Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 1

INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

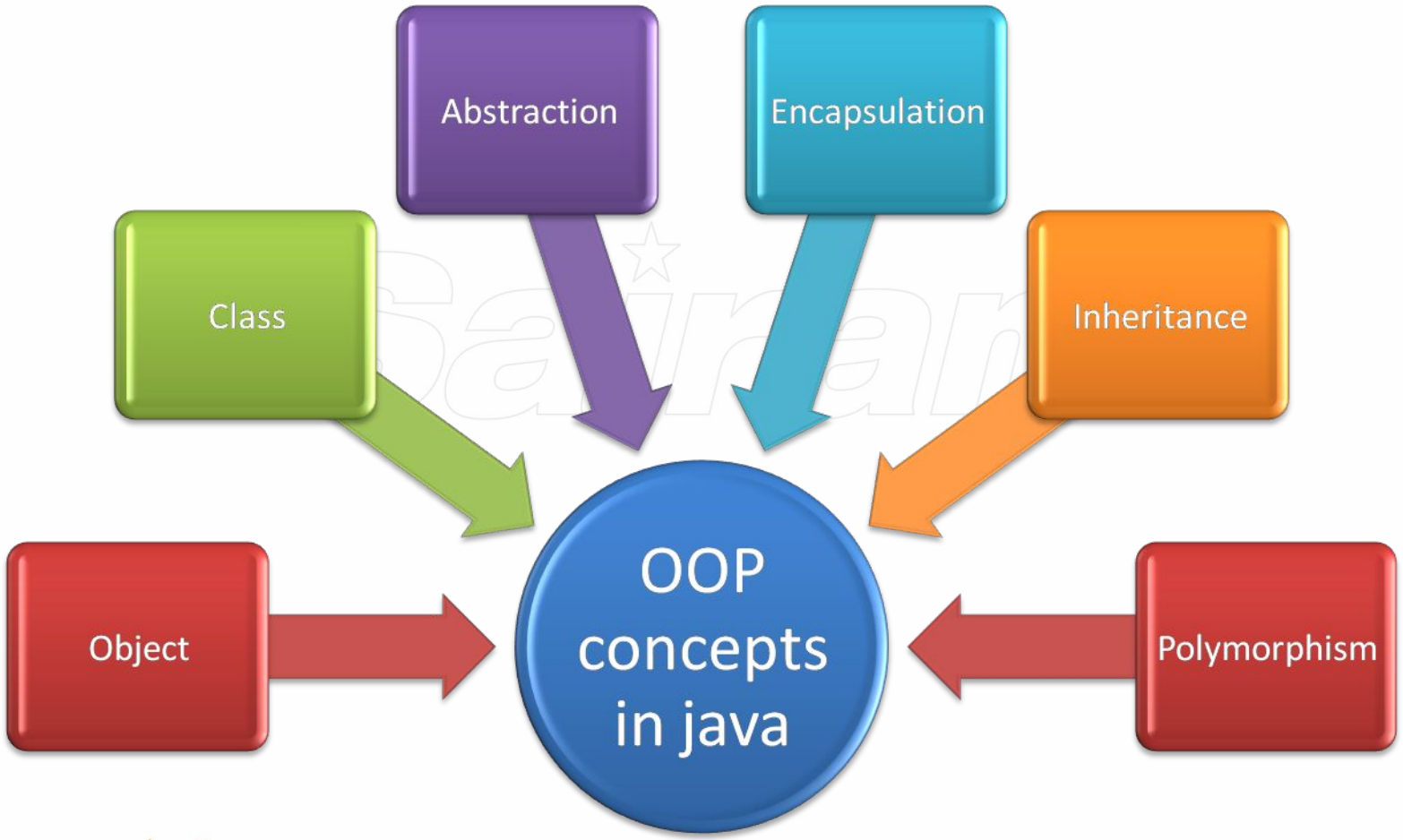
1.2 OOP in Java, Characteristics of Java

COMPUTER SCIENCE & ENGINEERING

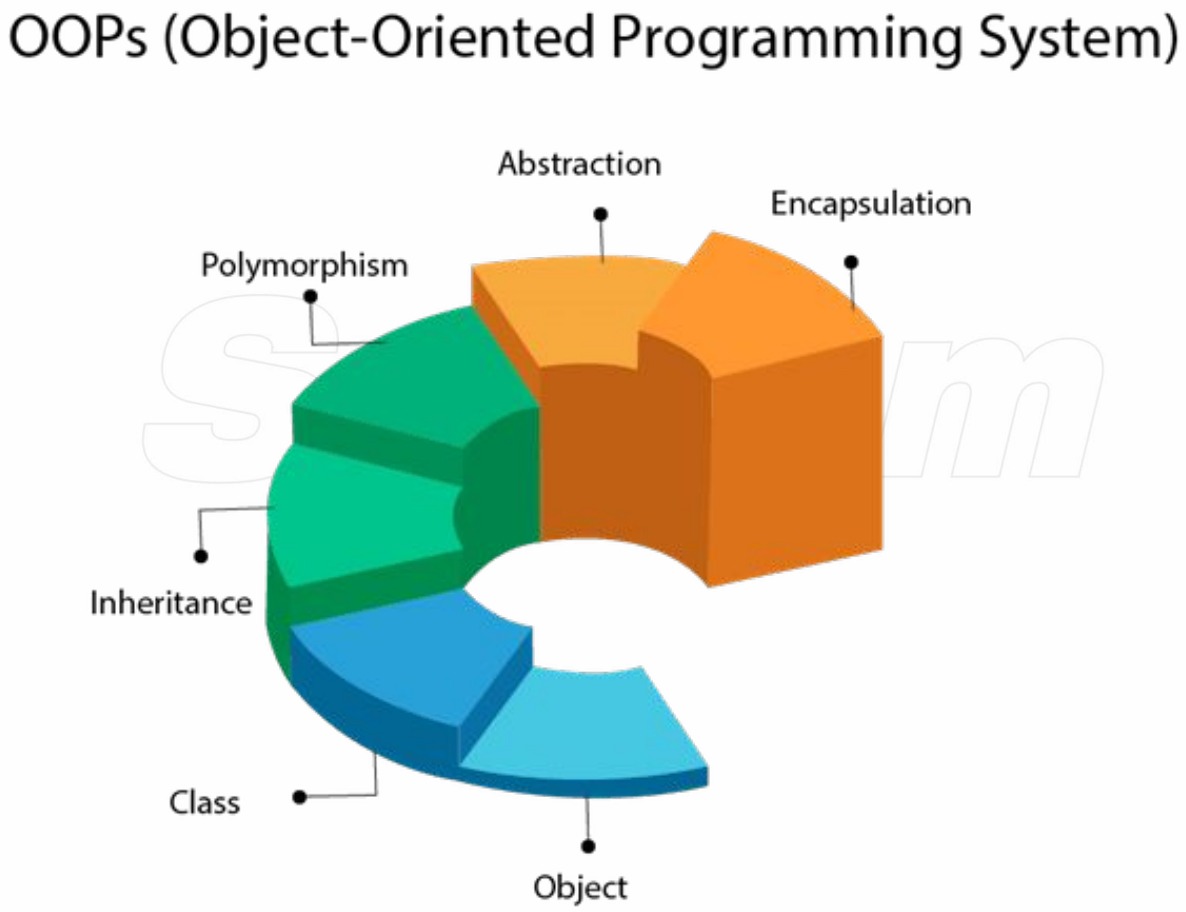
OOP in Java

- Java supports almost all the OOP concepts.
- Object-oriented programming aims to implement real-world entities using object, class, inheritance, hiding, polymorphism in programming
- **OOP concepts in java.**
  - Object
  - Class
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

OOP in Java


Abstraction
Encapsulation
Class
Inheritance
Object
OOP concepts in java
Polymorphism

OOP in Java


OOPs (Object-Oriented Programming System)
Abstraction
Encapsulation
Polymorphism
Inheritance
Class
Object

OOP in Java : object

Object:

- It represents any real world / life entities.
- Object means a real-world entity such as a book, chair, laptop,car,bike etc.
- An Object can be defined as an instance of a class.
- Any entity that has identity, state (properties) and behavior is known as an object.

create objects in Java:

```
className object = new className();
```

Example:

```
cse  s1=new cse(); // assume cse is class name
```

OOP in Java : class

Class:

- Collection of objects is called class.
- A class is a blueprint from which you can create an individual object.
- Class contains attributes and methods that can be accessed by its instances(object).
- Class has the set of properties or methods that are common to all objects of one type.

OOP in Java : class

Syntax to define a class in Java:

```
class ClassName {
   // variables
   // methods
}
```

Example:

```
class Cse
{
String name;
int roll_no;
String dept;
void getdata()
{
// function definition
}
void putdata()
{
// function definition
}
} // end of class
```

OOP in Java : abstraction

Abstraction:

- Hiding internal details and showing functionality is known as abstraction.
- Data Abstraction is the property of which only the essential details are displayed to the user.

Example:

- Using Bank ATM for cash withdrawal, money transfer, retrieve mini-statement…etc. (internal technical info is hidden)
- Man driving a car. He does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc
- Phone call, internal processing is not known
- Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

OOP in Java : abstraction

Abstract Class:

- All abstract methods of an abstract class.
- An abstract class is a class that cannot be instantiated .
- Objects of an abstract class cannot be created.
- Allows creation of subclasses from abstract class.
- A subclass must override

Abstract Method:

- An abstract method is declared without an implementation.
- An abstract method doesn't have any implementation.
- only an abstract class can contain abstract methods.
- It is must override abstract methods of the superclass in the subclass.

OOP in Java : abstraction example

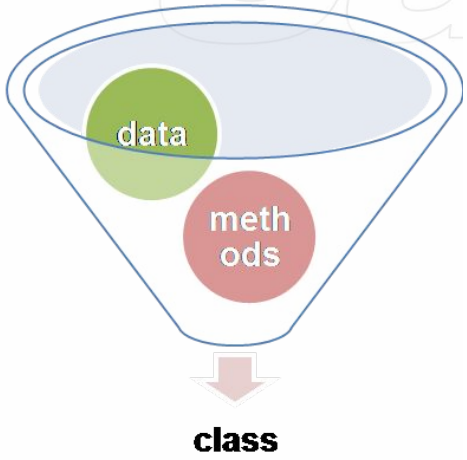
```
// Abstract class
abstract class Interest {
  // Abstract method (does not have a body)
  public abstract void ComputeInterest();
  // Regular method
  public void show() {
    System.out.println("Hai");
  }
}
 // Subclass (inherit from Interest)
class Homeloan extends Interest {
  public void ComputeInterest() {
    // The body of ComputeInterest()
    System.out.println("Homeloan: interest");
  }
}
```

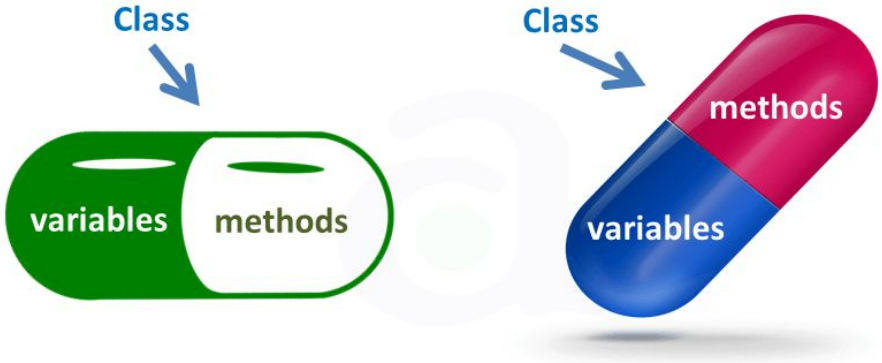
```
// Subclass (inherit from Interest)
class Carloan extends Interest {
  public void ComputeInterest() {
    // The body of ComputeInterest()
    System.out.println("Homeloan: interest");
  }
}
class MainClass {
  public static void main(String[] args) {
    Homeloan myhomeloan = new Homeloan(); //
Create a homeloan object
    myhomeloan.ComputeInterest();
    myhomeloan.show();
Carloan mycarloan = new Carloan(); // Create a
carloan object
    mycarloan.ComputeInterest();
    mycarloan.show();  } }
```

OOP in Java : encapsulation

Encapsulation:

- Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.





Encapsulation

OOP in Java : encapsulation

Example for Encapsulation:

```
class cse
    {
    int id; //field or data member or instance variable
    String name;
    public void show()
    {
    id=5;
    name="SureshAnand";
    System.out.println(id);    System.out.println(name);
    }
 public static void main(String args[])
    {
    cse s1=new cse();
    s1.show();
     }
     }  // end of class cse
```

OOP in Java : inheritance

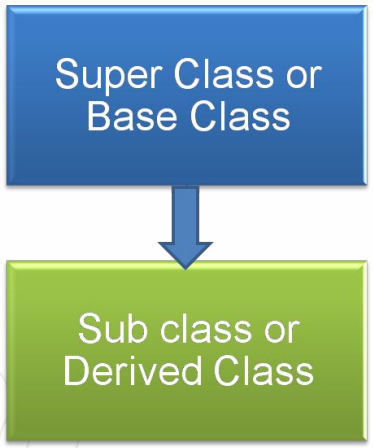
Inheritance:

- An object acquires all the properties and behaviors of a parent object, it is known as inheritance.
- One class is allowed to acquire (inherit) the features (fields and methods) of another class.
- Super Class: The class whose features are inherited is known as superclass (or a base class or a parent class).
- Sub Class: The class that inherits the other class is known as subclass (or a derived class, extended class, or child class).

OOP in Java : inheritance

Syntax for inheritance:

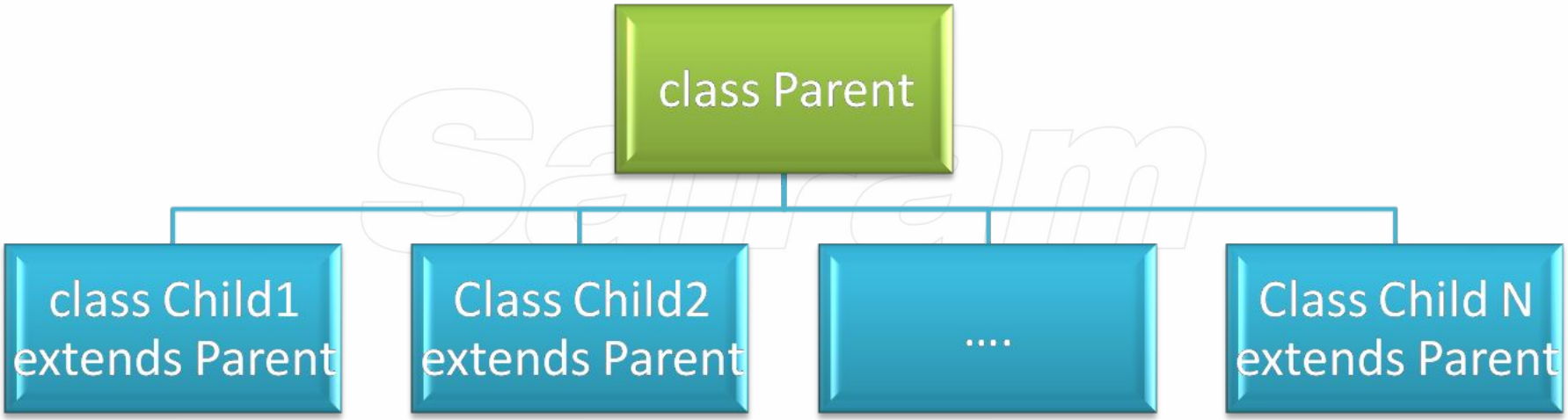
```
class derived-class extends base-class
{
 //methods and fields
}
```



Example :

```
class Parent
{
 //methods and fields
}

class Child extends Parent
{
 //methods and fields  of Parent class acquired here
//methods and fields  Child

}
```

OOP in Java : inheritance



OOP in Java : inheritance example

```
// base class
class Interest
{
public void getdata()
        {
        //get customer details
        }
public abstract void ComputeInterest();
}

// Subclass
class Homeloan extends Interest
{
  public void ComputeInterest()
        {
       System.out.println("Homeloan: interest");
        }
}
```

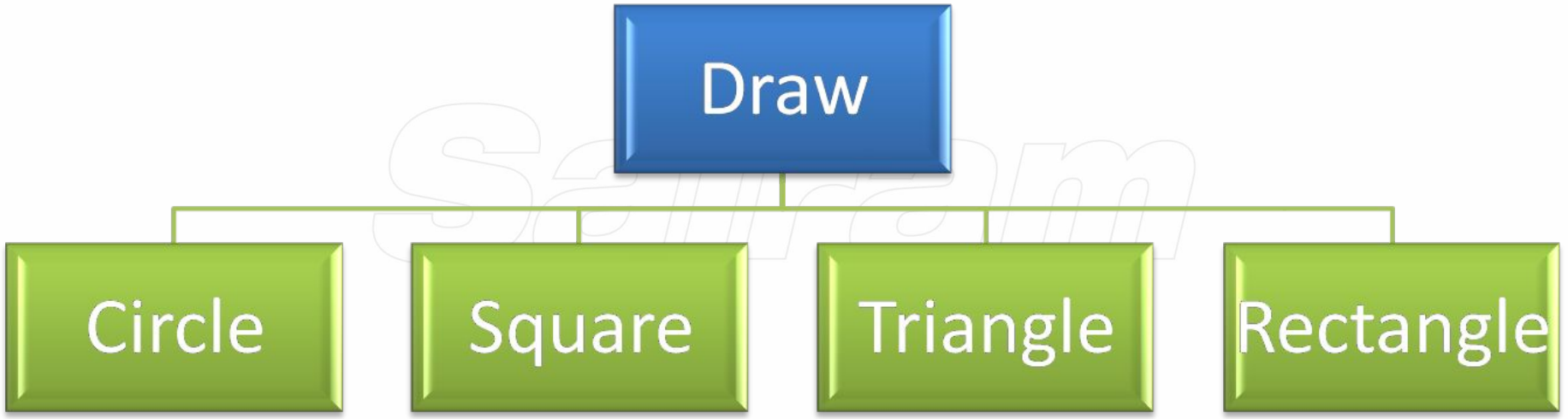
```
// Subclass
class Carloan extends Interest
{
  public void ComputeInterest()
        {
      System.out.println("Carloan: interest");
        }
}

class Bank  //main method class
{
  public static void main(String[] args)
 {
Homeloan myhomeloan = new Homeloan();
myhomeloan.getdata();
myhomeloan.ComputeInterest();

Carloan mycarloan = new Carloan();
mycarloan.getdata();
mycarloan.ComputeInterest();
 }
}
```

OOP in Java : polymorphism

Polymorphism

- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.
- If one task is performed in different ways is known as polymorphism.
- For example: A drawing task to the user. They draw different shapes like circle, triangle, rectangle, etc.
- In Java method polymorphism is achieved through method overloading and method overriding.

OOP in Java : polymorphism



OOP in Java : polymorphism

OOP in Java : polymorphism

Polymorphism Types

Ø There are **two types polymorphism**.

- •Compile Time polymorphism
- •Runtime polymorphism
- •

Ø **Compile Time polymorphism:**

- •It is also known as static polymorphism.
- •It is achieved by function overloading , operator overloading.

Ø **Runtime Time polymorphism:**

- •It is also known as Dynamic Method Dispatch.
- •It is a process in which a function call to the overridden method at Runtime.
- •This type of polymorphism is achieved by Method Overriding

OOP in Java : polymorphism example

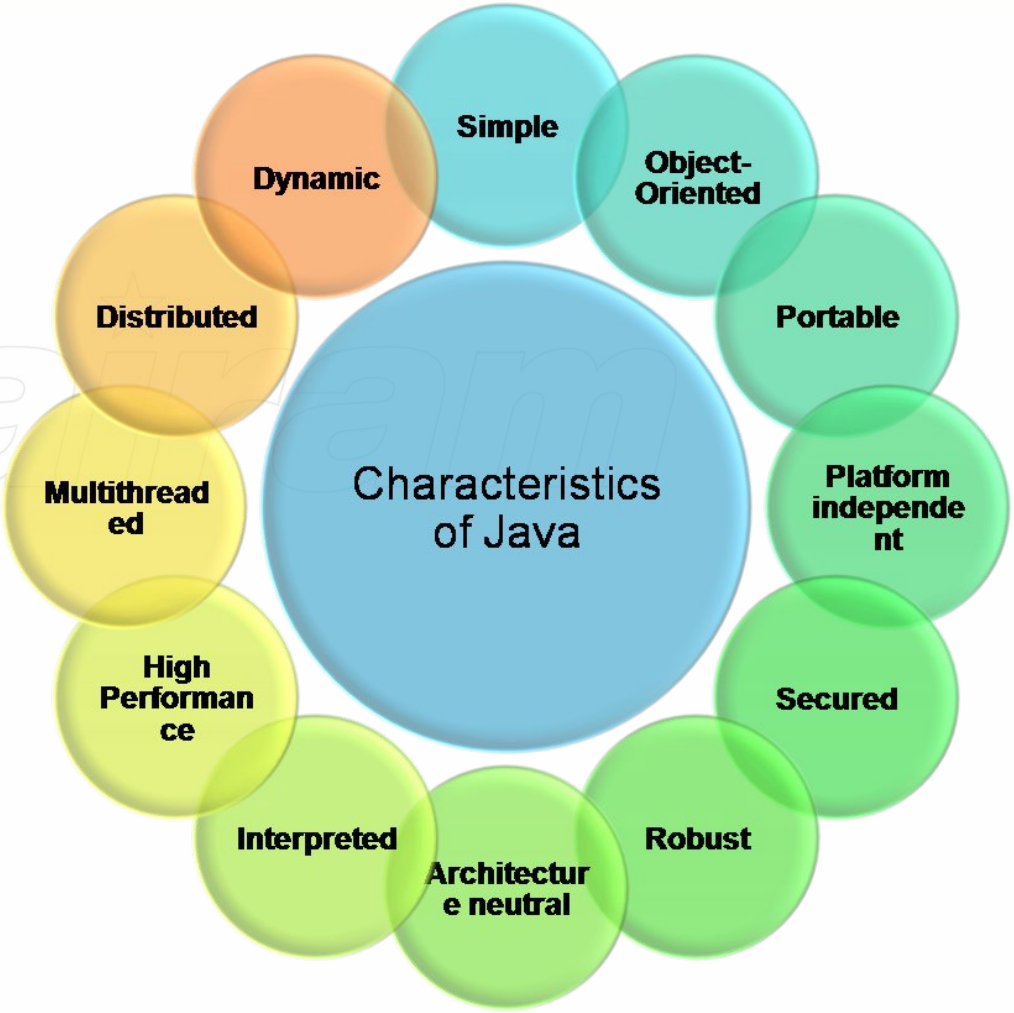
```
public class Sum
{

   public int sum(int x, int y)
    {
        return (x + y);
    }
  public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }
  public double sum(double x, double y)
    {
        return (x + y);
    }
```

```
public static void main(String args[])

    {
        Sum s = new Sum();

        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }


}   //end of class Sum
```

Characteristics of Java

Characteristics of java are:

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic



Characteristics of Java

1.Simple:

- Java is very easy to learn, and its syntax is simple, clean and easy to understand.
- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Characteristics of Java

2.Object-oriented:

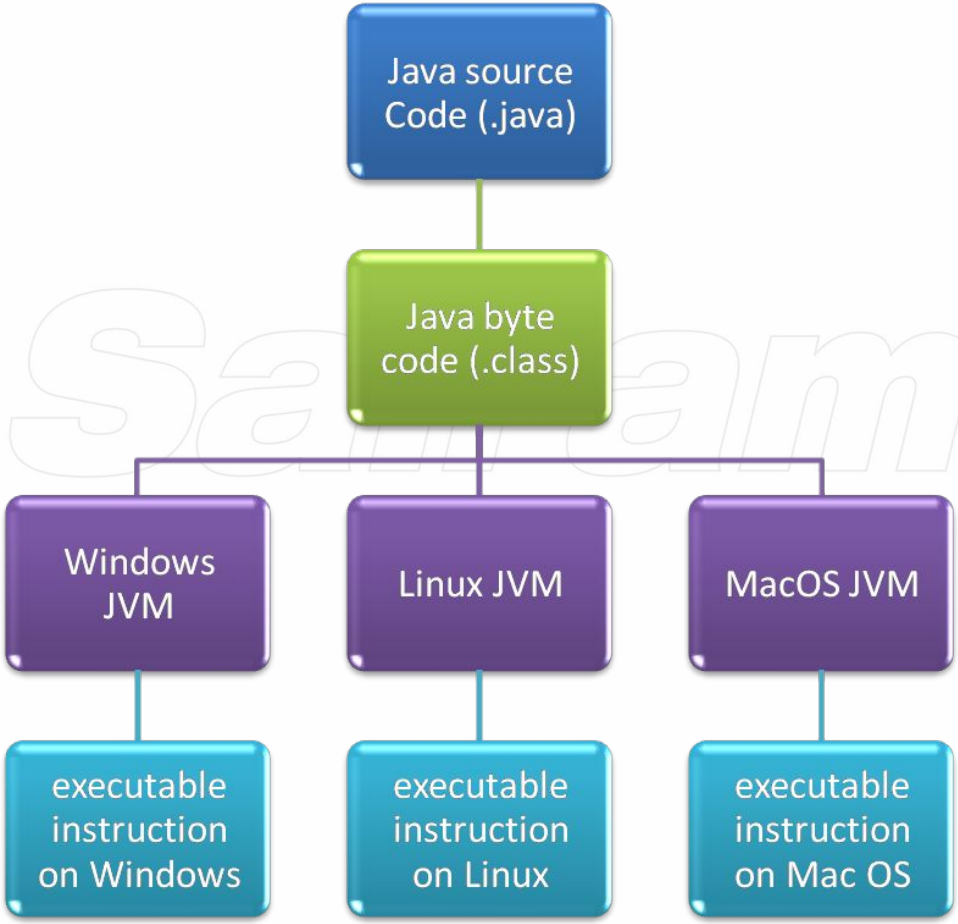
- Java is an object-oriented programming language.
- Everything in Java is an object.
- Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Java supports these OOPs concepts:
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation

Characteristics of Java

3.Platform Independent :

- Java is platform independent because write once, run anywhere language.
- Platform refers to hardware or software environment in which a program runs.
- Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.
- Java code is compiled by the compiler and converted into bytecode.
- This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Characteristics of Java : platform independent



Characteristics of Java

5.Secured :

- Java is secured because it has No explicit pointer and Java Programs run inside a virtual machine sandbox.

Java is secured by following :

- **Classloader (** It adds security by separating the package for the classes of the local file system)
- **Bytecode Verifier(** It checks the code fragments for illegal code that can violate access right to objects)
- **Security Manager(**It determines what resources a class can access such as reading and writing to the local disk)

Characteristics of Java

6. Robust :

- Robust simply means strong.
- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java.

All these points make Java robust.

Characteristics of Java

7. Architecture-neutral :

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies

- 2 bytes of memory for 32-bit architecture
- 4 bytes of memory for 64-bit architecture.
- In java it occupies 4 bytes of memory for both 32 and 64-bit architectures

Characteristics of Java

8.Portable :

- Java is portable because it facilitates you to carry the Java bytecode to any platform.
- Byte code is very small in size.
- It doesn't require any implementation.

9. High-performance :

- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
- It is still a little bit slower than a compiled language (e.g., C++).
- Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Characteristics of Java

10.Distributed :

- Java is distributed because it facilitates users to create distributed applications in Java.
- RMI and EJB are used for creating distributed applications.
- Because of this feature we can access files by calling the methods from any machine on the internet.

11. Multi-threaded :

- A thread is one task or program, executing concurrently.
- We can write Java programs that deal with many tasks at once by defining multiple threads.
- Multi-threading doesn't occupy memory for each thread.
- It shares a common memory area.
- Threads are used in gaming, multi-media and any Web applications.

Characteristics of Java

12. Dynamic :

- Java is a dynamic language.
- It supports dynamic loading of classes. (on demand class loading)
- It also supports functions from its native languages, i.e., C and C++.
- Java supports dynamic compilation and automatic memory management (garbage collection).

Video Link

https://youtu.be/a6YfG-QRewI

https://youtu.be/jQQaCoG_bAM