

BASUDEV GODABARI DEGREE COLLEGE ,

KESAIBAHAL

Department of Computer Science

“SELF STUDY MODULE”

Module Details :

- **Class - 3rd Semester (2019-20) Admission Batch**
 - **Subject Name : COMPUTER SCIENCE**
 - **Paper Name : Programming in JAVA**
-

UNIT – 2 : STRUCTURE

- 2.1 Object-Oriented Programming Overview
- 2.2 Principles of Object-Oriented Programming, Defining & Using Classes
- 2.3 Objects, Object reference
- 2.4 Objects as parameters, final classes, Garbage Collection
- 2.5 Introduction to Constructor
- 2.6 Types of constructor, Polymorphism
- 2.7 Method Overriding and restrictions.

Learning Objective

After Learning this unit you should be able to

- Know the Basic Concept Data Base Management system.
- Different types of Database.
- Normalization

You Can use the Following Learning Video link related to above topic :

<https://youtu.be/IL2PXC1fmnQ>

<https://youtu.be/lmy9TEJkKa8>

<https://youtu.be/Zt-ou1EPdlg>

You Can also use the following Books :

S.NO	Book Title	Author
1	An Introduction to Data Base System	CJ Pearson Edition
2	Fundamental of Database system	Remezemasari

And also you can download any book in free by using the following website.

- <https://www.pdfdrive.com/>

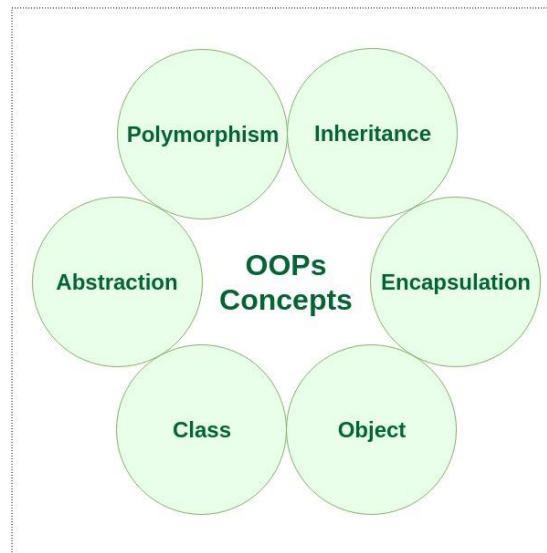
Unit-2

Object Oriented Programming (OOPs) Concept in Java

Object-oriented programming: As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOPs Concepts:

- [Polymorphism](#)
- [Inheritance](#)
- [Encapsulation](#)
- [Abstraction](#)
- [Class](#)
- [Object](#)
- [Method](#)
- [Message Passing](#)



Let us learn about the different characteristics of an Object-Oriented Programming language:

1. **Polymorphism:** Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.

For example:

1.

```
// Java program to demonstrate Polymorphism

// This class will contain
// 3 methods with same name,
// yet the program will
// compile & run successfully
public class Sum {

    // Overloaded sum().
    // This sum takes two int parameters
    public int sum(int x, int y)
    {
        return(x + y);
    }
}
```

```

// Overloaded sum().
// This sum takes three int parameters
public int sum(int x, int y, int z)
{
    return(x + y + z);
}

// Overloaded sum().
// This sum takes two double parameters
public double sum(double x, double y)
{
    return(x + y);
}

// Driver code
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));
}
}

```

2. **Output:**

3. 30

4. 60

5. 31.0

6. Polymorphism in Java are mainly of 2 types:

- [Overloading in Java](#)
- [Overriding in Java](#)

7. **Inheritance:** Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Important terminology:

- **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). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is **extends**.

Syntax:

```

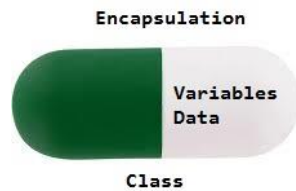
class derived-class extends base-class
{
    //methods and fields
}

```

8. **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. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.

- As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
- Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.



9. **Abstraction**: Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components. Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

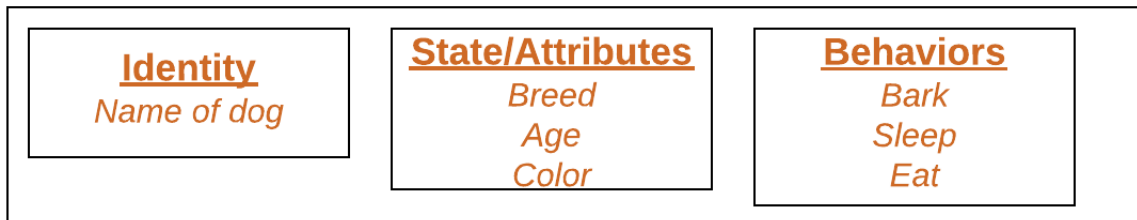
In java, abstraction is achieved by [interfaces](#) and [abstract classes](#). We can achieve 100% abstraction using interfaces.

10. **Class**: A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:
1. **Modifiers**: A class can be public or has default access (Refer [this](#) for details).
 2. **Class name**: The name should begin with a initial letter (capitalized by convention).
 3. **Superclass(if any)**: The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
 4. **Interfaces(if any)**: A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
 5. **Body**: The class body surrounded by braces, { }.

Object: It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

0. **State** : It is represented by attributes of an object. It also reflects the properties of an object.
1. **Behavior** : It is represented by methods of an object. It also reflects the response of an object with other objects.
2. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog

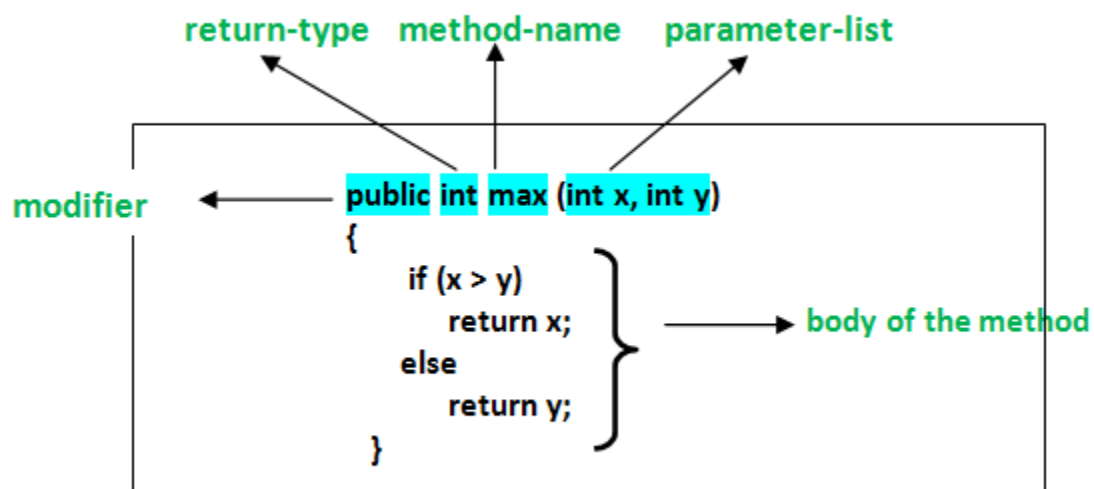


Method: A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python. Methods are **time savers** and help us to **reuse** the code without retyping the code.

Method Declaration

In general, method declarations has six components:

- **Access Modifier:** Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
 - **public:** accessible in all class in your application.
 - **protected:** accessible within the package in which it is defined and in its **subclass(es)(including subclasses declared outside the package)**
 - **private:** accessible only within the class in which it is defined.
 - **default (declared/defined without using any modifier):** accessible within same class and package within which its class is defined.
- **The return type:** The data type of the value returned by the method or void if does not return a value.
- **Method Name:** the rules for field names apply to method names as well, but the convention is a little different.
- **Parameter list:** Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
- **Exception list:** The exceptions you expect by the method can throw, you can specify these exception(s).
- **Method body:** it is enclosed between braces. The code you need to be executed to perform your intended operations.



Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the [DSA Self Paced Course](#) at a student-friendly price and become industry ready.

Constructor

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other start-up procedures required to create a fully formed object.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

Syntax

Following is the syntax of a constructor –

```
class ClassName {
    ClassName() {
    }
}
```

Java allows two types of constructors namely –

- No argument Constructors
- Parameterized Constructors

No argument Constructors

As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

Example

```
Public class MyClass {
    Int num;
    MyClass() {
        num = 100;
    }
}
```

You would call constructor to initialize objects as follows

```
public class ConsDemo {
    public static void main(String args[]) {
        MyClass t1 = new MyClass();
        MyClass t2 = new MyClass();
        System.out.println(t1.num + " " + t2.num);
    }
}
```

This would produce the following result

100 100

Parameterized Constructors

Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

Example

Here is a simple example that uses a constructor –

```
// A simple constructor.
class MyClass {
    int x;

    // Following is the constructor
    MyClass(int i ) {
        x = i;
    }
}
```

You would call constructor to initialize objects as follows –

```
public class ConsDemo {
    public static void main(String args[]) {
        MyClass t1 = new MyClass( 10 );
        MyClass t2 = new MyClass( 20 );
        System.out.println(t1.x + " " + t2.x);
    }
}
```

This would produce the following result –

```
10 20
```

Super and this keywords in Java

super keyword is used to access methods of the **parent class** while this is used to access methods of the **current class**.

[this keyword](#)

1. **this** is a reserved keyword in java i.e, we can't use it as an identifier.
2. **this** is used to refer **current-class's instance as well as static members**.

```
// Program to illustrate this keyword
// is used to refer current class
classRR {
    // instance variable
    inta = 10;

    // static variable
    staticintb = 20;

    voidGFG()
    {
        // referring current class(i.e, class RR)
        // instance variable(i.e, a)
        this.a = 100;

        System.out.println(a);

        // referring current class(i.e, class RR)
        // static variable(i.e, b)
```

```

        this.b = 600;

        System.out.println(b);
    }

    publicstaticvoidmain(String[] args)
    {
        // Uncomment this and see here you get
        // Compile Time Error since cannot use
        // 'this' in static context.
        // this.a = 700;
        newRR().GFG();
    }
}
100
600

```

Super keyword

1. **super** is a reserved keyword in java i.e, we can't use it as an identifier.
2. **super** is used to refer **super-class's instance as well as static members.**

```

// Program to illustrate super keyword
// refers super-class instance

classParent {
    // instance variable
    inta = 10;

    // static variable
    staticintb = 20;
}

classBase extendsParent {
    voidrr()
    {
        // referring parent class(i.e, class Parent)
        // instance variable(i.e, a)
        System.out.println(super.a);

        // referring parent class(i.e, class Parent)
        // static variable(i.e, b)
        System.out.println(super.b);
    }

    publicstaticvoidmain(String[] args)
    {
        // Uncomment this and see here you get
        // Compile Time Error since cannot use 'super'
        // in static context.
        // super.a = 700;
        newBase().rr();
    }
}
10
20

```

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class Super {
    .....
    .....
}
class Sub extends Super {
    .....
    .....
}
```

Sample Code

Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely Calculation and My_Calculation.

Using extends keyword, the My_Calculation inherits the methods addition() and Subtraction() of Calculation class.

Copy and paste the following program in a file with name My_Calculation.java

Example

```
classCalculation{
int z;

publicvoid addition(int x,int y){
    z = x + y;
System.out.println("The sum of the given numbers:"+z);
}

publicvoidSubtraction(int x,int y){
    z = x - y;
System.out.println("The difference between the given numbers:"+z);
}
}

publicclassMy_CalculationextendsCalculation{
publicvoid multiplication(int x,int y){
    z = x * y;
System.out.println("The product of the given numbers:"+z);
}

publicstaticvoid main(String args[]){
int a =20, b =10;
My_Calculation demo =newMy_Calculation();
    demo.addition(a, b);
    demo.Subtraction(a, b);
    demo.multiplication(a, b);
}
}
```

Compile and execute the above code as shown below.

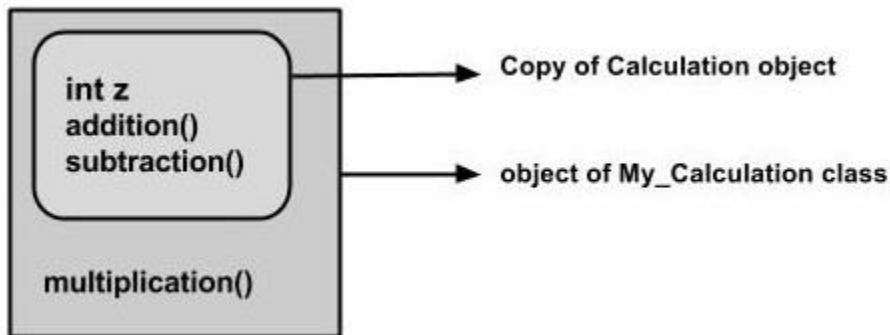
```
javac My_Calculation.java
java My_Calculation
```

After executing the program, it will produce the following result –

Output

```
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

In the given program, when an object to **My_Calculation** class is created, a copy of the contents of the superclass is made within it. That is why, using the object of the subclass you can access the members of a superclass.



The Superclass reference variable can hold the subclass object, but using that variable you can access only the members of the superclass, so to access the members of both classes it is recommended to always create reference variable to the subclass.

If you consider the above program, you can instantiate the class as given below. But using the superclass reference variable (**cal** in this case) you cannot call the method **multiplication()**, which belongs to the subclass **My_Calculation**.

```
Calculation demo =newMy_Calculation();
demo.addition(a, b);
demo.Subtraction(a, b);
```

Note – A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

The super keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.

- It is used to **invoke the superclass** constructor from subclass.

Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

```
super.variable  
super.method();
```

Sample Code

This section provides you a program that demonstrates the usage of the **super** keyword.

In the given program, you have two classes namely *Sub_class* and *Super_class*, both have a method named `display()` with different implementations, and a variable named `num` with different values. We are invoking `display()` method of both classes and printing the value of the variable `num` of both classes. Here you can observe that we have used `super` keyword to differentiate the members of superclass from subclass.

Copy and paste the program in a file with name `Sub_class.java`.

Example

```
class Super_class {  
    int num = 20;  
  
    // display method of superclass  
    public void display() {  
        System.out.println("This is the display method of superclass");  
    }  
}  
  
public class Sub_class extends Super_class {  
    int num = 10;  
  
    // display method of sub class  
    public void display() {  
        System.out.println("This is the display method of subclass");  
    }  
  
    public void my_method() {  
        // Instantiating subclass  
        Sub_class sub = new Sub_class();  
  
        // Invoking the display() method of sub class  
        sub.display();  
  
        // Invoking the display() method of superclass  
        super.display();  
  
        // printing the value of variable num of subclass  
        System.out.println("value of the variable named num in sub class:" + sub.num);  
  
  
        // printing the value of variable num of superclass  
        System.out.println("value of the variable named num in super  
class:" + super.num);  
    }  
  
    public static void main(String args[]) {  
        Sub_class obj = new Sub_class();  
        obj.my_method();  
    }  
}
```

```
}
```

Compile and execute the above code using the following syntax.

```
javac Super_Demo  
java Super
```

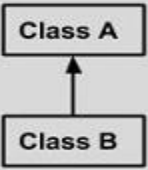
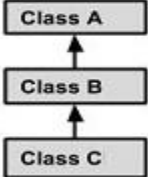
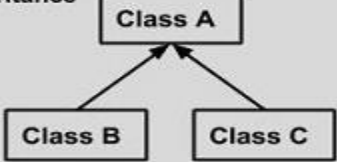
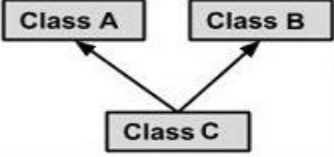
On executing the program, you will get the following result –

Output

```
This is the display method of subclass  
This is the display method of superclass  
value of the variable named num in sub class:10  
value of the variable named num in super class:20
```

Types of Inheritance

There are various types of inheritance as demonstrated below.

Single Inheritance  <pre>graph BT; B[Class B] --> A[Class A]</pre>	<pre>public class A { } public class B extends A { }</pre>
Multi Level Inheritance  <pre>graph BT; C[Class C] --> B[Class B]; B --> A[Class A]</pre>	<pre>public class A { } public class B extends A { } public class C extends B { }</pre>
Hierarchical Inheritance  <pre>graph BT; B[Class B] --> A[Class A]; C[Class C] --> A</pre>	<pre>public class A { } public class B extends A { } public class C extends A { }</pre>
Multiple Inheritance  <pre>graph BT; C[Class C] --> A[Class A]; C --> B[Class B]</pre>	<pre>public class A { } public class B { } public class C extends A,B { } // Java does not support multiple inheritance</pre>

A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class. Therefore following is illegal –

Example

```
public class extends Animal, Mammal {}
```

However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritance.

1) What is Java?

Java is the high-level, object-oriented, robust, secure programming language, platform-independent, high performance, Multithreaded, and portable programming language. It was developed by James Gosling in June 1991. It can also be known as the platform as it provides its own JRE and API.

2) List the features of Java Programming language.

There are the following features in Java Programming Language.

- **Simple:** Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.
- **Object-Oriented:** Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.
- **Portable:** Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.
- **Platform Independent:** Java is a platform independent programming language. It is different from other programming languages like C and C++ which needs a platform to be executed. Java comes with its platform on which its code is executed. Java doesn't depend upon the operating system to be executed.
- **Secured:** Java is secured because it doesn't use explicit pointers. Java also provides the concept of ByteCode and Exception handling which makes it more secured.
- **Robust:** Java is a strong programming language as it uses strong memory management. The concepts like Automatic garbage collection, Exception handling, etc. make it more robust.
- **Architecture Neutral:** Java is architectural neutral as it is not dependent on the architecture. In C, the size of data types may vary according to the architecture (32 bit or 64 bit) which doesn't exist in Java.
- **Interpreted:** Java uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.
- **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++).
- **Multithreaded:** We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

- **Distributed:** Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
- **Dynamic:** Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

3) What is the difference between JDK, JRE, and JVM?

JVM

JVM is an acronym for Java Virtual Machine; it is an abstract machine which provides the runtime environment in which Java bytecode can be executed. It is a specification which specifies the working of Java Virtual Machine. Its implementation has been provided by Oracle and other companies. Its implementation is known as JRE.

JVMs are available for many hardware and software platforms (so JVM is platform dependent). It is a runtime instance which is created when we run the Java class. There are three notions of the JVM: specification, implementation, and instance.

JRE

JRE stands for Java Runtime Environment. It is the implementation of JVM. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK

JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

4. How many types of memory areas are allocated by JVM?

Many types:

1. **Class(Method) Area:** Class Area stores per-class structures such as the runtime constant pool, field, method data, and the code for methods.
2. **Heap:** It is the runtime data area in which the memory is allocated to the objects

3. **Stack:** Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
4. **Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
5. **Native Method Stack:** It contains all the native methods used in the application.

5. What if I write static public void instead of public static void?

The program compiles and runs correctly because the order of specifiers doesn't matter in Java.

6. What is object-oriented paradigm?

It is a programming paradigm based on objects having data and methods defined in the class to which it belongs. Object-oriented paradigm aims to incorporate the advantages of modularity and reusability. Objects are the instances of classes which interact with one another to design applications and programs. There are the following features of the object-oriented paradigm.

- Follows the bottom-up approach in program design.
- Focus on data with methods to operate upon the object's data
- Includes the concept like Encapsulation and abstraction which hides the complexities from the user and show only functionality.
- Implements the real-time approach like inheritance, abstraction, etc.
- The examples of the object-oriented paradigm are C++, Simula, Smalltalk, Python, C#, etc.

7) What is the constructor?

The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

8) How many types of constructors are used in Java?

Based on the parameters passed in the constructors, there are two types of constructors in Java.

- **Default Constructor:** default constructor is the one which does not accept any value. The default constructor is mainly used to initialize the instance variable with the default values. It can also be used for performing some useful task on

object creation. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.

- **Parameterized Constructor:** The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say that the constructors which can accept the arguments are called parameterized constructors.

9) What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

```
class Student3{
int id;
String name;

void display(){System.out.println(id+ " "+name);}

public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
}
}
```

10. Does constructor return any value?

Ans: yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor)

1. Who is known as father of Java Programming Language?

James Gosling

2. In java control statements break, continue, return, try-catch-finally and assert belongs to?

Transfer statements

3. Which provides runtime environment for java byte code to be executed?

JVM

4. What is byte code in Java?

Code generated by a Java compiler

5. Which of the following are not Java keywords ?

Then

6. Which variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed?

Instance variables

7. Which symbol is used to contain the values of automatically initialized arrays?

Braces

8. Which of these operators is used to allocate memory to array variable in Java?

New

9. JIT stands for ____

Just In Time

10. _____ programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time.

11. ____ which allows programs to access SQL databases.

JDBC

12. A Java constructor is like a method without ____.

return type

13. The name of a constructor and the name of a class are ____.

Same

14. The placement of a constructor inside a class should be ____.

Anywhere in the class

15. The purpose of a Java constructor is ____.

All the above