**keywords**

There are 50 keywords in java. They are as follows

1. **Abstract:**

The keyword abstract can be used on class declaration known as abstract class. The keyword abstract can be used on method declaration can be known as abstract method. When a class is declared as abstract it cannot be instantiated. When a method is declared as abstract, it cannot be defined. Only an abstract class should have abstract methods. Abstract class does not necessarily require it’s all methods to be abstract.

The purpose of abstract class is to serve purely as a parent class. The difference between abstract class and normal class is that, the abstract class methods cannot have definitions.

The syntax for abstract class and abstract keyword is as follows.

abstract class Test // abstract class

{

abstract void display(); // abstract method

}

**Note:**

* Variables cannot be abstract.
* Constructors cannot be abstract.
* We can define a static method in abstract class.
* When a class extends an abstract class, it is said to implement the abstract class.
* We cannot create object to abstract class.
* The main purpose of constructor is to perform initialization of an object. i.e. to perform initialization for instance variables. Abstract class can contain instance variables which are required for child class object to perform initialization for these instance variables, constructor concept is required for abstract classes.
* Every variable present inside interface is always public static final whether we are declaring or not. Hence there is no chance of existing instance variables, inside interface. Because of this constructor concept not required for interfaces.

1. **Assert**

An assertion allows testing the correctness of any assumptions that have been made in the program. Assertion is achieved using the assert statement in Java. While executing assertion, it is believed to be true. If it fails, JVM throws an error named AssertionError. It is mainly used for testing purposes during development.

The assert statement is used with a Boolean expression and can be written in two different ways.

assert expression;

Or

assert expression1: expression2;

for example:

class Test

{

Public static void mian(String args[])

{

Int value=15;

Assert value>= 20:”underweight”;

System.out.println(“the value is “+value);

}

}

When we run the code normally we get output as 15.

When we enable assertion we get an exception saying “Exception in thread "main" java.lang.AssertionError: Underweight”

We can enable assertion as follows

java –ea Test

or

java –enableassertions Test

**why?**

* To make sure that assumptions written in comments are right.
* To make sure default switch case is not reached.
* To check object’s state in the beginning of the method after method invocation.

1. **Boolean**

Boolean is a keyword which designates the boolean primitive type. There are only two possible boolean values. True and false. Default is false. This keyword can be used with variables, method parameters and method return types.

The size of boolean is not precisely defined and is dependent upon the JVM.

1. **Break**

The break statement in java has two usages.

* When the break statement is encountered inside the loop, it is immediately terminated and the program control resumes at the next statement following the loop.
* It can be used to terminate a case in the switch statement.

**For example:**

class Test

{

public static void main(String args[])

{

int[] a={10,20,30,40,50};

for(int i:a)

{

if(i==30)

bsreak;

System.out.println(i+”\n”);

}

}

}

Here when the I value reaches to 30 it terminates and exists the loop.

**Output:**

10

20

1. **Byte**

Byte keyword is used to declare a numeric value. A byte value can hold an 8-bit integer number which ranges from -128 to 127.

1. **Case**

Case keyword is used with switch keyword in java. It is used to specify the code block which needs to be executed in case of specific values which are passed in switch statement.

It is followed by the value to be compared and a colon.

**For example**,

switch(i)

{

case 1:

System.out.println(“this is 1”);

break;

case 2:

System.out.println(“this is 2”);

break;

default:

Sysem.out.println(“not a valid number”);

break;

}

1. **Catch**

Catch is used in conjunction with a try block and an optional finally block. The statements in the catch block specify what to do if a specific type of exception is thrown by the try block.

**Example:**

try

{

int 1=10/0;

System.out.println(i);

}

catch(ArithmeticException e)

{

System.out.println(e);

}

There are two possibilities of catch blocks.

Multiple catch blocks and multiple exceptions in single catch block

**Multiple catch blocks:**

We can write multiple catch blocks for a single try block. Incase, exception match with one of the exception in the catch blocks, the exception will be handled there.

**For example,**

try

{

int I;

System.out.println(i/0);

}

catch(ArithmeticException e)

{

System.out.println(e);

}

catch(NullPointerException e)

{

System.out.println(e);

}

**Multiple exceptions in single catch block:**

we can catch multiple exceptions in a single catch block.

catch(ArithemeticException | NullPointerException e)

1. **Char**

Char keyword is used to declare a variable as character type. A char variable represents a single character. Char can be created from character literals, numeric representation as below.

char c=’a’;

char c2=65;

1. **Class**

In java, class keyword is used to declare a class or define a class. The general syntax is as follows.

[access-specifier] class [identifier]

{

}

1. **Continue**

Continue keyword in java is used to skip the remaining part of the loop statements and and continue with the next iteration of the loop.

**Example:**

public class Continue

{

public static void main(String args[])

{

for(int i=0;i<10;i++)

{

if(i==5)

continue;

System.*out*.println(i);

}

}

}

**Output:**

0

1

2

3

4

6

7

8

9

1. **Default**

Default keyword is used in the following ways.

* Optionally be used in switch statement.

**Example:**

switch(i)

{

case 1:

System.out.println(“this is 1”);

break;

case 2:

System.out.println(“this is 2”);

break;

default:

Sysem.out.println(“not a valid number”);

break;

}

Here when the value in switch does not match with any other case condition, then default block gets executed.

* Can be used to declare default values in java annotations.
* From java 8 onwards, can be used in interface to introduce methods with implementation.

Default methods enable you to add new functionality to the interface of your library (API) and ensure binary compatibility with code written for older version of that interface.

**For example:**

interface Test

{

void show();

default void display()

{

System.out.println(“this is display of interface”);

}

}

class Testing extends Test

{

Public void show()

{

System.out.println(“this is display of main”);

}

Testing t=new Testing();

t.show();

t.display();

}

**Output:**

This is display of main

This is display of interface

1. **Do**

Do keyword is used in control statements to declare a loop. It provides a repetitive tasks as long as the condition specified with the while keyword is true.

Do while loop comes into the category of exit control loop, where the condition is checked after the execution of the loop.

**Example:**

int i=1;

do

{

System.out.println(i)

}while(I<10);

1. **Double**

Double is one of the java primitive types. It is used for storing double precisions 64-bit of information for a float type.

**Syntax;**

double x=22.22;

The Double wrapper class defines the constants MIN\_VALUE, MAX\_VALUE, NEGATIVE\_INFINITY, POSITIVE\_INFINITYand NaN.

1. **Else**

else keyword in java introduces statements that are executed when the condition in an if statement isn’t true.

The else keyword is always used in association with the if keyword in an if−else statement. The else clause is optional and is executed if the if condition is false.

1. **Enum**

**enum** is java keyword to declare enum types. It is a special data type that enables for a variable to be a set of predefined constants.

Syntax for declaring enum is as below:

public enum Day {

    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

}

In java, enum types are automatically converted to class type in compilation time. Compiler automatically adds few methods while converting enum to class. These methods are:

* public static T[] values() – returns an array containing all of the values of the enum in the order they are declared.
* public static T valueOf(String s) – returns enum constant for which string value is equal to ‘s’.

1. **Extends**

The extends is a Java keyword, which is used in inheritance process of Java. It specifies the super class in a class declaration using extends keyword. It is a keyword that indicates the parent class that a subclass is inheriting from and may not be used as identifiers i.e. you cannot declare a variable or class with this name in your Java program.

**Example:**

class A

{

}

class B extends A

{

}

Extends keyword is also used in an interface declaration to specify one or more super interfaces.

**Example:**

interface A

{

}

interface B extends A

{

}

1. **Final**

Final keyword is used in different contexts. First of all, final is a non-access modifier applicable only to a variable, a method or a class.

When a variable is declared with final keyword, its value can’t be modified, essentially, a constant. A final variable can only be initialized once, either via an initializer or an assignment statement. There are three ways to initialize a final variable:

* You can initialize a final variable when it is declared. This approach is the most common. A final variable is called blank final variable, if it is not initialized while declaration. Below are the two ways to initialize a blank final variable.
* A blank final variable can be initialized inside instance-initializer block or inside constructor. If you have more than one constructor in your class then it must be initialized in all of them, otherwise compile time error will be thrown.
* A blank final static variable can be initialized inside static block.

When a class is declared with final keyword, it is called a final class. A final class cannot be extended(inherited).

When a method is declared with final keyword, it is called a final method. A final method cannot be overridden.

1. **Finally**

Finally creates a block of code that will be executed after a try/catch block has completed and before the code following the try/catch block. The finally block will execute whether or not an exception is thrown.

The finally clause is optional

1. **Float**

In java, float keyword is used to store the 32-bit float primitive types. Floats are generally used when evaluating expressions that require single fractional precision.

**Syntax:**

Float a=22.2;

Float class has following fields :

**Field Description:**

MAX\_EXPONENT - Maximum exponent a finite float variable may have.

MAX\_VALUE - A constant holding the largest positive finite value of type float, (2-2-23)·2127.

MIN\_EXPONENT - Minimum exponent a normalized float variable may have.

MIN\_NORMAL - A constant holding the smallest positive normal value of type float, 2-126.

MIN\_VALUE - A constant holding the smallest positive nonzero value of type float, 2-149.

NaN - A constant holding a Not-a-Number (NaN) value of type float.

NEGATIVE\_INFINITY - A constant holding the negative infinity of type float.

POSITIVE\_INFINITY - A constant holding the positive infinity of type float.

1. **For**

For is a keyword used for executing the same code many times using looping concept.

**Syntax:**

for(int i=0;i<10;i++)

The syntax consists of three phases, like initialization, condition and increment or decrement.

For looping an array, we can do as follows

int[] a={10,20,30,40};

for(int a1:a)

System.out.println(a1);

1. **Goto**

A goto statement provides an unconditional jump from the goto to a labeled statement in the same function.

The syntax for a goto statement in Go is as follows −

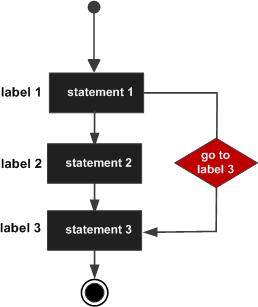
goto label;

label: statement;

Here, label can be any plain text except Go keyword and it can be set anywhere in the Go program above or below to goto statement.

Java doesn’t support goto.

**Flow diagram:**



1. **If**

If is used in the selection statements. If is a block executed when condition specified with the if clause is satisfied with the scenario.

**Syntax:**

if(condition)

{

}

1. **Implements**

Implements means you are using the elements of a Java Interface in your class. It is used to implement an interface (parent class with functions signatures only but not their definitions) by defining it in the child class.

**Example:**

interface A

{

}

class B implements A

{

}

**Difference between extends and implements:**

Generally, implements used for implementing an interface and extends used for extension of base class behavior or abstract class.

**extends:** A derived class can extend a base class. You may redefine the behavior of an established relation. Derived class "is a" base class type

**implements:** You are implementing a contract. The class implementing the interface "has a" capability.

1. **Import**

The import statement in Java allows to refer to classes which are declared in other packages to be accessed without referring to the full package name. You do not need any import statement if you are willing to always refer to java.util.List by its full name, and so on for all other classes. But if you want to refer to it as List, you need to import it, so that the compiler knows which List you are referring to.

1. **instanceOf**

The java instanceof operator is used to test whether the object is an instance of the specified type. The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

**Example:**

public class InstanceOf

{

public static void main(String args[])

{

InstanceOf io=new InstanceOf();

System.out.println(io instanceof InstanceOf);

}

}

1. **int**

int is a primitive type. Variables of type int store the actual binary value for the integer you want to represent. An int variable holds a 32-bit signed integer value.

**Syntax:**

int a=10;

1. **interface**

Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract. Interfaces specify what a class must do and not how. It is the blueprint of the class. If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.

**Syntax:**

interface A

{

public void display();

}

* It is used to achieve total abstraction.
* Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance.
* Interface abstract classes may contain non-final variables, whereas variables in interface are final, public and static.es are used to implement abstraction.
* We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
* A class can implement more than one interface.
* An interface can extends another interface or interfaces (more than one interface) .
* A class that implements interface must implements all the methods in interface.
* All the methods are public and abstract. And all the fields are public, static, and final.
* It is used to achieve multiple inheritance.
* It is used to achieve loose coupling.

1. **Long**

Long is a keyword in java that is used to store 64-bit integer. The keyword long in java is also used to declare an expression, method return value, or variable of type long integer. The Long class in java programming language wraps a long type primitive value in an object. An Long type object includes only a single field whose type is long

The Long class in java programming language is a wrapper class in order to support the long primitive type. Constants MIN\_VALUE and MAX\_VALUE are used to represent the range of values for this type.

In java programming language all integer literals are 32−bit int values unless the value is followed by l or L as in 235L, which specifies that the value should be interpreted as a long.

1. **native**

Native is a modifier applicable only for methods and we can’t apply anywhere else. The methods which are implemented in non-java (mostly C or C++) are called native methods or foreign methods.

The main objectives of native keyword are

* To improve performance of the system

Where the java is performance wise in critical position, that functionality could be developed in C or C++ and can bring that functionality into java using native keyword.

* To achieve machine level or memory level communication

Java is simple. Java is programmer friendly but not machine friendly language. But C is machine friendly and not programmer friendly. Java cannot communicate directly with machine. Thus for memory level and machine level, java is incapable of developing system. So we go for non-java for this purpose

Example: hash code method

Public native int hashCode ()

* To use already existing legacy non java code

Pseudocode to use native keyword in java.

class Native

{

static

{

System.loadLibrary(“native library path”); (1) load native library

}

public native void m1(); (2) declare a native method

}

class Client

{

public static void main(String args[])

{

Native n=new Native();

n.m1(); (3) invoke a native method

}

}

For native methods, implementations are already available in old languages like C or C++ and we are not responsible to provide implementation. Hence, native method declaration should end with semi colon.

Example:

public native void m1()

public native void m1() {} // exception “native method cannot have a body”

For native implementation is already available in old languages. But for abstract methods implementation should not be available. Hence we can’t declare native method as abstract. i.e. native abstract combination is illegal combination for methods.

We can’t declare native method as strictfp, because there is no guarantee that old languages follow IEEE 754 standard. Hence native strictfp combination is illegal combination for methods.

The main advantage of native keyword is performance will be improved, but the main disadvantage of native keyword is it breaks platform independent nature of java.

1. **new**

The main objective of new operator is to create an object. The main purpose of constructor is to initialize object. First object will be created by using new keyword, and then initialization will be performed by constructor.

The new keyword in java is used to create instance of a class. The new keyword allocates a space in the memory and initialize the object.

**Syntax:**

A ob1 = new A (12,56);

Shape ob2 = new Shape(ob,32,98);

First line creates the object of A class and second class create an object of shape class.

Each of these statement has three parts;

* ob1 and ob2 are all variable declaration associated with an object type, this is called declaration.
* Creating an object by new keyword this is nothing but instantiation.
* The new keyword is followed by a call to constructors and which initializes the new object this is nothing but initialization.

While using new keyword some points to be remember:

The new keyword is followed by a class name and passing values to the constructors.

Number of arguments to the constructors must be the same in signature of constructor.

The type of variable on the left side of assignment is compatible with the class being instantiated.

1. **Package**

Package keyword is used to organize your classes into namespaces

so, package has the following uses:

* Group related classes and interfaces together
* Allow classes with same class name to exist in different packages
* If you want certain methods of your class to have restricted access, declaring the access modifier as protected or leaving it as default would make the methods accessible only from within the same package

1. **Private**

Private is an access modifier. The private (most restrictive) fields or methods cannot be used for classes and Interfaces. It also cannot be used for fields and methods within an interface. Fields, methods or constructors declared private are strictly controlled, which means they cannot be accesses by anywhere outside the enclosing class. A standard design strategy is to make all fields private and provide public getter methods for them.

1. **Protected**

The protected fields or methods cannot be used for classes and Interfaces. It also cannot be used for fields and methods within an interface. Fields, methods and constructors declared protected in a superclass can be accessed only by subclasses in other packages. Classes in the same package can also access protected fields, methods and constructors as well, even if they are not a subclass of the protected member’s class.

1. **Public**

Fields, methods and constructors declared public (least restrictive) within a public class are visible to any class in the Java program, whether these classes are in the same package or in another package.

1. **return**

return is a reserved keyword in Java i.e, we can’t use it as an identifier. It is used to exit from a method, with or without a value.

return can be used with methods in two ways:

**Methods returning a value:** For methods that define a return type, return statement must be immediately followed by return value.

**Methods not returning a value:** For methods that don’t return a value, return statement can be skipped.

Return statement can be used at various places in the method but we need to ensure that it must be the last statement to get executed in a method.

Note: return statement need not to be last statement in a method, but it must be last statement to execute in a method.

1. **Short**

The short keyword is used to declared a variable as a numeric type. A short value can hold a 16-bit integer number which ranges from -32,768 to 32,767. For example:

short length = 20120;

The short keyword can be used to declared return type of a method as well:

public short getLength()

{

return 20120;

}

1. **Static**

static is a non-access modifier in Java which is applicable for the following:

* blocks
* variables
* methods
* nested classes

To create a static member (block, variable, method, nested class), precede its declaration with the keyword static. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. For example, in below java program, we are accessing static method m1() without creating any object of Testclass.

class test

{

public static void main(String args[])

{

m1();

}

public static void m1()

{

System.out.println(“hello”);

}

}

When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**Example:**

class Student

{

private int roll;

private String name;

static String schlname;

static

{

Schlname=”sss”;

}

public Student (int roll, String name)

{

this.roll=roll;

this.name=name;

}

public void display ()

{

System.out.println(roll+” “+name+” “+schlname);

}

}

public class Demo

{

public static void mainString args[])

{

Student s1=new Student(101,”khan”);

Student s2=new Student(102,”john”);

s1.display();

s2.display();

}

}

**Output:**

101 khan sss

102 john sss;

Important points for static variables: -

* We can create static variables at class-level only. See here
* static block and static variables are executed in order they are present in a program.
* When a method is declared with static keyword, it is known as static method. The most common example of a static method is main () method. As discussed above, any static member can be accessed before any objects of its class are created, and without reference to any object. Methods declared as static have several restrictions:
* They can only directly call other static methods.
* They can only directly access static data.
* They cannot refer to this or super in any way.
* Use the static variable for the property that is common to all objects. For example, in class Student, all students share the same college name. Use static methods for changing static variables.

1. **Strictfp**

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language have provided the strictfp keyword, so that you get same result on every platform. So, now you have better control over the floating-point arithmetic.

The strictfp keyword can be applied on methods, classes and interfaces.

strictfp class A {}//strictfp applied on class

strictfp interface M {}//strictfp applied on interface

class A {

strictfp void m (){}//strictfp applied on method

}

1. **Super**

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

**Usage of Java super Keyword**

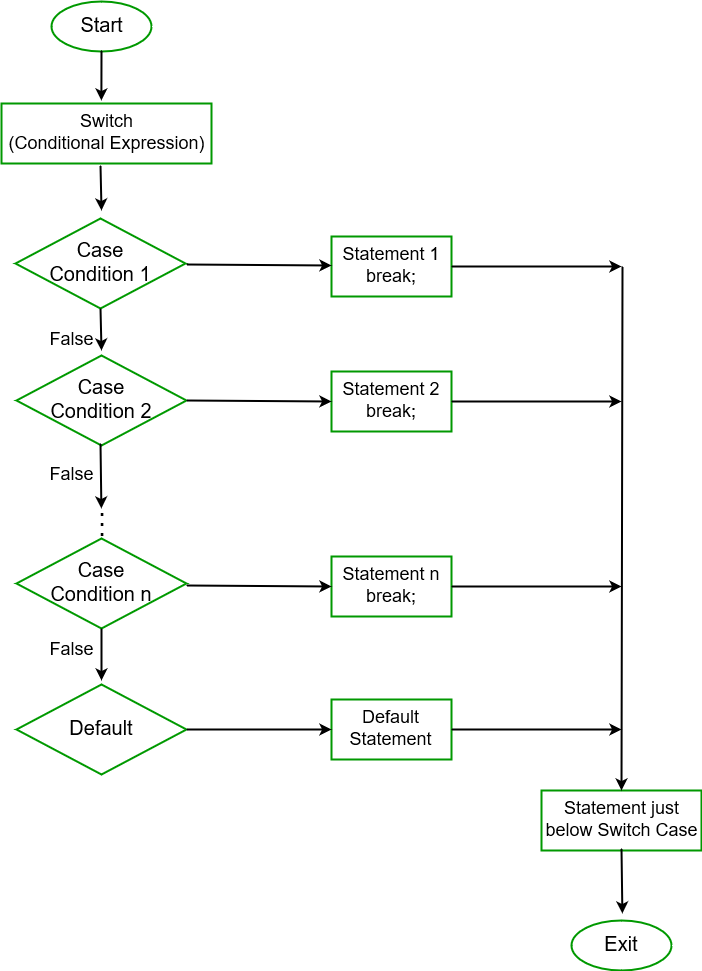
super can be used to refer immediate parent class instance variable.

super can be used to invoke immediate parent class method.

Super () can be used to invoke immediate parent class constructor.

1. **Switch**

The switch statement is a multi-way branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. Basically, the expression can be byte, short, char, and int primitive data types. Beginning with JDK7, it also works with enumerated types (Enum in java), the String class and Wrapper classes.



Some Important rules for switch statements:

* Duplicate case values are not allowed.
* The value for a case must be the same data type as the variable in the switch.
* The value for a case must be a constant or a literal. Variables are not allowed.
* The break statement is used inside the switch to terminate a statement sequence.
* The break statement is optional. If omitted, execution will continue on into the next case.
* The default statement is optional, and can appear anywhere inside the switch block. In case, if it is not at the end, then a break statement must be kept after the default statement to omit the execution of next case statement.

1. **Synchronized**

Java provides a way of creating threads and synchronizing their task by using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

The synchronization is mainly used to

* To prevent thread interference.
* To prevent consistency problem.

There are two types of synchronization

* Process Synchronization
* Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive

* Synchronized method.
* Synchronized block.
* static synchronization.

1. Cooperation (Inter-thread communication in java)

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

* by synchronized method
* by synchronized block
* by static synchronization

**By synchronized method:**

**Example: without synchronized method**

public class SynchronizedKeyword extends Thread

{

public static void main (String args [])

{

Table t=new Table ();

Thread1 t1=new Thread1(t);

Thread2 t2=new Thread2(t);

t1.start();

t2.start();

}

}

class Table

{

public void printTable(int n)

{

for(int i=0;i<5;i++)

{

System.out.println(i\*n);

try

{

Thread.sleep(400);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

}

class Thread1 extends Thread

{

Table t;

Thread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(2);

}

}

class Thread2 extends Thread

{

Table t;

Thread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

**Output:**

0

0

2

5

4

10

6

15

8

20

**With synchronized block:**

public class SynchronizedKeyword extends Thread

{

public static void main(String args[])

{

Table t=new Table();

Thread1 t1=new Thread1(t);

Thread2 t2=new Thread2(t);

t1.start();

t2.start();

}

}

class Table

{

synchronized void printable (int n)

{

for(int i=0;i<5;i++)

{

System.out.println(i\*n);

try

{

Thread.sleep(400);

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

}

class Thread1 extends Thread

{

Table t;

Thread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(2);

}

}

class Thread2 extends Thread

{

Table t;

Thread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

**Output:**

0

2

4

6

8

0

5

10

15

20

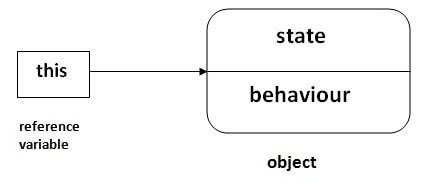
This method seems to do same works as the join() method. But the difference between them is as follows.

join is to be used only when you need to ensure that the thread dies and you need to do something after that. It allows multiple threads to execute on same instance but one by one.

Synchronization prevents multiple threads from executing the synchronized part of code on the same instance.

1. **This**

In java, this is a reference variable that refers to the current object.



Here is given the 6 usage of java this keyword.

* this can be used to refer current class instance variable.
* this can be used to invoke current class method (implicitly)
* this () can be used to invoke current class constructor.
* this can be passed as an argument in the method call.
* this can be passed as argument in the constructor call.
* this can be used to return the current class instance from the method.

**Example:**

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();

}

}

**Output:**

111 ankit 5000

112 sumit 6000

If the local variables and instance variables are different, there is no need of this keyword. See below example.

class Student

{

int rollno;

String name;

float fee;

Student (int r, String n, float f)

{

this.rollno=r;

this.name=n;

this.fee=f;

}

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();

}

}

**Output:**

111 ankit 5000

112 sumit 6000

1. **throw**

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing try block is checked to see if it has a catch statement that matches the type of exception. If it finds a match, controlled is transferred to that statement otherwise next enclosing try block is checked and so on. If no matching catch is found then the default exception handler will halt the program.

**Example:**

// Java program that demonstrates the use of throw

class ThrowExcep

{

static void fun()

{

try

{

throw new NullPointerException("demo");

}

catch(NullPointerException e)

{

System.out.println("Caught inside fun().");

throw e; // rethrowing the exception

}

}

public static void main(String args[])

{

try

{

fun();

}

catch(NullPointerException e)

{

System.out.println("Caught in main.");

}

}

}

1. **throws**

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions.

In a program, if there is a chance of rising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, otherwise we will get compile time error saying unreported exception XXX must be caught or declared to be thrown. To prevent this compile time error, we can handle the exception in two ways:

* By using try catch
* By using throws keyword

**Example:**

// Java program to illustrate error in case

// of unhandled exception

class test

{

public static void main(String[] args)

{

Thread.sleep(10000);

System.out.println("Hello Geeks");

}

}

**Output:**

error: unreported exception InterruptedException; must be caught or declared to be thrown

**Explanation**: In the above program, we are getting compile time error because there is a chance of exception if the main thread is going to sleep, other threads get the chance to execute main () method which will cause InterruptedException.

// Java program to illustrate throws

class tst

{

public static void main(String [] args) throws InterruptedException

{

Thread.sleep(10000);

System.out.println("Hello Geeks");

}

}

Important points to remember about throws keyword:

* throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
* throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
* By the help of throws keyword, we can provide information to the caller of the method about the exception.

1. **Transient**

Transient is a variables modifier used in serialization (mechanism of writing the state of an object into a byte stream). At the time of serialization, if we don’t want to save value of a particular variable in a file, then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type. It is good habit to use transient keyword with private confidential fields of a class during serialization.

A java object is said to be serializable when its class or its super class implement either the java.io.Serializable interface or its sub interface java.io.Externalizable.

**Example:**

// A sample class that uses transient keyword to

// skip their serialization.

class Test implements Serializable

{

// Making password transient for security

private transient String password;

// Making age transient as age is auto-

// computable from DOB and current date.

transient int age;

// serialize other fields

private String username, email;

Date dob;

// other code

}

Since static fields are not part of state of the object, there is no use/impact of using transient keyword with static variables. However, there is no compilation error.

final variables are directly serialized by their values, so there is no use/impact of declaring final variable as transient. There is no compile-time error though.

**Example:**

// Java program to demonstrate transient keyword

// Filename Test.java

import java.io.\*;

class Test implements Serializable

{

// Normal variables

int i = 10, j = 20;

// Transient variables

transient int k = 30;

// Use of transient has no impact here

transient static int l = 40;

transient final int m = 50;

public static void main(String[] args) throws Exception

{

Test input = new Test();

// serialization

FileOutputStream fos = new FileOutputStream("abc.txt");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(input);

// de-serialization

FileInputStream fis = new FileInputStream("abc.txt");

ObjectInputStream ois = new ObjectInputStream(fis);

Test output = (Test)ois.readObject();

System.out.println("i = " + output.i);

System.out.println("j = " + output.j);

System.out.println("k = " + output.k);

System.out.println("l = " + output.l);

System.out.println("m = " + output.m);

}

}

**Output:**

i = 10

j = 20

k = 0

l = 40

m = 50

1. **Try**

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax:

try

{

//code that may throw exception

}

catch(Exception\_class\_Name ref)

{

}

**Or**

try

{

//code that may throw exception

}

finally

{

}

1. **Void**

The void keyword denotes that a method does not have a return type. However, even though a constructor method can never have a return type, it does not have the void keyword in its declaration.

**Example:**

class Test

{

private int roll;

private String name;

public Test(int roll,String name)

{

this.roll=roll;

this.name=name;

}

public void display()

{

System.out.println(roll+” “+name);

}

}

1. **Volatile**

Using volatile is yet another way (like synchronized, atomic wrapper) of making class thread safe. Thread safe means that a method or class instance can be used by multiple threads at the same time without any problem.

1. **While**

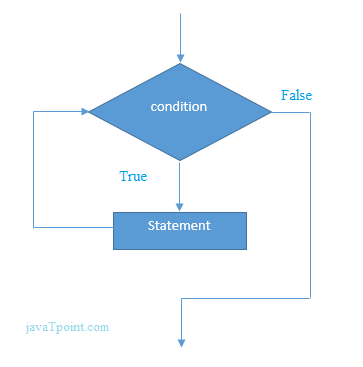
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop. While loop comes under the category of entry control loop as the loop executes after the condition checking.

**Syntax:**

while(condition)

{

}



**Example:**

class Test

{

public static void main(String args[])

{

int i=1;

while(i<10)

{

System.out.println(i+” “);

i+=1;

}

}

}

**Output:**

1

2

3

4

5

6

7

8

9

10

**Java infinitive while loop:**

public class WhileExample2

{

public static void main(String[] args)

{

while(true)

{

System.out.println("infinitive while loop");

}

}

}

**Output:**

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

ctrl+c