DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560078

Department of Information Science and Engineering

(Accredited by NBA & NAAC with 'A' Grade)

DSCE

Dept. of Information Science & Engg

(2020-2021)

COURSE NAME: Java and J2EE

COURSE CODE: 18IS6DCJVA

CONTENTS

Unit - 1

1. Introduction to Java

Java is a general-purpose, object-oriented programming language designed for the development of software for consumer electronic devices, such as TVs, VCRs, toasters, etc.

Java is a platform neutral language, which means that it is not tied to any particular hardware or operating system. It guarantees users to 'write once, run anywhere.' The Java language is supported by almost every operating system, such as Sun Solaris, RedHat, Windows, etc.

1.1 The Creation of Java

The Java programming language was developed by Sun Microsystems of the USA in 1991; it was originally called Oak by James Gosling, who was one of the inventors of the language. The main goal for the developers was to make the language highly reliable, portable and simple.

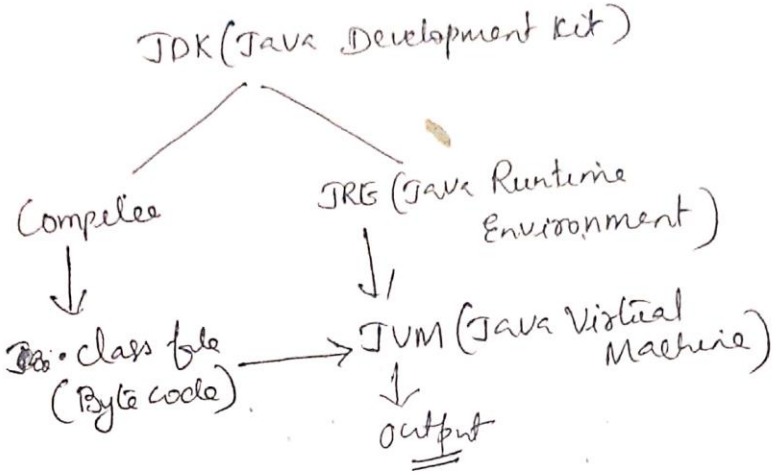
The team for the development of Java language included Patrick Naughton, who discovered that the existing languages such as C and C++ had some major drawbacks in terms of reliability and portability. They modeled the new language Java on C and C++, while removing some features that they considered constraints. This made Java a really simple, portable, and powerful language.

The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the seamanship for Java. In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called *OpenJDK*.

Over time new enhanced versions of Java have been released. The current version of Java is Java 1.8 which is also known as *Java 8*.

Java is defined by a specification and consists of a programming language, a compiler, core libraries and a runtime (Java virtual machine) The Java runtime allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine. The *Java platform* is usually associated with the *Java virtual machine* and the *Java core libraries*.

Java Programming



File extension for java is
- .java
  ↓ compilation
- .class (byte code)
  ↓
  JVM
  ↓
  output.

Compilation — javac filename.java

Execution → java filename

Java program is platform independent

JVM is platform dependent

- we can execute any where from the system if we set the path.

① Set classpath:

Cmd prompt

↓

Set path = c:\program files\java\jdk1.8\bin

② System properties

↳ Environment variables

↳ User variables

↳ New

↳ Variable name [Path]

Variable value [location]

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine.

The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows.

Java program is platform independent and JVM is platform dependent.

The Java language was designed with the following properties:

- **Platform independent**: Java programs use the Java virtual machine as abstraction and do not access the operating system directly. This makes Java programs highly portable. A Java program (which is standard-compliant and follows certain rules) can run unmodified on all supported platforms, e.g., Windows or Linux.
- **Object-orientated programming language**: Except the primitive data types, all elements in Java are objects.
- **Strongly-typed programming language**: Java is strongly-typed, e.g., the types of the used variables must be pre-defined and conversion to other objects is relatively strict, e.g., must be done in most cases by the programmer.
- **Interpreted and compiled language**: Java source code is transferred into the bytecode format which does not depend on the target platform. These bytecode instructions will be interpreted by the Java Virtual machine (JVM). The JVM contains a so called Hotspot-Compiler which translates performance critical bytecode instructions into native code instructions.
- **Automatic memory management**: Java manages the memory allocation and de-allocation for creating new objects. The program does not have direct access to the memory. The so-called garbage collector automatically deletes objects to which no active pointer exists.

The Java syntax is similar to C++. Java is case-sensitive, e.g., variables called `myValue` and `myvalue` are treated as different variables.

2. Java's Magic: The Bytecode

Byte code in Java is the reason java is platform-independent. As soon as a Java program is compiled byte code is generated in the form of a .class file. A byte code in Java is the instruction set for Java Virtual Machine and acts similar to an assembler. The working of bytecode is as follows:

- When a Java program is executed, the compiler compiles the piece of code and a Byte code is generated in the form of a .class file.
- But the byte code is a non-runnable code that requires or relies on an interpreter. This is where JVM plays an important part.
- The byte code generated after the compilation is run by the Java virtual machine.
- Resources required for the execution are made available by the Java virtual machine for smooth execution which calls the processor to allocate the resources.
- Pictorially the process can be represented as follows in figure 1.1:

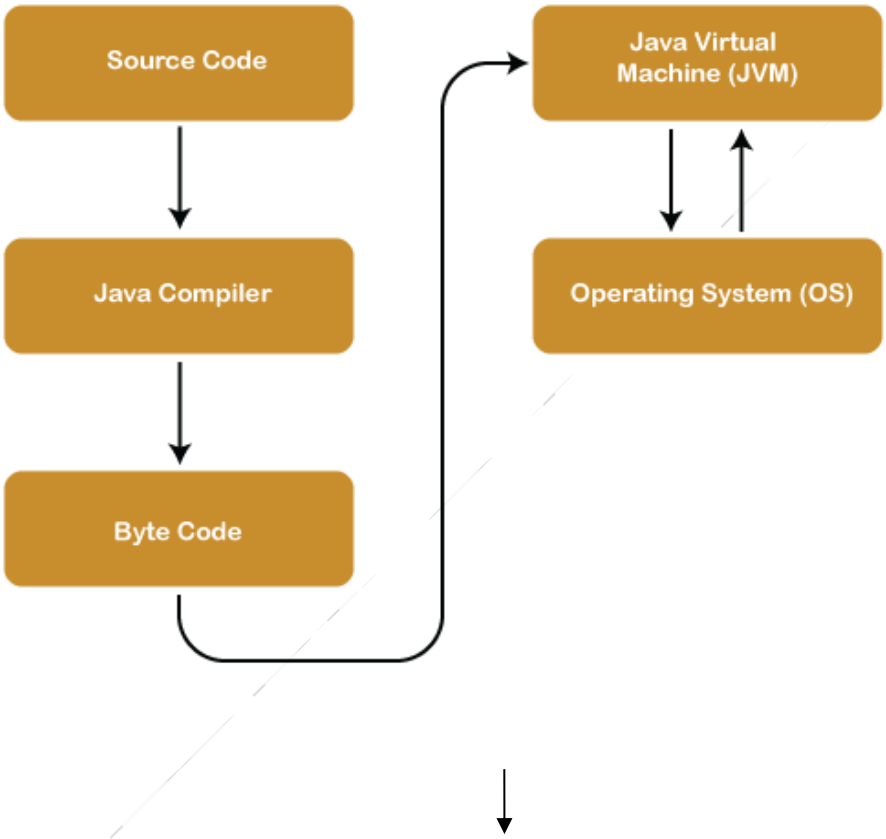


Figure 1.1: Compilation of source code

The Java Buzzwords

- The primary objective of Java programming language creation was to make it portable, simple and secure programming language.
- The features of Java are also known as java *buzzwords* (figure 1.2).

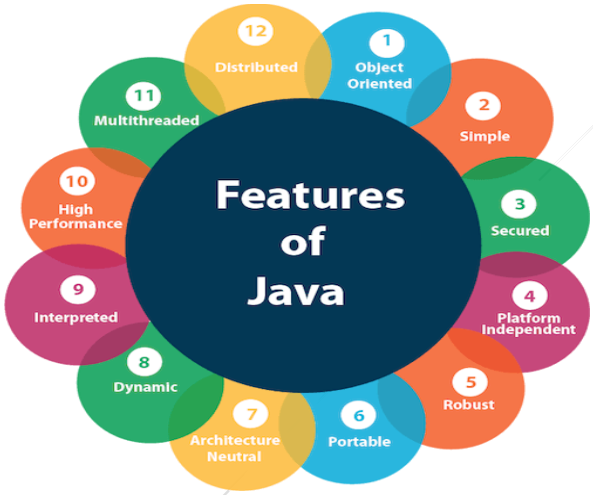


Figure 1.2: Features of Java

- A list of most important features of Java language are:

➢ Simple:

✓ Java is very easy to learn, and its syntax is simple, clean and easy to understand.

✓ According to Sun, Java language is a simple programming language because:

✓ 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.

- 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.
  - ✓ Basic concepts of OOPs are:
    - Object
    - Class
    - Inheritance
    - Polymorphism
    - Abstraction
    - Encapsulation

- Secured:

  - ✓ Java is best known for its security.
  - ✓ Java is secured because:

  - The **byte-code verification** takes place before the execution.

    Example: **Some of the checks that verifier performs:**
    - Uninitialized Variables
    - Access rules for private data and methods are not violated.
    - Method calls match the object Reference.

  - The **automatic** bounds checking of arrays, by raising an exception. Null checking of references, verification of casts prevents the program from making any type errors.

  - ✓ Java also provides library level safety.

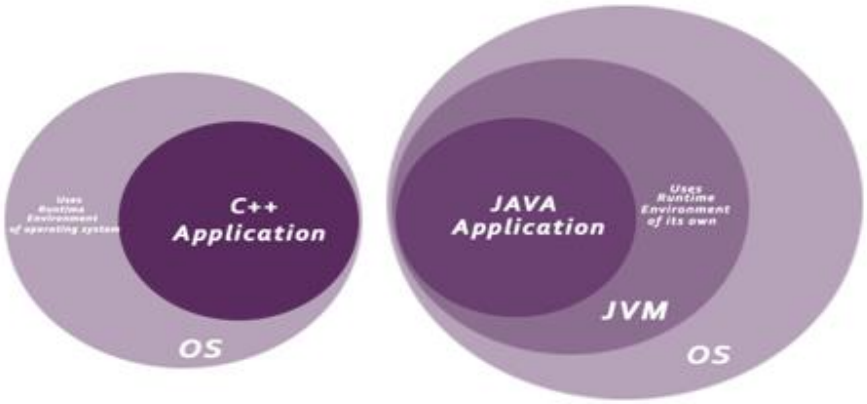


Figure 1.3: Java's security

- Platform independent:
  - ✓ Java is platform independent because Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.
  - ✓ There are two types of platforms software-based and hardware-based. Java provides a software-based platform.
  - ✓ The Java platform is a software-based platform that runs on the top of other hardware-based platforms.
  - ✓ It has two components:
  - ✓ Runtime Environment
  - ✓ API(Application Programming Interface)
  - ✓ 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).
- Robust
  - ✓ Java is robust because:
  - ✓ 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.
- ✓ Note: C program does not prevent the programmer from writing outside an array's bounds.

- Portable
  - ✓ Java is portable because it facilitates you to carry the Java byte code to any platform.
  - ✓ No dependency on the underlying hardware operating system.

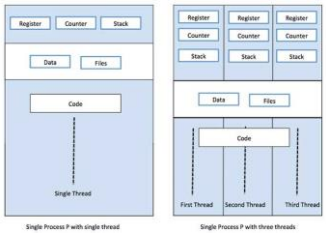
- Architecture neutral
  - ✓ Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
  - ✓ Java language and Java Virtual Machine helped in achieving the goal of "write once; run anywhere."
  - ✓ Changes and upgrades in operating systems, processors and system resources will not force any changes in Java Programs.
  - ✓ Note: In C the size of the data type depends on the architecture of the compiler. Eg if we take int datatype, if compiler is 16 bit, then int data type occupies 2 byte of memory, if compiler is 32 bit, then integer occupies 4 bytes), In java whatever be the architecture, memory allocation will not vary.

- 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++.
  - ✓ Java supports dynamic compilation and automatic memory management (garbage collection).

- Compiled and Interpreted
  - ✓ Usually a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.
  - ✓ Compiled : Java enables creation of a cross platform programs by compiling into an intermediate representation called Java Byte code.
  - ✓ Interpreted : Byte code is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.

- High Performance
  - ✓ Java performance is high because of the use of byte code.
  - ✓ The byte code was used, so that it was easily translated into native machine code.

- Multithreaded
  - ✓ A thread is like a separate program, executing concurrently.
  - ✓ We can write Java programs that deal with many tasks at once by defining multiple threads. (concurrent execution of several parts of the same program-Increases CPU utilization)
  - ✓ The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.



Single Process P with single thread Single Process P with three threads

  - ✓ Threads are important for multi-media, Web applications, etc.

- Distributed
  - ✓ Java is designed for distributed environment of the Internet.(Software that runs on multiple computers connected to a network at the same time)

- ✓ It is used for creating applications on networks.
- ✓ RMI and EJB are used for creating distributed applications.
- ✓ Java applications can access remote objects on Internet as easily as they can do in local system.
- ✓ Java enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Differences between C++ and Java

- **Pointers**
  - **Java:** Java does not support pointers, templates, pointer overloading, unions, etc.
  - **C++:** C++ does support pointers, structures, unions, templates, operator overloading, or pointers arithmetic.
- **Support Destructors**
  - **Java:** Java doesn't support destructors; it has an automatic **garbage collection system**.
  - **C++:** It supports destructors; it gets invoked when an object is destroyed.
- **Conditional Compilation and Inclusion**
  - **Java:** It doesn't support conditional compilation and inclusion.
  - **C++:** These are the major features of C++.
- **Thread Support**
  - **Java:** It has built-in supports -**threads in Java**.
  - **C++:** It has no built-in supports. It depends on third-party libraries.
- **Default Arguments**

– **Java:** Java does not support default arguments.

– **C++:** C++ supports default arguments.

- **Goto Statement**
  - **Java:** There is no goto statement in Java. The keywords const and goto are reserved, even though they are not used.
  - **C++:** C++ has goto articulation. Nonetheless, it isn't viewed as a great practice to a utilization of goto explanation.

- **Multiple Inheritances**
  - **Java:** Java doesn't provide multiple inheritances, at least not in the same sense that C++ does.
  - **C++:** C++ supports different inheritance. The keyword virtual utilize to determine ambiguities amid various legacy, if there is any.
- **Method Overloading and Operator Overloading**
  - **Java:** Java has method overloading but no operator overloading. The String class does use the + and += operators to concatenate strings and String expressions use automatic type conversion, but that's a special built-in case.
  - **C++:** C++ supports both technique over-loading and administrator over-loading.
- **Platform Independent**
  - **Java:** Java is interpreted for the most part and, hence, platform independent.
  - **C++:** C++ creates object code, and a similar code may not keep running on various stages

3. Object Oriented Programming

OOP is an approach to program organization and development, which attempts to eliminate some of the drawbacks of conventional programming methods by incorporating the best of structured programming features with several new concepts. OOP allows us to decompose a problem into number of entities called objects and then build data and methods (functions) around these entities. The data of an object can be accessed only by the methods associated with the object. An object-oriented program can be characterized as *data controlling access to code* by switching the controlling entity to data.

- **Some of the Object-Oriented Paradigm are:**

- Emphasis is on data rather than procedure.
- Programs are divided into objects.
- Data Structures are designed such that they characterize the objects.
- Methods that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through methods.

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.

```
Class student
{
Char  name[10];
Int rollno
Public:
      Void read()
{
 Name=”aaa”
  Rollno=12
}
} ;
```

Student S1, S2;

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.

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.

-Posses different behavior in different situations.

```
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);
}
```

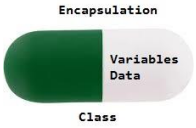
Inheritance: Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

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

}
Class Employee{
 float salary=40000;
}
class Programmer extends Employee
{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

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.



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.

NOTE:

IOStreams—Bytes of data-sequence of data

System.in----Read data from the keyboard

System.out---Display the data on the screen

System.err---Display the error message

All these iostreams is included in java.lang package.

Eg. import java.lang.*

System.out.print—output will be printed in single line

Eg: System.out.print("welcome")
System.out.print("java")
Output—welcome java

System.out.println—output will be printed in Multiple line

Eg: System.out.print("welcome")
System.out.println("java")
Output—welcome
java

4. First Simple Program

```
//This application program prints Welcome to Java!

class Welcome {
```

```
        public static void main(String[] args) {
            System.out.println("Welcome to Java!");
        }
    }
```

The most recognizable method in java is public static void main(string[] args) where public means that users have access to the method, static means that the method is based on a class rather than an instance, void means that nothing will be returned from the method, and main is the name of the function.

Compilation: javac Welcome.java

Execution: java Welcome

Output: Welcome to Java!

In Java, comments are preceded by two slashes (//) in a line(single line) , or enclosed between /* and */ in one or multiple lines.

Main function in java

(9)

Public static void main (String args[])

↓

access specifier

↓

- See the Public classes, methods can be accessed throughout the Program
- Making the main() method Public makes it globally available
- It is made public so that JVM can invoke it from outside the class as it is not present in the current class

- Any variables or methods can be invoked through an Objcit
- Static variables and methods can be invoked without any objcit
- main is also a method
- If we declare the method main as static no need to create an object. we can have direct access

Return type

In java every function should have a return type.

Public Static void main(String args[])

(10)

main → name of the functions

String args[] → In order to take the input from command line, we are using String args[]

Eg:-

javac cmd.java → cmd.class

java cmd 10 20 30

args[0] args[1] args[2]

↓

arguments which are passed are stored in string array.

5. Type Conversion and Casting

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly.

6.1 Widening or Automatic Type Conversion

- Widening conversion takes place when two data types are automatically converted. This happens when:
  - The two data types are compatible.
  - When we assign value of a smaller data type to a bigger data type.
  - For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or Boolean. Also, char and boolean are not compatible with each other.

Example:

```
class Test
{
    public static void main(String[] args)
    {
            int i = 100;

            // automatic type conversion
            long l = i;

            // automatic type conversion
            float f = l;
            System.out.println("Int value "+i);
            System.out.println("Long value "+l);
            System.out.println("Float value "+f);
    }
}
```

Output:

```
Int value 100
Long value 100
Float value 100.0
```

6.2 Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing. This is useful for incompatible data types where automatic conversion cannot be done.
Here, target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

Example:

```
//Java program to illustrate explicit type conversion
class Test
{
    public static void main(String[] args)
    {
            double d = 100.04;

            //explicit type casting
            long l = (long)d;

            //explicit type casting
            int i = (int)l;
            System.out.println("Double value "+d);

            //fractional part lost
            System.out.println("Long value "+l);

            //fractional part lost
            System.out.println("Int value "+i);
    }
}
```

Output:

```
Double value 100.04
Long value 100
Int value 100
```

6.3 Type promotion in Expressions

- While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted.
- Some conditions for type promotion are:
  - Java automatically promotes each byte, short, or char operand to int when evaluating an expression.

– If one operand is a long, float or double the whole expression is promoted to long, float or double respectively.

Example:

//Java program to illustrate Type promotion in Expressions

```
class Test
{
    public static void main(String args[])
    {
        byte b = 42;
        char c = 'a';
        short s = 1024;
        int i = 50000;
        float f = 5.67f;
        double d = .1234;

        // The Expression
        double result = (f * b) + (i / c) - (d * s);

        //Result after all the promotions are done
        System.out.println("result = " + result);
    }
}
```

Output:

Result = 626.7784146484375

6. Arrays

An array is a collection of similar types of data. It is a container that holds values of one single type. In Java, arrays are a fundamental construct that allows you to store and access a large number of values conveniently.

Declaring arrays:

<type>[] <arr_name>;

Or

<type> <arr_name[]>;

```
class ArrayDemo{
    public static void main(String args[]){
      int array[] = new int[7];
      for (int count=0;count<6;count++){
        array[count]=count+1;
      }
      for (int count=0;count<7;count++){
        System.out.println("array["+count+"] = "+array[count]);
      }

    }
}
```

Output:

```
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
```

7.1 **Multidimensional arrays**

Multidimensional arrays are actually arrays of arrays.

Ex: int twoD[ ][ ] = new int[4][5] ;

When allocating memory for a multidimensional array, specify the memory for the first dimension. The remaining dimensions can be allocated separately.

In Java, array length of each array in a multidimensional array is under user's control.

```
public class Guru99 {
        public static void main(String[] args) {

        // Create 2-dimensional array.
         int[][] twoD = new int[4][4];

         // Assign three elements in it.
         twoD[0][0] = 1;
         twoD[1][1] = 2;
         twoD[3][2] = 3;
         System.out.print(twoD[0][0] + " ");
    }
```

Output:

```
1
```

```
}
```

7. Introducing Classes

A class is a representation of some entity or thing which has **state** and **behavior**. **The state of a class is what the class *has*, while the behavior of a class is what the class can *do.***

8.1 Class Fundamentals

In Java everything is **encapsulated under classes**. Class is the core of Java language. It can be defined as **a template** that describe the behaviors and states of a particular entity.

A class defines new data type. Once defined this new type can be used to create object of that type.

Object is **an instance of class**. You may also call it as physical existence of a logical template class.

In Java, to declare a class **class** keyword is used. A class contain both **data and methods** that operate on that data. The data or variables defined within a class are called **instance variables** and the code that operates on this data is known as **methods**.

Thus, the instance variables and methods are known as **class members**.

- It may optionally extend only one parent class. By default, it extends **Object** class.
- The variables and methods are declared within a set of curly braces.

A Java class can contains fields, methods, constructors, and blocks. Lets see a general structure of a class.

Java class Syntax

```
class class_name{
        data;
        methods;
}
```

The fields declared inside the class are known as **instance variables**. It gets memory when an object is created at runtime.

Methods in the class are similar to the functions that are used to perform operations and represent behavior of an object.

8.2 Java Object

Object is an instance of a class while class is a blueprint of an object. An object represents the class and consists of **properties** and **behavior**. Properties refer to the fields declared with in class and behavior represents to the methods available in the class.

In real world, we can understand object as a cell phone that has its properties like: name, cost, color etc and behavior like calling, chatting etc.

So we can say that object is a real world entity. Some real world objects are: ball, fan, car etc.

There is a syntax to create an object in the Java.

Class_name var_name = new classname();

Example:

```
class Student{

        String name;
        int roll no;
        int age;


      public static void main(String[] args) {
            Student student = new Student();

            // Accessing and property value
            student.name = "Aditya";
            student.rollno = 23;
            student.age = 20;

            // Calling method
            System.out.println("Name: "+ student.name);
            System.out.println("Roll Number: "+ student.rollno);
            System.out.println("Age: "+ student.age);

      }
}
```

Output:

```
Name: Aditya
Roll Number: 23
Age: 20
```

8.3 Assigning Object Reference Variables

When one object reference variable is assigned to another object reference variable, a copy of the object is not created; it only makes a copy of the reference.

For example,

```
class Rectangle {
 double length;
 double breadth;
}



class RectangleDemo {
 public static void main(String args[]) {

 Rectangle r1 = new Rectangle();
 Rectangle r2 = r1;

 r1.length = 10;
 r2.length = 20;

 System.out.println("Value of R1's Length : " + r1.length);
 System.out.println("Value of R2's Length : " + r2.length);

 }
}
```

Output:

Value of R1's Length : 20.0

Value of R2's Length : 20.0

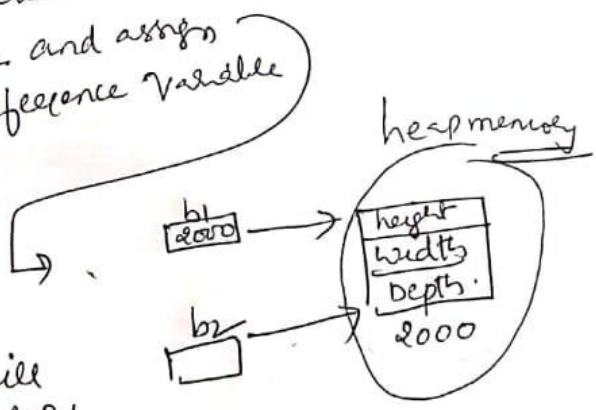
Object Vs Object Reference Variable

- An object is a chunk of memory, and a reference to the object is a way to reach upto that object in memory
- In simple words an object reference variable contains address of the object.

```
class Box
{
  double height;
  double width;
  double depth;
}
```

```
Box b1;          // declare reference
b1 = new Box();  // create an instance and assign it to reference variable b1
```

dynamic memory allocation



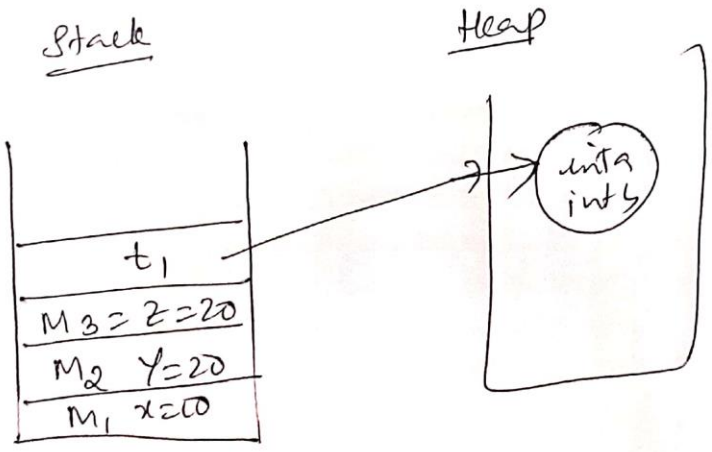

```
Box b2 = b1; // This will create copy of reference, Not object
```

Note:- To Run an application we require heap memory & stack. Stack stores all the local variable within the methods and methods, as well as reference variables.

Code:-

```
Public class Test
{
  int a,b;
void m1()
 {
   int x=10;
 }
Vooid m2()
 {
   int y=20
 }
void m3()
 {
  int z=30;
 }
```

```
main(---)
{
Test t1 = new Test();
 ---
 ---
}
```



8.4 **Introducing Methods**

Generally, methods in Java consist of these parts -

- Access Modifer (Optional) - public, private, or protected.
- Return Type - This is required, it denotes what value the method returns, or void if nothing is returned
- Method Name - follows camelCase convention
- Parameter List - List of parameters with their name and type, empty if no parameters are accepted
- Method body surrounded by { }

It is compulsory to define the return type of any method; otherwise it will fail at compile time. If nothing needs to be returned use void return type.

The general form of a method:

```
Type name( parameter-list) {
        //body of method
}
```

```
class Test {
   double item1;
   double item2;
   double item3;

      void value() {
       System.out.println("value is");
       System.out.println( item1 + item2 + item3);
   }
}

class Testdemo() {
   public static void main (String args[]) {
```

```
                Test test1 = new test();
                 Test test2 = new test();

                 test1.item1 = 10;
                 test1.item2 = 20;
                 test1.item3 = 15;

                 test1.value();
        }
}
```

Output:

```
value is 45.0
```

8.5 Constructors

Constructor is a block of code that initializes the newly created object. A constructor resembles an instance method in java but it's not a method as it doesn't have a return type. In short constructor and method are different. Constructor has same name as the class

8.5.1 Types of **Constructors**

There are three types of constructors:

Default: If there is no constructor in the class, Java compiler inserts a default constructor into code. This constructor is known as default constructor.

```
public class MyClass{
        public static void main(String[] args) {
                MyClass obj = new MyClass();
  }
  ..
}
```

→

```
public class MyClass{
  //This is the constructor
  MyClass(){
  }
        public static void main(String[]
       args) {
              MyClass obj = new MyClass();
  }
}
```

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```
class Student3{
int id;
String name;
//method to display the value of id and name
void display()
{
System.out.println(id+" "+name);
}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Output:
```
0 null
0 null
```

No-arg constructor: Constructor with no arguments is known as no-arg constructor. The signature is same as default constructor; however body can have any code unlike default constructor where the body of the constructor is empty.

```
lass Demo1{
```

```
        int width;
          int height;
             Demo1()
             {
                 width=0;
                  height=0;
             }
         void  display()
        {
        System.out.println(width);
        System.out.println(height);
        }
}

public class Demo
{

             public static void main(String args[]) {
               Demo1 d1=new Demo1();
                      d1.display();
             }
        }
```

Parameterized Constructor: Constructor with arguments (or parameters) is known as Parameterized constructor.

```
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display()
{
System.out.println(id+" "+name);
}
```

```
    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

8.5.2 Constructor vs. Method in Java

A Java method is a piece of code that has some specific name. It can be invoked during any point in the program by simply using the method name. It can also be understood as a subprogram that operates on data and returns some value.

The Java constructor is a special type of method. Both are similar in many ways, but not identical. Here are some of the most important differences between a Java constructor and a Java method:

- **Invoking –** While the constructor is invoked implicitly, the method is invoked explicitly
- **Java Compiler –** The Java compiler never provides a Java method. However, the Java compiler provides a default constructor if one isn't defined in a Java class
- Naming Convention – The name of the constructor in Java must be the same as that of the class. However, the method may or may not have the same name as that of the class containing it
- **Number of Calls –** A Java constructor is called once and only during the time of object creation. A Java method, on the other hand, can be called as many times are required
- **Return Type –** A Java method must have a return type but having the same for a constructor isn't mandatory
- Use – While a method is used for exposing the behavior of a Java object, a constructor is used for initializing the state of the same

8. The this Keyword

In Java, this keyword is used to refer to the current object inside a method or a constructor. For example:

```
class Demo {
   int instVar;

   Demo(int instVar){
      this.instVar = instVar;
      System.out.println("this reference = " + this);
   }

   public static void main(String[] args) {
      Demo obj = new Demo(8);
      System.out.println("object reference = " + obj);
   }
}
```

Output:

```
this reference = edu.ThisAndThat.MyClass@74a14482
object reference = edu.ThisAndThat.MyClass@74a14482
```

- this can be used to overcome Ambiguity in variable names : In Java, it is not allowed to declare two or more variables having the same name inside a scope (class scope or method scope). However, instance variables and parameters may have the same name where in the ambiguity can be overcome using this keyword.

- this with Getters and Setters: this keyword can also be used to assign value inside the setter method and to access value inside the getter method.

- Using this in Constructor overloading : While working with constructor overloading, one might have to invoke one constructor from another constructor. In such case, constructor cannot be called explicitly. Instead this keyword can be used.

- In the below example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

```
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();
}}
```

Test it Now

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
```

```
    void display(){System.out.println(rollno+" "+name+" "+fee);}
    }

    class TestThis2{
    public static void main(String args[]){
    Student s1=new Student(111,"ankit",5000f);
    Student s2=new Student(112,"sumit",6000f);
    s1.display();
    s2.display();
    }}
```

Test it Now

Output:

```
111 ankit 5000
112 sumit 6000
```

this() : to invoke current class constructor

```
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```

9. The finalize( ) Method

- Sometimes an object will need to complete some important task before it gets destroyed such as closing an open connection, etc. If these connections are not closed, then it will create problem in some applications (e.g., web application).

- In C or C++, the programmer has to deallocate the memory, but in Java deallocation of memory is handled by the garbage collector rather than the programmer. The garbage collector frees the memory for all the unreferenced object.

- Garbage collector does not perform such activities of making the resources free held by objects. Therefore it is a responsibility of the programmer to make these resources free just before destroying the object. For that, the programmer can define a very important method called finalize() method in which all the important tasks that object will need to perform before it gets destroyed are done.

- Garbage collector calls this finalize() method automatically every time just before destroying the object. Thus all the resources held by object become free before the object gets destroyed and the application runs smoothly. It performs the clean up process.

A method named ***finalize()*** is executed before an object is de-allocated from the heap memory by the *garbage collector*. Each class inherits the ***finalize()*** method from the **Object** class, which is a superclass of all java classes. Override the ***finalize()*** method to give it own preferred definition.

Syntax:

```
protected void finalize()
{
  // statements
}
```

Example:

```
public class Car
{
  protected void finalize()
  {
    System.out.println("Finalize method called");
    System.out.println("Objects got destroyed");
  }

  public static void main(String args[])
  {
    Car c1 = new Car();
```

```
        Car c2 = new Car();
        c1 = null;
        c2 = null;
        System.gc();
    }
}
```

Note: The output may be different every time the program is run because it can't be predic ted whether garbage collector will be invoked or not by **JVM**.