Object Oriented Programming (with JAVA) (UNIT-1)

by
Dr. Partha Roy,
Associate Professor,
Bhilai Institute of Technology, Durg

According to Oracle

- Java is positioned as first for programming language
- Number one programming language for today's tech trends
- No.1 programming language for cloud
- Globally 45 billion active JVMs
- Java is the popular choice for many organizations than any other programming language.
- Positioned as no.1 for DevOps, AI, VR, Big Data, Continuous Integration, Analytics, Mobile, and Chatbots.

Ref: https://www.interviewbit.com/blog/java-developer-salary/

Important skills required in a Java developer

- Fundamentals of Object-Oriented Principles
- Basic understanding of Relational Database Management Systems like ORM or SQL
- Spring Framework: An important component of Java Development that allows companies to build web apps.
- Learn about the Application Program Interfaces (APIs) and Common Libraries. : Some common examples are Guava, Apache Commons, Maven, etc.
- In-depth understanding of Java Virtual Machine and its elements.
- Algorithm Methodologies
- Data Structures

Ref: https://www.interviewbit.com/blog/java-developer-salary/

UNIT- I

Introduction & Fundamentals of JAVA, Background of JAVA, About Java Technology, Java's architecture, Reading console inputs, Arrays, Constructors, Finalize method, final, this method and reference, static members.

Background of JAVA

- **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June **1991**.
- Java was originally designed for interactive television.
- The language was initially called Oak after an oak tree that stood outside Gosling's office; it went by the name Green later, and was later renamed Java, from Java coffee, said to be consumed in large quantities by the language's creators.
- Gosling aimed to implement a virtual machine and a language that had a familiar C/C++ style of notation.

Some major release versions of Java

- JDK 1.0 (1996)
- J2SE 5.0 (2004)
- Java SE 6 (2006) and many more

Presently

- Java SE 18 March 2022

JAVA Programming Platforms

- 1. JAVA SE
- 2. JAVA EE
- 3. JAVA ME
- 4. JAVA Card

JAVA Programming Platforms

- **JavaSE = Standard Edition.** This is the core Java programming platform. It contains all of the libraries and APIs that any Java programmer should learn (java.lang, java.io, java.math, java.net, java.util, etc...). Java SE is for developing desktop applications and it is the foundation for developing in Java language. It consists of development tools, deployment technologies, and other class libraries and toolkits used in Java applications.
- **JavaEE = Enterprise Edition**. Java EE is built on top of Java SE, and it is used for developing web applications and large-scale enterprise applications. The Java platform (Enterprise Edition) differs from the Java Standard Edition Platform (Java SE) in that it adds libraries which provide functionality to deploy fault-tolerant, distributed, multi-tier Java software, based largely on modular components running on an application server.

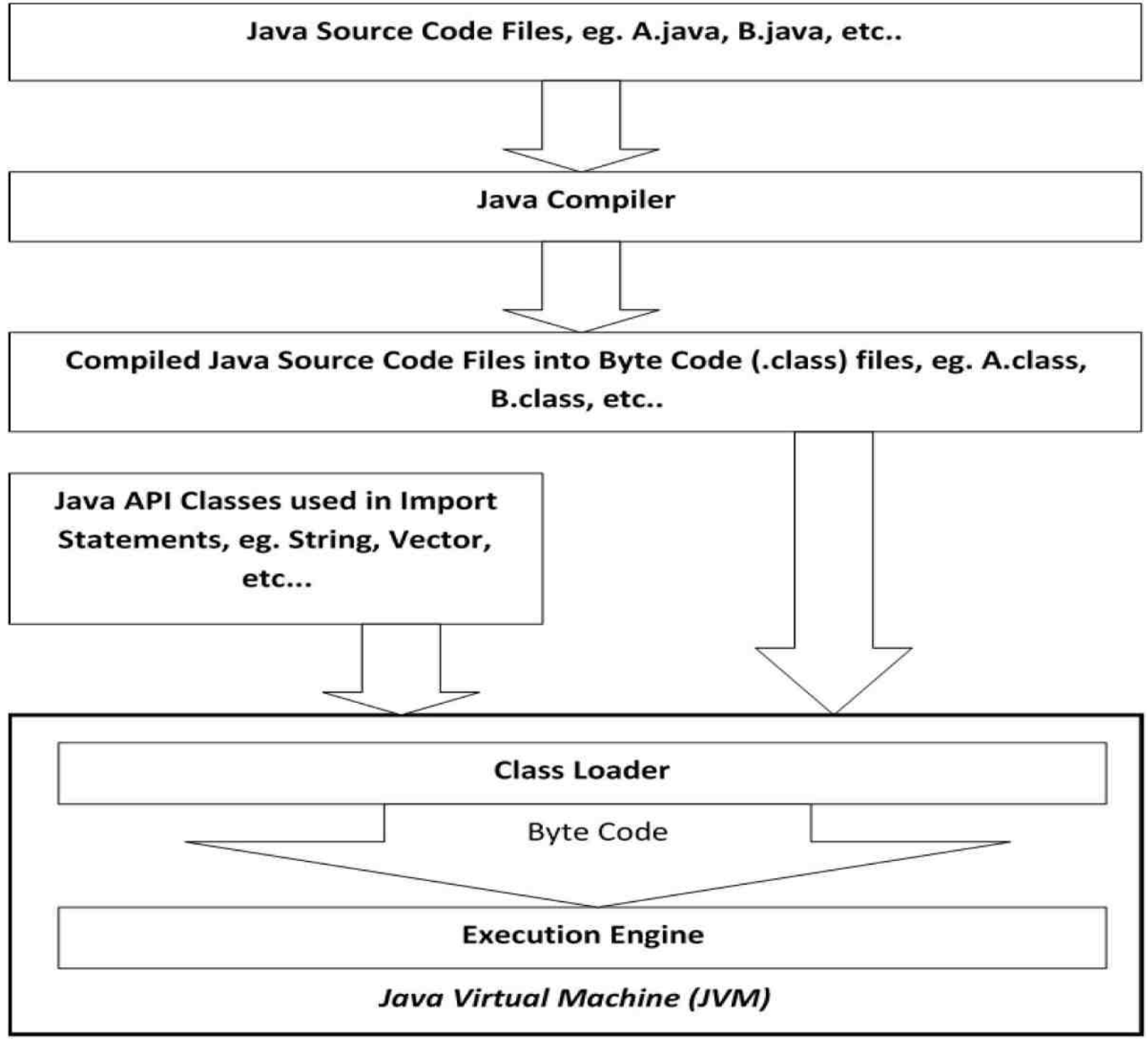
JAVA Programming Platforms

- **JavaME = Micro Edition.** This is the platform for developing applications for mobile devices and embedded systems such as set-top boxes. JavaME provides a subset of the functionality of JavaSE, but also introduces libraries specific to mobile devices. Java ME is a subset of the Java SE. It provides an API and a small-footprint virtual machine for running Java applications on small devices.
- **Java Card** is a software technology that allows Java-based applications (applets) to be run securely on smart cards and similar small memory footprint devices. Java Card is the tiniest of Java platforms targeted for embedded devices. Java Card gives the user the ability to program the devices and make them application specific. It is widely used in ATM cards.The first Java Card was introduced in 1996.

About Java Technology

- The Java technology consists of **three** main components:
- **The Java programming language:** It is the set of programming statements and syntax using which various type of Java applications are written. The programming language has its specific syntax and style of writing the code.
- **The Java Virtual Machine:** When a Java program is compiled, it is converted to byte codes that are the portable machine language of a CPU architecture known as the Java Virtual Machine (also called the Java VM or JVM). The JVM can be implemented directly in hardware, but it is usually implemented in the form of a software program that interprets and executes byte codes.
- **The Java platform:** The Java platform is distinct from both the Java language and Java VM. The Java platform is the predefined set of Java classes that exist on every Java installation; these classes are available for use by all Java programs. The Java platform is also sometimes referred to as the Java runtime environment or the core Java APIs (application programming interfaces). The Java platform can be extended with optional packages (formerly called standard extensions).

Java's architecture



Java Virtual Machine (JVM)

- The Java Virtual Machine knows nothing of the Java programming language, only of a particular binary format, the class file format.
- A class file contains Java Virtual Machine instructions (or bytecodes) and a symbol table, as well as other ancillary information.
- For the sake of security, the Java Virtual Machine imposes strong syntactic and structural constraints on the code in a class file. However, any language with functionality that can be expressed in terms of a valid class file can be hosted by the Java Virtual Machine.

Case-Sensitivity and Naming Convention

- Java is a case-sensitive language.
- Its keywords are written in lowercase and must always be used that way.
- The standard naming convention used in java is as follows:
  - The Class names should start with Capital case letter.
  - The Method names should start with Small case letter.
  - Whenever there are multiple words in the name of a class or method then starting letter of the word should be a capital letter followed by small letter, eg. Integer is a class and parseInt() is a method.

Reserved Keywords

- The following words are reserved in Java: they are part of the syntax of the language and may not be used to name variables, classes, and so forth.

abstract, final, int, public, throw,
continue, finally, interface, return , throws,
boolean, default, float, long, short, transient,
break, do, for, native, static, true,
byte, double, new, strictfp, try,
case, else, if, null, super, void,
catch, enum, implements, package, switch, volatile,
char, extends, import, private synchronized, while,
class, false, instanceof, protected, this

Identifiers

- An identifier is simply a name given to some part of a Java program, such as a class, a method within a class, or a variable declared within a method.
- Identifiers may be of any length and may contain letters and digits drawn from the entire Unicode character set.
- An identifier may not begin with a digit, however, because the compiler would then think it was a numeric literal rather than an identifier.
- It cannot contain white spaces, for eg. int my var=10; is not valid as it should be int myvar=10; or int my_var=10;

Literals

- Literals are values that appear directly in Java source code.
- They include integer and floating-point numbers, characters within single quotes, strings of characters within double quotes,
- They also include the reserved words true, false and null.
- Examples, 10.23, ‘a’, “my string”, 100

Primitive Data-types

Type	Contains	Default	Size	Range
boolean	true or false	false	1 bit	NA
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
byte	Signed integer	0	8 bits	-128 to 127
short	Signed integer	0	16 bits	-32768 to 32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
long	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
float	floating point	0.0	32 bits	1.4E-45 to 3.4028235E+38
double	floating point	0.0	64 bits	4.9E-324 to 1.7976931348623157E+308

Strings

- In addition to the char type, Java also has a data type for working with strings of text (usually simply called strings).
- The String type is a class, however, and is not one of the primitive types of the language.
- Because strings are so commonly used, though, Java does have a syntax for including string values literally in a program.
- A String literal consists of arbitrary text within double quotes. Eg. “This is a String literal”

Comparison between C++ and Java

C++	Java
Strongly influenced by C syntax, with Object-Oriented features added.	Strongly influenced by C++/C syntax.
Pointers can be used explicitly	Pointers are not available to the programmers.
Garbage collection is not automatic.	Garbage collection is automatic and can be initiated by the user.
Supports structs, union and classes.	Supports classes and interfaces.
Type conversion between basic types are not restricted.	Type conversions between primitive types is limited when high precision type data is converted to low precision type data.
Operator overloading can be done explicitly	Operators like + and += is already overloaded. Explicit operator overloading not allowed.

Comparison between C++ and Java

C++	Java
Function overloading can be done explicitly	Function overloading can be done explicitly
Multiple class inheritance is allowed.	Multiple class inheritance is not allowed, but multiple interface inheritance is allowed.
const keyword is used for making classes, functions, objects and variables non inheritable and non mutable.	**final** classes become non-inheritable, **final** functions become non-overridable but can be inherited and **final** objects and variables become non mutable, but can be inherited.
Supports goto statement.	Does not support goto statement.
Source code has to be re-written if the underlying operating system changes.	Source code need not be changed if the underlying operating system changes, only the appropriate JVM is required.
Exception handling process is not strong.	Strong exception handling procedures are provided.

Important Links (JDK Downloads)

JDK-8 Download and
Netbeans 8 bundled with JDK8 Download:

https://drive.google.com/drive/folders/1NRVOSdOgVZjB0fHcBD0qJiYU90oGYlJk?usp=sharing

Important Links (Online Compilers)

- https://www.jdoodle.com/online-java-compiler/
- https://www.programiz.com/java-programming/online-compiler/
- https://www.onlinegdb.com/online_java_compiler

Writing a basic-level program in JAVA

Following is a simple program in Java:

```
class A
{
        public static void main(String arg[])
        {
                System.out.println("WELCOME TO JAVA");
        }
}
```

Writing a basic-level program in JAVA

We write the above program in a text editor and save it with any name, say, MyProgram1.java and save it in the window's D: (D drive).
Now in order to compile this program we need to type in the command prompt the following code:
D:\> **javac** MyProgram1.java
This would create a .class file of the class defined as A in the file. So we would get A.class file generated as the bytecode file for the defined class A. Now to run this class file we need to type in the command prompt the following code:
D:\> **java** A
And the output would be
WELCOME TO JAVA

Points to remember

- **1.** It is not necessary to save the java file with the same name as the class name containing the main method.
- **2.** It is not necessary to declare the class containing the main method as public.
- **3.** The only access specifier that can be used with class is either public or no specifier at all. Using specifiers like private and protected is not allowed with any class inside a java file.

The syntax of the main method

public static void main(String arg[])

- The main method is called by the JVM to start the execution of our program.
- The main method is declared as public because the JVM can call it from anywhere.
- The main method is declared as static because static members can be called without the need of any object. Also if the program execution starts from main method then there might not be any object existing before the main starts, so it is needed to call the main method without the need of any object reference. Hence the main method can be called by using the class_name.main(...) as it is declared as static.

The syntax of the main method

public static void main(String arg[])

- The return type of the main method is void as it is not required to return any value to the JVM. It would be appropriate to have some return type if we call the main method from any other program.
- The name main is used because it is configured in the JVM's code to search for this name. If we want another name then we have to customize the JVM code accordingly.
- String array is present in the input argument of main method. It is used to store the command-line inputs while running the class file.

Reading Input from the Command Prompt

- There are mainly three ways using which we can read input during runtime through command prompt.
- **1.** Using Command line inputs.
- **2.** Using java.io.InputStreamReader class.
- **3.** Using java.util.Scanner class.

Reading input using Command Line Input

File name: A.java

```
public class A
{
  public static void main(String arg[])
  {
        System.out.println("Data      Entered
through command line:");
        for(int i=0;i<arg.length;i++)
        {
              System.out.println(arg[i]);
        }
  }
}
```

Output:

```
C:\>javac A.java
C:\>java A This Is My Java Program
Data Entered through command line:
This
Is
My
Java
Program
```

Using "InputStreamReader" and "BufferedReader" classes.

Click here for Code

Using "Scanner" class

Click here for Code

Creating and using arrays in Java

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. Each item in an array is called an element, and each element is accessed by its numerical index.

Basic syntax for declaring an array is:

DataType[ ] ArrayName = new DataType[size];
// where size should be a positive integer.

OR

DataType ArrayName[ ] = new DataType[size];
// where size should be a positive integer.

Creating and using arrays in Java

```
class ArrayDemo {
    public static void main(String[] args) {
            // declares an array of integers
        int[] anArray;

            // allocates memory for 5 integers
        anArray = new int[5];

            // initialize first element
        anArray[0] = 100;
             // initialize second element
        anArray[1] = 200;
            // and so forth
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;

        System.out.println("Element at index 0: " + anArray[0]);
        System.out.println("Element at index 1: " + anArray[1]);
        System.out.println("Element at index 2: " + anArray[2]);
        System.out.println("Element at index 3: " + anArray[3]);
        System.out.println("Element at index 4: " + anArray[4]);
    }
}
```

Tip:
We can use the built-in length property to determine the size of any array.
The following code prints the array's size to standard output:

System.out.println(anArray.length); // the output is 5

Output:
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500

Object Oriented Concepts

- **1.** Object
- **2.** Class
- **3.** Encapsulation
- **4.** Abstraction
- **5.** Data Hiding
- **6.** Inheritance
- **7.** Polymorphism
- **8.** Dynamic binding
- **9.** Message Passing

Object

- An Object is an Instance of a Class. An Instance is the existence in the computer system by acquiring a memory space.
- An Object is the only way a Class becomes usable.
- Objects are basis of Object Oriented Programming.
- An Object has all the Characteristics of the Class using which the Object is created.
- It implements reusability of code written in Classes.

Class

- A Class is a Blueprint or Stencil for creating Objects.
- A Class specifies all the Member Data and Member Functions that would be present in the Objects created using the Class.
- Using a Class any number of Objects can be created.
- It's a User Defined Data Type also called Abstract Data Type, which acts like a basic data type when used.
- Usually Classes are used to represent computer understandable formats for Real World Entities.

Encapsulation, Abstraction & Data Hiding

Encapsulation:

- It's the process of wrapping up of Data and Functions inside a single Entity (usually Classes).
- Encapsulation helps to establish Abstraction and Data Hiding.
- Classes, Interface and Member methods implement Encapsulation at programmer level.

Abstraction:

- It's the process of Hiding the complexities of implementation and providing a simple Interface for easy use.
- Its implemented using Encapsulation.
- Classes, Interface and Member methods implement Abstraction at programmer level.

Data Hiding:

- It's the process of keeping the Data under such an Access mode that its only accessible to permitted Functions.
- Using Private, Protected specifiers, we can hide Data from access from outside a Class.
- Classes and Member methods implement Data Hiding at programmer level.

Inheritance

- It's the process of Passing Attributes (data members) and Characteristics (member functions) of one Class to other Classes.
- Here the Class which gets the Inherited properties is called the Child Class and from which it acquires the properties is called the Base or Parent Class.
- During Inheritance the Protected and Public Data and Functions get passed on as copies from the Parent or Base Classes to Child Classes.
- It forms a Hierarchical Structure from Parent down to Children Classes.
- It implements Reusability of Parent Class Data and Functions.
- Interface and Classes can be used to implement Inheritance.
- **Types :- Single, Multilevel, Multiple and Hierarchical.**

Polymorphism, Dynamic binding & Message Passing

Polymorphism:

- It's the process of defining more than one kind implementation (coding) of a method, using same name with either Different number of Arguments or Different Data types of Arguments.
- Implementation :- Method Overloading and Constructor Overloading
- Polymorphism Types:- Compile Time (the compiler recognizes the binding between the method and its code during compile time) and Run Time(the compiler recognizes the binding between the method and its code during runtime)

Dynamic binding:

- Its also called Runtime Polymorphism, where the compiler recognizes the binding between the method call and its code during runtime.
- Its usually implemented using Interface Inheritance.

Message Passing:

- It's the process of passing Data between Objects of same Class and also of different Classes.
- Its implemented using member method calling.
- When we pass values in the input arguments in a member method then we are passing message to the object which is used to call the method.
- When a member method returns a value then that value is the message received by us from an object that is used to call the method.

Constructors in Java

- It's a special method which has the same name as that of the Class name.
- It has No Return-Type, not even void.
- Constructor overloading is possible.
- Their accessibility gets affected by the Access specifiers (default, private, public and protected).
- They are to be invoked explicitly as soon as Instance of a Class is created.
- We can explicitly call Constructors of a Class using this() method, but this() can only be invoked from another constructor of the same class and should be the first statement in the constructor.
- They can Not be Inherited.

Constructors in Java

There are two basic categories of Constructors

- **Default Constructor**

Syntax : ***class-name ( ) { code }***

- **Parameterized Constructor**

Syntax: ***class-name (parameter-list) { code }***

- For every Class, the compiler creates a Default implicitly.
- If any of the two categories of Constructors are defined in the Class then the compiler in any case would never use the implicitly created Constructor, it would only use the user defined Constructor.

Garbage Collection in Java

- **Finalize** is the method used for the purpose of garbage collection when an object goes out of scope or becomes null.
- **Finalize** method is called by the garbage collector just before destroying an object to perform cleanup activities.
- **finalize()** method is defined in **java.lang.Object** class, which means it available to all the classes for sake of overriding.

Garbage Collection in Java

- One of the most important point of finalize method is that its not automatically chained like constructors. If we are overriding finalize method then its our responsibility to call **finalize()** method of super-class, if we forget to call it then finalize of super class will never be called. The best way to call super class finalize method is to call them in finally block using **super.finalize().**
- Any Exception thrown by finalize method is ignored by Garbage Collector thread and it will not be propagated further.
- There are two ways of running the finalize method, is by calling **System.gc()** or explicitly calling the finalize() method like a member method using the object.
- The prototype of finalize() method is as follows:

  protected void finalize()

Garbage Collection in Java (code #1)

```
class A
{
   A(){System.out.println("Constructor A");}
   protected void finalize(){System.out.println("Finalize A");}
}
public class MyFinalize
{
   static void test()
   {
      A ob=new A();
      ob.finalize(); //finalize() can be called by the object itself
   }
public static void main(String arg[])
   {
      test();
      System.gc();
     //requests the garbage collector to perform cleaning up operations
   }
}
```

Output:

Constructor A
Finalize A
Finalize A

Garbage Collection in Java (code #2)

```
class A
{
   String obname="";
   public A(String nm)       //parameterized constructor function
   {
      obname=nm;
      System.out.println("Class-A constructor called for object: "+obname);
   }
   public void finalize()     //garbage collector function
   {
      System.out.println("Class-A finalize called for object: "+obname);
   }
}
class B
{
   public B()
   {
      A ob1=new A("OB1");
      A ob2=new A("OB2");
   }
}
```

```
public class MyClassObject
{
   public static void main(String args[])
   {
      B ob=new B();
      System.gc();
   }
}
```

Array of Objects

- An array of objects is created using the following syntax:

  Class-name arrayname[]=new class-name[size];

  Or

  Class-name[] arrayname=new class-name[size];

- When an array of objects is created then the objects as elements in the array are not instantiated by themselves, hence they cannot be used to invoke the members.
- So, let there be class A that contains a member method test() and we are creating an array of objects of this class,

  then we would write A ar[]=new A[5]; and to invoke the member method we can use ar[2].test();, this would generate an error, because the elements in the array are not instantiated.
- All the object elements in the array are to be instantiated individually using the new operator and constructor method.

Array of Objects

```
class MyClass {
    MyClass()
            {System.out.println("Construtor Called");}
    void test()
            {System.out.println("test() Called");}
}
public class NewClass {
    public static void main(String arg[]){
        MyClass ar[]=new MyClass[5];
        for(int i=0;i<ar.length;i++){
            ar[i].test();
        }
    }
}
```

Output:

```
Exception in thread "main"
java.lang.NullPointerException at
NewClass.main(NewClass.java:13)
Java Result: 1
```

The output to this program displays a NullPointerException as no constructors were called, hence the object elements are not instantiated. So we cannot call any members from any element in the array.

Corrected Program Code

The use of "final" keyword

- The keyword final is used to declare constants.
- The use of final along with a variables makes the assigned value as constant or immutable.

Example: final int a=100; would make a's value as 100 and cannot be changed in the entire program but can be inherited in child classes.

- The use of final along with the class declaration would not allow inheritance of the class. So other classes cannot inherit a final class.

Example: final class A{...}, now any other class cannot inherit this class-A.

- The use of final along with method declaration would not allow that method to be over-ridden by the child classes.

Example: class A{ **public final void test(){...}},** so if at all any other class inherits A then method test() cannot be re-written (overridden) in the child classes.

Activity: Create your own code to test the final keyword

The use of “this” keyword

- Usage of this keyword
- • to refer the current class instance variable
- • to invoke the current class constructor
- • to invoke the current class method
- The this keyword can be used to refer current class instance variable.
- Let's prove that this keyword refers to the current class instance variable. In the following program, we are printing the reference variable and this, output of both variables are same.

The use of “this” keyword

```
class A{
  void m(){
        System.out.println(this);//prints same reference ID
  }

  public static void main(String args[]){
        A obj=new A();
        System.out.println(obj);//prints the reference ID
        obj.m();
  }
  }
```

Output:

```
A@13d9c02
A@13d9c02
```

The use of “this” keyword

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

        Student(int id,String name){
        this.id = id;
        this.name = name;
        }
        void display(){System.out.println(id+" "+name);}
        public static void main(String args[]){
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        s1.display();
        s2.display();
    }
    }
```

Output

111 Karan

222 Aryan

Point to ponder:

What will happen if in this code we do not use “this” ?

this() can be used to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor (constructor chaining).
- This approach is better if we have many constructors in the class and want to reuse that constructor.
- **Rule:** Call to this() must be the first statement in constructor only.

this() can be used to invoke current class constructor

```
class Student{
    int id;
    String name;
    Student (){System.out.println("default constructor is invoked");}

    Student(int id,String name){
    this ();                    //it is used to invoked current class constructor.
    this.id = id;
    this.name = name;
    }
    void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
    Student e1 = new Student(111,"karan");
    Student e2 = new Student(222,"Aryan");
    e1.display();
    e2.display();
   }
  }
```

Output:

default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan

Point to ponder:

How would you invoke a parameterized constructor using "this" ?

The “this” keyword can be used to invoke current class method (implicitly).

- We may invoke the method of the current class by using the “this” keyword.
- If we don't use the “this” keyword then also the compiler automatically adds this keyword while invoking the method.

The "this" keyword can be used to invoke current class method (implicitly).

```
class S{
   void m(){
   System.out.println("method is invoked");
   }
   void n(){
   this.m();      //no need because compiler does it implicitly.
   }
   void p(){
   n();           //complier will add this to invoke n() method as this.n()
   }
   public static void main(String args[]){
   S s1 = new S();
   s1.p();
   }
 }
```

Output: method is invoked

Point to ponder:
How would you invoke a parameterized member method using "this" ?

The "static" keyword

- The static keyword is used in java mainly for memory management. We may apply static keyword with ***data-members, methods, blocks and nested class***.
- The static keyword makes the member belong to the class rather than instance of the class.
- The static variable will get the memory only once, if any object changes the value of the static variable, it will retain its modified value for all objects that try to access it.
- There are three properties of static context. They are:
  - The static method cannot use non static data member or call non-static method directly.
  - But non-static member methods can use static members.
  - ***this*** and ***super*** cannot be used in static context.

The “static” keyword

```
class Counter{
    static int count=0;
            //will get memory only once and retain its value

    Counter(){
    count++;
    System.out.println(count);
    }

    public static void main(String args[]){

    Counter c1=new Counter();
    Counter c2=new Counter();
    Counter c3=new Counter();

    }
}
```

Output:
1
2
3

Point to ponder:
What would happen if we don’t use the static in the declaration of count ?

Static method

```
//Program to get cube of a given number by static method

class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);
      // calling a static method directly using the class name.
  System.out.println(result);
  }
}
```

Output: 125

Point to ponder:
Observe what happens when we call a static method using an object

The “static” block

- It is used to initialize the static data member.
- It is executed before main method at the time of class-loading.

Example of static block

```
class A{

  static{System.out.println("static block is invoked");}

  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Output:
static block is invoked
Hello main

The "static" control flow

- **Step-1:** Identification of static members from top to bottom.
- **Step-2:** Execution of static variable assignments and static blocks from top to bottom.
- **Step-3:** Execution of main() method.

Can you guess the output ?

```
public class NewClass
{
   static int i=10; //static variable
   static //1st static block
   {
      m1();
      System.out.println("1st static block");
   }
   public static void main(String arg[])
   {
      m1();
      System.out.println("main method");
   }
   static void m1() //static method
      {
         System.out.println("j= "+j);
      }

   static //2nd static block
      {
         System.out.println("2nd static block");
      }
      static int j=20; //static variable
}//end of NewClass
```

Output

```
j= 0
1st static block
2nd static block
j= 20
main method
```

Point to ponder:
Analyze the output using the information that was given in the Static Control Flow section

The “instance” control flow

- **Step-1:** All the steps of static control flow.
- **Step-2:** When an instance of the class gets created: Identification of instance members from top to bottom.
- **Step-3:** Execution of instance variable assignments and instance blocks from top to bottom.
- **Step-4:** Execution of constructor method.

Can you guess the output ?

```
public class NewClass
{
   int i=10;
   {      //first instance block
      m1();
      System.out.println("1st Instance block");
   }
   NewClass()
   {
      System.out.println("i= "+i);
      System.out.println("j= "+j);
      System.out.println("Constructor");
   }
   public static void main(String arg[])
      {
         NewClass ob=new NewClass();
         System.out.println("main method");
      }
      void m1()
      {
         System.out.println("i= "+i);
         System.out.println("j= "+j);
      }
      {      //second instance block
         System.out.println("2nd Instance block");
      }
      int j=20;
} //end of NewClass
```

Output

```
i= 10
j= 0
1st Instance block
2nd Instance block
i= 10
j= 20
Constructor
main method
```

Point to ponder:
Analyze the output using the information that was given in the Instance Control Flow section

End of UNIT-I