rectangle bd =new Rectangle();
        bd.show();
        bd.area();
    }
}
// it is better to give the word public where ever required, if you omit it still default is public so no error occurs.

// constructors will have the class name and no return value, while functions can have any name and should return at least void. Constructors can be overloaded.

class Rectangle
{
    int length, breath;
    public Rectangle()
    {
        length=50;
        breath=100;
    }
    Rectangle (int w, int h)
    {
        length=w;
        breath=h;
    }
    void show()
    {
        System.out.println("length=" + length + ", breadth=" + breath);
    }
    public static void main(String args[])
    {
        Rectangle bd1 = new Rectangle();
        Rectangle bd2 = new Rectangle(200, 200);
        bd1.show();
        bd2.show();
    }
}

// parameters can have the same name as class variables. You have to use the keyword this in order to qualify the class variables to avoid ambiguity.

class Rectangle
{
    int length, breath;
    public Rectangle()
    {
        length=50;
        breath=100;
public class Rectangle
{
    int length,breath;
    public Rectangle()
    {
        length=50;
        breath=100;
    }
    Rectangle(int width,int height)
    {
        this.length=length;
        this.breath=breath;
    }
    void show()
    {
        System.out.println("length=\" + length + ", breath=\" +breath);
    }
    public static void main(String args[])
    {
        Rectangle bd1 =new Rectangle();
        Rectangle bd2 =new Rectangle(200,200);
        bd1.show();
        bd2.show();
    }
}
```java
System.out.println("area=" + length * breath);
System.out.println("perimeter=" + 2*(length * -- * breath));
}
public static void main(String args[])
{
    Rectangle bd1 = new Rectangle();
    Rectangle bd2 = new Rectangle(200, 200);
    bd1.show();
    bd2.show();
    bd1.show(10, 12);
}
}

// classes can be defined outside the class in which main() is specified
class Rect
{
    int length, breath;
    public Rect()
    {
        length = 50;
        breath = 100;
    }
    Box(int width, int height)
    {
        this.length = length;
        this.breath = breath;
    }
    void show()
    {
        System.out.println("area=" + length * breath);
    }
    void show(int length, int breath)
    {
        System.out.println("area=" + length * breath);
        System.out.println("perimeter=" + 2*(length * breath));
    }
}
public class RectDemo
{
    public static void main(String args[])
    {
```
Rect bd1 = new Rect();
Rect bd2 = new Rect(200, 200);
bd1.show();
bd2.show();
bd1.show(10, 12);
}

// if you use the word public it will give an error saying public classes should be in a separate file. By default the methods are package access qualifier, that is you can use within the package or directory.

2.4 Static Keywords

The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:
1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class
1. Java static variable
   - If you declare any variable as static, it is known static variable.
   - The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
   - The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program memory efficient (i.e it saves memory).

Understanding problem without static variable

class Student {
    int rollno;
    String name;
    String college = "ITS";
}

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.
Example of static variable

```java
//Program of static variable
class Student8 {
    int rollno;
    String name;
    static String college = "ITS";
    Student8(int r, String n) {
        rollno = r;
        name = n;
    }
    void display() {
        System.out.println(rollno + "\n" + name + "\n" + college);
    }
    public static void main(String args[]) {
        Student8 s1 = new Student8(111, "Kiran");
        Student8 s2 = new Student8(222, "Sri");
        s1.display();
        s2.display();
    }
}
```

Output: 111 Kiran ITS
        222 Sri ITS

2.5 Constructors

Constructor in java is a special type of method that is used to initialize the object.

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

We have used methods such as getData( ) or input( ) to initialize the variables. When this method is invoked with the help of object, it assigns the initial values to the data items of object.

It is a tedious job to initialize all the variables in the class. We have to use separate method each time when instances created to initialize. We have to call separate method; this initialization can be done automatically is performed through the use of constructor.

The methods cannot be used to initialize member variables at the time of creation of their objects hence to perform initialization of members while creating object is performed by special method in Java called constructor.
Hence special method called the constructor which enables an object to initialize itself when it is created. This is known as automatic initialization of objects. It also provides another method called the destructor that destroys the objects when they are no longer required.

Constructors

**Definition:** A constructor is a special method which enables object to initialize itself when it is created. It is invoked automatically having same name as the name of the class.

**Constructor function has following special characteristics:**

1. Constructor is special method used to initialization of object.
2. Constructor method is having the same name as name of the class.
3. Constructor method is automatically invoked when the object of its associated class is created.
4. They should be declared in public section.
5. They do not have return types, they cannot return any value. Hence they cannot have even void return type.
6. Like other methods the constructor method also can have parameters the constructor function with parameters are called parameterised constructor.
7. The Default constructor which is called to initialize the object is the non-parameterised default constructor.
8. Like other functions, they can have default arguments because the constructors can be parameterised constructors.
9. Constructors supports polymorphism. Constructor can be overloaded like method overloading. Hence one class can have more than one constructor with the different number of arguments.
10. Constructor can be inherited in Java that means base class constructor can be inherited by using super Keyword.
11. The constructor makes implicit calls to the two operations.
   (a) new
   (b) delete
   - When memory allocation is required for the first time.
   - The new operator is called when object is created and it is having memory allocation for the first time.
   - Delete operator is called when object is not used and it is deallocated from memory.
12. The constructor cannot be inherited further, though derived class cannot call the base class constructor.
13. Constructors cannot be virtual.
14. When object is created the default (non-parameterised) constructor is invoked. If the object is without arguments then empty constructor is called.
   - If object is created with parameters, then parameterised constructor is called.
15. It is called constructor because it construct the object and assign data members of the class by initializing some value.
Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

1. `<class_name>() {}`

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1 {
    Bike1() {System.out.println("Bike is created");}
    public static void main(String args[])
    {
        Bike1 b=new Bike1();
    }
}
```

Example:

```
Class square
{
    Double width;
    Double Height;
    Box ()
    {
        System.out.println ("Constructing Box")
        Width = 10;
        Height = 10;
    }
    Double area()
    {
        Return width*height;
    }
Class squaredemo
{
    Public static void main (string args [])
    {
        square mysq = new square();
        Int s = mysq.area ();
    }
```
Parameterised Constructors:

(i) Sometimes we need to initialize the various data elements of different objects with different values when they are created. This can be achieved by passing arguments to the constructor function when object is to be created.

(ii) The object is to be initialized through the constructor by passing arguments to the constructor function when object is created.

(iii) The constructor that takes arguments is called parameterised constructor.

Constructor Overloading:

A java program can have more than one constructor condition that constructor should be parameterized constructor. We can overload the default constructor. Default constructor is without parameter then we can overload default constructor with parameterized constructor.

Example 1:

Write a program to give the demo of addition of two integers, three integers using constructor overloading.

```java
class addition {
    Int x, y, a, b, c;
    Addition ()
    {
        System.out.println("default constructor is called");
    }
    Addition (int x1, int y1)
    {
        x=x1;
        y=y1;
    }
    Addition (int a1, int b1, int c1)
    {
        a=a1;
        b=b1;
        c=c1;
    }
    Int add2 ()
    {
        Return (x+y);
    }
    Int add3 ()
```
Example 2:
Constructors will have the class name and no return value, while functions can have any name and should return at least void. Constructors can be overloaded.
(a) Create a ‘car’ class, Initialize the color and body with blue and wagon respectively.
(b) Create two constructor of which one is a default constructor that creates a blue car and other contains two arguments ‘color’ and ‘body’.
(c) Write a method ‘toString’ that returns the color and body of the car

Program:
```java
class car
{
    String color="blue";
    String body="wagon";

    car()
    {
        color="blue";
    }
    car(String c,String b)
    {
        color=c;
        body=b;
    }
    public String toString()
    {
        return("Color of car is "+color +" body is "+body);
    }
}
```
class cardemo {
    public static void main(String args[]) {
        car c1=new car();
        car c2=new car("red","santro");
        //System.out.println("c1=\"+c1.color + \" +c1.body);  
        //System.out.println("c2=\"+c2.color + \" +c2.body);  
        System.out.println("car c1:\"+c1);  
        System.out.println("car c2:\"+c2);  
    }
}

Output:
C:\javac cardemo.java
C:\java cardemo
Car c1:Color of car is blue body is wagon
Car c2:Color of car is red body is santro

Difference between constructor and method in java
There are many differences between constructors and methods. They are given below.

<table>
<thead>
<tr>
<th>Java Constructor</th>
<th>Java Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor is used to initialize the state of an object.</td>
<td>Method is used to expose behaviour of an object.</td>
</tr>
<tr>
<td>Constructor must not have return type.</td>
<td>Method must have return type.</td>
</tr>
<tr>
<td>Constructor is invoked implicitly.</td>
<td>Method is invoked explicitly.</td>
</tr>
<tr>
<td>The java compiler provides a default constructor if you don't have any constructor.</td>
<td>Method is not provided by compiler in any case.</td>
</tr>
<tr>
<td>Constructor name must be same as the class name.</td>
<td>Method name may or may not be same as class name.</td>
</tr>
</tbody>
</table>

Java Copy Constructor
There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:
- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.
class Student6 {
    int id;
    String name;
    Student6(int i,String n){
        id = i;
        name = n;
    }
    Student6(Student6 s){
        id = s.id;
        name = s.name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
        Student6 s1 = new Student6(111,"Karan");
        Student6 s2 = new Student6(s1);
        s1.display();
        s2.display();
    }
}

Output:
111 Karan
111 Karan

2.6 this Keyword

this keyword indicates object pointer to current class.

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.
1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Suggestion: If you are beginner to java, lookup only three usage of this keyword.
1. *this:* to refer current class instance variable

The `this` keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

**Understanding the problem without this keyword**

```java
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee){
        rollno=rollno;
        name=name;
        fee=fee;
    }
    void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
    public static void main(String args[]){
        Student s1=new Student(111,"ankit",5000f);
        Student s2=new Student(112,"sumit",6000f);
        s1.display();
        s2.display();
    }
}
```

**Output:**

0 null 0.0

0 null 0.0

**Example of single inheritance with this keyword:**

Write a program in java to define the following classes

1. A class called superclass that defines one integer variables & Accept variable value through parameterised constructor with same variable name as declared within class, provide one method to display that number.
2. A class sub inherits superclass , It defines one integer variable, In the sub class constructor, inherit the variables of superclass through super class constructor along with that accept one parameter through derived class. provide display method to display both variables of superclass and sub class.
3. Create simpleinh class with main method to invoke the methods of base class.

```java
class superclass{
    int x;
    superclass(int x)
```
```java
{
    this.x = x;
}

void display()
{
    System.out.println("the value of x is " + x);
}
}
class sub extends superclass
{
    int y;
    sub(int x, int y)
    {
        super(x);
        this.y = y;
    }
    void display()
    {
        System.out.println("the x is " + x);
        System.out.println("the y is " + y);
    }
}
class inhsup
{
    public static void main(String args[])
    {
        sub s = new sub(12, 10);
        s.display();
    }
}
```

**Note:** This program demonstrates the use of this keyword which indicate object pointer to current class.

### 2.7 Inheritance

Inheritance is a concept of reusability. It is also called derivation. Reusability is the important feature of OOP’s. Reusability is a technique where we can reuse the class which is already tested and debugged many times.

This technique is used for creating new class from old class and reusing the properties of existing one.
The mechanism of deriving new class from old class is known as inheritance. The old class is referred as base class and the new class is referred as derived class. The base class is also called as super class and the derived class is also called as sub-class. It is also known as parent-child relationship. The derived class can access properties and methods of base class during any type of inheritance as well as derived class has its own properties and methods.

**Types of Inheritance:**

1. **Single Inheritance:** A derived class with only one base class is called single inheritance. With this one derived class can be possible with one base class.

   ![Single Inheritance Diagram](image1.png)

2. **Multilevel Inheritance:** The mechanism of deriving a class from another derived class is known as Multilevel Inheritance.

   ![Multilevel Inheritance Diagram](image2.png)

   In multilevel first class is derived from base class and again from derived class another class will be derived.

3. **Multiple Inheritance:** A derived class with several base classes, is called multiple inheritance.

   ![Multiple Inheritance Diagram](image3.png)

   This type of inheritance can have one or more base classes from that we can drive only derived class is called as Multiple Inheritance. This type of Inheritance is not directly available in Java, It can be implemented by using interface concept.

4. **Hierarchical Inheritance:** The properties of one class may be inherited by more than one class is called hierarchical Inheritance. In this type of inheritance one base class, number of classes can be derived is known as hierarchical Inheritance.
5. **Hybrid Inheritance:** It is a combination of Multilevel and Multiple Inheritance as well as it can be combination of hierarchical and multiple inheritances hence it is combination of different types of inheritance.

How Inheritance can be Achieved in Java

Base class can be defined as normal Java class which may consist of some data and functions.

Derived class is a class which acquires properties from base class is called derived class.

Derived class can have its own properties and methods.

**Defining a subclass:**

```
Class subclassname extends superclassname
{
    Variable declarations;
    Method definition
}
```

The keyword “extends” signifies the properties of the superclass name. The subclass contain its own properties and methods as well as the properties & methods of super class.

**Single Inheritance Examples:**

(I) Example of single inheritance:

Write a program in java to define the following classes.

1. A class called Addition that defines two integer variables & one method to display the two numbers.

2. A class summation inherits Addition , It defines one integer variable, provide display method to display that variable, Provide sum method to print sum of three integers , two inherited from superclass and one from sub class.

3. Create simpleinh class with main method to invoke the methods of base class.
class Addition
{
    int a,b;
    void displayab()
    {
        System.out.println("a and b are" +a +" + " + b);
    }
}

class Summation extends Addition
{
    int c;
    void displayc()
    {
        System.out.println("c:" +c);
    }
}

void sum()
{
    System.out.println("a+b+c" +(a+b+c));
}

class simpleinh
{
    public static void main(String args[])
    {
        Addition superob = new Addition();
        Summation subob = new Summation();
        superob.a=10;
        superob.b=20;
        System.out.println("contents of super class");
        superob.displayab();
        subob.a=7;
        subob.b=8;
        subob.c=9;
        System.out.println("contents of sub class is");
        subob.displayab();
        subob.displayc();
        System.out.println("sum of a and b and c");
        subob.sum();
    }
}
Multilevel Inheritance Example

File: TestInheritance2.java

class Animal {
    void eat() {System.out.println("eating...");}
}
class Dog extends Animal {
    void bark() {System.out.println("barking...");}
}
class BabyDog extends Dog {
    void weep() {System.out.println("weeping...");}
}
class TestInheritance2 {
    public static void main(String args[]) {
        BabyDog d = new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}

Output:
weeping...
barking...
eating...

Hierarchical Inheritance Example

File: TestInheritance3.java

class Animal {
    void eat() {System.out.println("eating...");}
}
class Dog extends Animal {
    void bark() {System.out.println("barking...");}
}
class Cat extends Animal {
    void meow() {System.out.println("meowing...");}
}
class TestInheritance3 {
    public static void main(String args[]) {
        Cat c = new Cat();
        c.meow();
    }
}
c.eat();
//c.bark();//C.T.Error
}
}

Output:
meowing...
eating...

**Aggregation in Java**

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```java
class Employee{
    int id;
    String name;
    Address address;//Address is a class
    ...
}
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

**Why use Aggregation?**
- For Code Reusability.

**Simple Example of Aggregation**

In this example, we have created the reference of Operation class in the Circle class.

```java
class Operation{
    int square(int n){
        return n*n;
    }
}
class Circle{
    Operation op;//aggregation
    double pi=3.14;

double area(int radius){
    op=new Operation();
    int rsquare=op.square(radius);//code reusability (i.e. delegates the method call).
    return pi*rsquare;
}```
public static void main(String args[]){
    Circle c=new Circle();
    double result=c.area(5);
    System.out.println(result);
}

Output: 78.5

2.8 Super Keyword

Single Inheritance Using Super:

When subclass want to inherit the superclass or base class variables then in Java it can be achieved by using Super Keyword.

Super has two general forms
1. It calls the superclass constructor
2. It is used to access a member of the superclass that has been hidden by a member of a subclass.
3. A subclass can call a constructor method defined by its superclass by use of super(parameters list)
4. The parameters in super should be same as the constructor parameters of superclass constructor.
5. super() must always be the first statement executed inside the subclass constructor.
6. subclass constructor is used to construct the instance variables of both the subclass and superclass.
7. super can only be used within a subclass constructor method.

2.9 Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.
Example
Let us look at an example.

```java
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

Now, the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above examples −

- A Deer IS-A Animal
- A Deer IS-A Vegetarian
- A Deer IS-A Deer
- A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal −

```java
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

All the reference variables d, a, v, o refer to the same Deer object in the heap.

2.9.1 Compile time polymorphism

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.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly

**Advantage of method overloading**
Method overloading increases the readability of the program.

**Different ways to overload the method**
There are two ways to overload the method in java
1. By changing number of arguments
2. By changing the data type
Compile time Polymorphism (METHOD OVERLOADING):

In Java, it is possible to define two or more methods within the same class that share the same name as long as their parameter declarations are different. When this is a case, the method is said to be overloading; this concept is called “Polymorphism.” Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object Java compiler.

Matches the method name first if the method name matches then it matches the no. of parameters. If the number of parameters is also same then it matches data type of parameters, that means same method can be used more than once either with different parameters same number of parameters with different data type.

```
sum (int a, int b)
sum (int x, int y, int z)
sum (float fx, float fy)
```

All three sum function can be used in one program. Through they are having the same name but the parameters are different, either no of parameters are different or data type of parameters is different.

**Example of method overloading:**

```java
Class overload demo
{
    void addition ()
    {
        System.out.println ("test function with no parameters");
    }
    void addition (int a)
    {
        System.out.println ("the value of parameters a is"+a);
    }
    void addition (int a, int b)
    {
        System.out.println ("the value of a is"+a + "the value of b is"+ b);
    }
    double addition (double a)
    {
        System.out.println ("the value of a is"+a);
        Return a*b;
    }
}
Class overload
{
    Public static void main (string args [])
}
```java
{
    overload demo obj = new overload demo ();
    double result;
    obj.addition ();
    obj.addition (10);
    obj.addition (10, 20);
    result = obj.addition (123.3);
    System.out.println ("The result is "+result);
}
```

### 2.9.2 Run time polymorphism

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

**Usage of Java Method Overriding**

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism.

**Rules for Java Method Overriding**

1. method must have same name as in the parent class.
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

**Run Time polymorphism**

```java
class shape{
    int a,b; double area=0.0;
    public void print(){
        System.out.println("area = "+area);
    }
}

class circle extends shape{
    circle(int x){
        a=x;
    }
    public void print(){
        area = Math.PI*a*a;
        super.print();
    }
}
```
class rect extends shape {
    rect(int x, int y) {
        a = x; b = y;
    }
    public void print() {
        area = a * b;
        super.print();
    }
}

class MyApp {
    public static void main(String args[]) {
        shape s1 = new shape();
        circle c1 = new circle(2);
        rect r1 = new rect(2, 3);
        s1.print(); c1.print(); r1.print();
        s1 = c1; s1.print();
        s1 = r1; s1.print();
    }
}

// when s1 is assigned c1 it prints its function. When it is assigned r1 it prints the rectangle area. This is polymorphism.

Benefits of Encapsulation

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.
- The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code.

2.10 Abstraction

Abstraction is the quality of dealing with ideas rather than events. For example, when you consider the case of e-mail, complex details such as what happens as soon as you send an e-mail, the protocol your e-mail server uses are hidden from the user. Therefore, to send an e-mail you just need to type the content, mention the address of the receiver, and click send.

Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

In Java, abstraction is achieved using Abstract classes and interfaces.