## 1.what are the supported Platforms by Java programming Language?

Ans1-  Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX/Linux like HP-Unix, Sun Solaris, Linux, Ubuntu, CentOS, Redhat etc.

**2.** Explain Five Important Features of Java?

### Ans2.1 simple

The Java language is easy to learn and its coding style is easy to read and write. It contains many features of other Languages like C and C++ and Java removes complexity because it doesn't use pointers and doesn't support Multiple Inheritance.

### 2.2 Platform Independent

This is where the "Write Once, run anywhere" motto for Java comes in. It means that you can develop on one environment(Operating System) and run on another environment without doing any modification of the code

### 2.3Architectural Neutral

Java application runs the same bytecodes regardless of any environment (Operating System). To enable a Java application to execute anywhere on the network, the compiler generates an architecture-neutral object file format.

### 2.4Dynamic and Extensible

Java has Dynamic and Extensible means with the help of OOPS you can add classes and plug in new methods to classes, creating new classes through sub classes. This makes Java very easy to augment with your own classes or even to modify.

### 2.5 Portable

Java programs can execute in any environment (Linux,Window,Mac etc.) for which there is a Java run-time system (JVM). That really means there is no implementation dependent features.

### 3.Describe various types of variable used in java with example also explain their scope in the program?

## Instance Variable

A variable which is declared inside the class but outside the method is called instance variable. It is not declared as static.

**members of the class:**

the methods and variables defined within a class are called members of the class.

**Why the name is instance:**Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another.

**Important Points about Instance Variable:**

* Instance variables are used in a class, but outside a method, constructor or any block.
* A slot for each instance variable value is created, when space is allocated for an object in the heap.
* Instance variables can be declared in the class level before or after use.

## Static variable

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

* Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
* There would only be one copy of each class variable per class, regardless of how many objects are created from it.Static variables store values for the variables in a common memory location. Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected.
* Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
* Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.

## Local Variable

A variable which is declared inside the methods, constructors, or blocks is called local variable.

* Local variables are declared in methods, constructors, or blocks.
* Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
* Access modifiers cannot be used for local variables.
* Local variables are visible only within the declared method, constructor, or block.
* Local variables are implemented at stack level internally.
* There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

4.Explain modifiers with example?

Ans4-Access modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

### **1.privateAccessModifier**

|  |
| --- |
| The private access modifier is accessible only within class. |

### **Simple example of private access modifier**

**class** A{  **private** **int** data=40;

**private** **void** msg(){System.out.println("Hello java");}

}

**public** **class** Simple{

**public** **static** **void** main(String args[]){

   A obj=**new** A();

   System.out.println(obj.data);//Compile Time Error

   obj.msg();//Compile Time Error

   }

}

### **2.default access modifier**

|  |
| --- |
| If you don't use any modifier, it is treated as **default** bydefault.  The default modifier is accessible only within package. |

### **Example of default access modifier**

**package** pack;

**class** A{

**void** msg(){System.out.println("Hello");}

}  **package** mypack;

**import** pack.\*;

**class** B{

**public** **static** **void** main(String args[]){

   A obj = **new** A();//Compile Time Error

   obj.msg();//Compile Time Error

  }

}

### **3) protected access modifier**

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

### **Example of protected access modifier**

### **package** pack;

**public** **class** A{

**protected** **void** msg(){System.out.println("Hello");}

}

**package** mypack;

**import** pack.\*;

**class** B **extends** A{

**public** **static** **void** main(String args[]){

   B obj = **new** B();

   obj.msg();

  }

}

### **4) public access modifier**

|  |
| --- |
| The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers. |

### **Example of public access modifier**

**package** pack;

**public** **class** A{

**public** **void** msg(){System.out.println("Hello");}

}

**package** mypack;

**import** pack.\*;

**class** B{

**public** **static** **void** main(String args[]){

   A obj = **new** A();

   obj.msg();

  }

}

Let's understand the access modifier

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

5.Describe various uses of final keyword in java with example?

# **Final Keyword In Java**

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

## 1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

### **Example of final variable**

**class** Bik9{

**final** **int** speedlimit=90;//final variable

**void** run(){

  speedlimit=400;

 }

**public** **static** **void** main(String args[]){

 Bik9 obj=**new**  Bik9();

 obj.run();

 }

}

## 2. Java final method

If you make any method as final, you cannot override it.

### **Example of final method**

**class** Bike{

**final** **void** run(){System.out.println("running");}

}

**class** Honda **extends** Bike{

**void** run(){System.out.println("running safely with 100kmph");}

**public** **static** **void** main(String args[]){

   Honda honda= **new** Honda();

   honda.run();

   }

}

## 3) Java final class

If you make any class as final, you cannot extend it.

### **Example of final class**

**final** **class** Bike{}

**class** Honda1 **extends** Bike{

**void** run(){System.out.println("running safely with 100kmph");}

**public** **static** **void** main(String args[]){

  Honda1 honda= **new** Honda1();

  honda.run();

  }

}

6.What are tokens Name types of tokens available in java?

Java language includes five types of tokens:

|  |
| --- |
| Smallest individual units in a program are known as tokens. |
| In simple, a Java program is a collection of tokens, comments and white space. |

1)  Reserved keywords

2) Identifiers

3) Literals

4) Operators

5) Separators

|  |  |
| --- | --- |
| **Keywords :** | |
| Keywords are an essential part of a language definition. | |
|  | Java language has reserved words as keyword. |
|  | Keywords have specific meaning in Java we cannot use them as names for variables, classes, methods and so … on. |
|  | All keywords are to be written in lower case letters. |
|  | Java is case sensitive. |
| **Identifies :** | |
|  | Identifiers are programmer designed tokens. |
|  | They are used for naming classes, methods, variable objects, labels, packages and interfaces in a program. |
|  | Java identifiers follow the following rules. |
|  | 1) They can have alphabets, digits & underscore & dollar sign characters.    2) They must not begin with a digit.    3) Upper case & lower case letters are distinct.    4) They can be of any length. |
| **Literal :** | |
| Literals in Java are a sequence of character (digits, letters and other character) that represent constant values to be stored in variables. | |
|  | Integer literals  Floating point literals  Character literals |
|  | String literals  Boolean literals |
| **Operators :**    An operator is a symbol that takes one or more argyments and operates on them to produce a result. | |
| **Separators :**    Separators are symbols used to indicate where groups of code are divided and arranged. | |
| **Parentheses ( ) :** Used to enclosed parameters in method definition.    **Braces { }:** | |
| 1. | Define a block of code for classes methods and local scope. |
| 2. | Automatically initialize values to array.  7. Difference between rint() and round() function in java? |

Ans7-Math.rint(double) returns a double value closest to the argument that is passed to it and equal to the mathematical integer. Math.round(double) returns a long value closest to the value passed to it   
This code demonstrates the thing

|  |  |  |
| --- | --- | --- |
| class RoundRint | | |
|  | { |

|  |  |  |
| --- | --- | --- |
|  | public static void main(String[] args) | |
|  | { |

|  |  |  |
| --- | --- | --- |
|  | double val = 424.122262; | |
|  |  |

|  |  |
| --- | --- |
|  | System.out.println(Math.rint(val)); |
|  | System.out.println(Math.round(val)); | |

}}

8.Write example programs showing the difference between call by value and call by reference?

### Ans8- **Example of call by value in java**

**class** Operation{

**int** data=50;

**void** change(**int** data){

 data=data+100;//changes will be in the local variable only

 }

**public** **static** **void** main(String args[]){

   Operation op=**new** Operation();

   System.out.println("before change "+op.data);

 op.change(500);

   System.out.println("after change "+op.data);

 }

}

Example of Call by Reference

class Number {  
   int x;  
}  
public class Reference {  
 public static void main ( String[] args ) {  
      Number a = new Number();  
      a.x=4;  
      System.out.println(a.x);  
      Number b=a;  
      b.x=5;  
      System.out.println(b.x);  
   }  
}

9.Name and give example of type of constructors?

Ans9-name and Types of constructors are as follows

### No-Arg Constructor

If a Java constructor does not accept any parameters, it is a no-arg constructor. It's syntax is:

class NoArgCtor {

int i;

// constructor with no parameter

private NoArgCtor(){

i = 5;

System.out.println("Object created and i = " + i);

}

public static void main(String[] args) {

NoArgCtor obj = new NoArgCtor();

}

}

### Default Constructor

If you do not create constructors yourself, the Java compiler will automatically create a no-argument constructor during run-time. This constructor is known as default constructor. The default constructor initializes any uninitialized instance variables.

### Example: Default Constructor

class DefaultConstructor { int a;

boolean b;

public static void main(String[] args) {

DefaultConstructor obj = new DefaultConstructor();

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

### Parameterized Constructor

A constructor may also accept parameters

### Example: Parameterized constructor

class Vehicle {

int wheels;

private Vehicle(int wheels){

wheels = wheels;

System.out.println(wheels + " wheeler vehicle created.");

}

public static void main(String[] args) {

Vehicle v1 = new Vehicle(2);

Vehicle v2 = new Vehicle(3);

Vehicle v3 = new Vehicle(4);

}

}

10.what are access specifier? Draw a table showing all the access specifier and their accessibility in class package subclasses and other packages?

Ans10- An access specifier is a defining code element that can determine which elements of a program are allowed to access a specific variable or other piece of data. Different programming languages have their own protocols for access specifiers, as well as defaults for some code elements including both individual variables and classes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **default** | **private** | **protected** | **public** |
| Same package (or same class) | Yes | Yes | Yes | Yes |
| Same package subclass | Yes | No | Yes | Yes |
| Same package non subclass | Yes | No | Yes | Yes |
| Different package sub class | No | No | Yes | Yes |
| Different package non subclass | No | No | No | Yes |

11.What are Wrapper classes?What is it’s advantages?also explain some function of it?

# Ans11- **Wrapper Classes in Java**

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

**Need of Wrapper Classes**

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) and [Vector](https://www.geeksforgeeks.org/vector-vs-arraylist-java/), store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

**Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

Example of Autoboxing   
import java.util.ArrayList;

class Autoboxing

{

    public static void main(String[] args)

    {

        char ch = 'a';

        // Autoboxing- primitive to Character object conversion

        Character a = ch;

        ArrayList<Integer> arrayList = new ArrayList<Integer>();

        // Autoboxing because ArrayList stores only objects

        arrayList.add(25);

        // printing the values from object

        System.out.println(arrayList.get(0));

    }

}

**Unboxing:** It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Example of unboxing

import java.util.ArrayList;

class Unboxing

{

    public static void main(String[] args){

        Character ch = 'a';

   // unboxing - Character object to primitive conversion

        char a = ch;

 ArrayList<Integer> arrayList = new ArrayList<Integer>();

        arrayList.add(24);

        // unboxing because get method returns an Integer object

        int num = arrayList.get(0);

        // printing the values from primitive data types

        System.out.println(num);

    }

}

12.what is fall-through and how it can be prevented?Explain with example

Ans12- **About Fall Through Switch Statements In Java :**   
It is not necessary for every case in the [switch Statement In Java](http://java.meritcampus.com/core-java-topics/switch-statement-in-java) to have a break. Some times we might have a situation where we have the same or similar handling for multiple cases.

class AlphabetsClassification  
{  
    public static void main(String arg[])  
    {  
        char alphabet = 'A';  
          
        switch(alphabet)  
        {  
            case 'A':  
                System.out.println("The alphabet '" + alphabet + "' is in first four letters");  
                // LINE A  
            case 'E':  
            case 'I':  
            case 'O':  
            case 'U':  
                System.out.println("The alphabet '" + alphabet + "' is an vowel");  
                break;  
            case 'B':  
            case 'C':  
            case 'D':  
                System.out.println("The alphabet '" + alphabet + "' is in first four letters");  
            default:  
                System.out.println("The alphabet '" + alphabet + "' is a consonant");  
        }      
    }  
}

13.can a constructor be made final?

Ans13-***When constructors are not inherited, no meaning of overriding the constructors. In Java, final is used by super class not to inherit its members by subclass. When constructor is not inherited (and subclass cannot override), then why a constructor should be declared final. Already, internally a constructor property is final. Why again, final is required. For this reason, constructor cannot be declared final. If*it is declared as *final, compiler raises error.*Finally to say, a constructor cannot be final as it is never inherited.**

14.why collection framework used in java?

Ans14-Java collection are used in every programming language and initial java release contained few classes for collections: Vector, Stack, Hashtable, Array. But looking at the larger scope and usage, Java 1.2 came up with Collections Framework that group all the collections interfaces, implementations and algorithms.

Java Collections have come through a long way with usage of Generics and Concurrent Collection classes for thread-safe operations. It also includes blocking interfaces and their implementations in java concurrent package.

Some of the benefits of collections framework are:

* Reduced development effort by using core collection classes rather than implementing our own collection classes.
* Code quality is enhanced with the use of well tested collections framework classes.
* Reduced effort for code maintenance by using collection classes shipped with JDK.
* Reusability and Interoperability

15.Describe vector class with an example?

##### Ans15- Vector class Java is the general purpose Data Structure used very often by the Programmers. Vector, though belongs to legacy classes (DS of JDK 1.0), it can use the methods of **List** interface as Vector implements List from JDK 1.2. In this way, Vector became part of [collections framework](http://way2java.com/collections/java-collections-framework/).

Import java.util.\*;

public class PrintElements

{

public static void main(String args[])

{

   Vector vect1 = new Vector();

   vect1.add("Jyostna");

   vect1.add(100);

   vect1.add(10.5);

   vect1.add(new Date());

   Enumeration e = vect1.elements();

   System.out.println("Vector Elements:");

   while(e.hasMoreElements())

   {

     System.out.println(e.nextElement());

   }

}

}