A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

There are two distinct types of multitasking

* process based multitasking
* thread based multitasking

**1. Process-Based Multitasking ( OS level)**

* **What it is:** Running multiple programs at the same time.
* **Example:** Imagine you’re using **Google Chrome** for browsing and **Spotify** for music at the same time. Each of these is a separate process, running independently.
* **How it works:** Each program has its own memory and doesn't share data directly with other programs.

**2. Thread-Based Multitasking (program level)**

* **What it is:** Doing multiple tasks inside the same program at the same time.
* **Example:** Imagine in **Google Chrome**, one thread loads the webpage, another thread plays a video, and another downloads a file—all happening simultaneously in the same app.
* **How it works:** Threads share the same memory and resources, which makes them faster.

**Thread class methods**

currentThread() Returns a reference to the thread

getName() Obtain a thread’s name

setName() set the name of a thread

getPriority() Obtain a thread’s priority

setPriority() to set a thread’s priority

isAlive() Determine if a thread is still running

join() Wait for a thread to terminate

run() Entry point for the thread.

sleep() suspend a thread for a period of time.

start() start a thread by calling its run method.

**Main Thread :**

When a java program starts up, one thread begins running immediately. This is usually called the main thread of your program. The main thread is important for two reasons.

* It is the thread from which other “child” threads will be spawned.
* Often, it must be the last thread to finish execution because it performs various shutdown actions
* Although the main thread is created automatically when your program is started, it can be controlled through a Thread object. Once you have a reference to the main thread, you can control it just like any other thread.
* Thread object displays, in order: the name of the thread, its priority and the name of its group. By default, the name of the main thread is main. Its priority is 5, which is the default value, and main is also the name of the group of threads to which this thread belongs.

Thread [main,5,main]

**Example :**

**public** **class** MainThreadDemo {

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

// **TODO** Auto-generated method stub

Thread t =Thread.*currentThread*();

System.***out***.println("current thread : "+t);

t.setName("myThread");

System.***out***.println("After name change : "+t);

**try** {

**for**(**int** n=5;n>0;n--) {

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

Thread.*sleep*(1000);

}

}**catch**(InterruptedException e) {

System.***out***.println("main thread interrupted");

}

}

}

**Two Ways to Create a child Thread**

In Java, there are two ways to create a child thread:

1. **Extending the Thread class.**
2. **Implementing the Runnable interface.**

### ****1. Extending the**** Thread ****Class****

#### **Concept:**

* You create a new class that extends the Thread class.
* Override the run() method to define the task that the thread will perform.
* Create an object of the class and call the start() method to begin the thread.

#### **Real-Time Example: Food Delivery App**

Let’s say we have a **food delivery app** where each delivery person is a thread, and their task is to deliver an order.

// Delivery person thread by extending Thread

**class DeliveryPerson extends Thread {**

**private String order;**

**public DeliveryPerson(String order,String name) {**

**super(name);**

**this.order = order;**

**}**

**public void run() {**

**System.out.println(getName() + " is delivering: " + order);**

**try {**

**Thread.sleep(2000); // Simulate time taken for delivery**

**} catch (InterruptedException e) {**

**System.out.println("Delivery interrupted for: " + order);**

**}**

**System.out.println(getName() + " completed delivery for: " + order);**

**}**

**}**

**public class FoodDeliveryApp {**

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

**DeliveryPerson person1 = new DeliveryPerson("Pizza","child1");**

**DeliveryPerson person2 = new DeliveryPerson("Burger","child2");**

**person1.start(); // Start thread for person1**

**person2.start(); // Start thread for person2**

**}**

**}**

Output :

child1 is delivering: Pizza

child2 is delivering: Burger

child1 completed delivery for: Pizza

child2 completed delivery for: Burger

### ****2. Implementing the**** Runnable ****Interface****

#### **Concept:**

* Create a class that implements the Runnable interface.
* Override the run() method to define the task.
* Create an object of your class, pass it to a Thread object, and call the start() method.

#### **Real-Time Example: Bank Transaction System**

Imagine a **bank system** where each thread processes a customer transaction (like depositing or withdrawing money).

// Bank transaction task by implementing Runnable

class BankTransaction implements Runnable {

private String transactionType;

public BankTransaction(String transactionType) {

this.transactionType = transactionType;

}

public void run() {

System.out.println(Thread.currentThread().getName() + " is processing: " + transactionType);

try {

Thread.sleep(1500); // Simulate time taken for processing

} catch (InterruptedException e) {

System.out.println("Transaction interrupted for: " + transactionType);

}

System.out.println(Thread.currentThread().getName() + " completed: " + transactionType);

}

}

public class BankSystem {

public static void main(String[] args) {

BankTransaction depositTask = new BankTransaction("Deposit");

BankTransaction withdrawTask = new BankTransaction("Withdraw");

Thread thread1 = **new** Thread(depositTask,"child1");

Thread thread2 = **new** Thread(withdrawTask,"child2");

thread1.start(); // Start thread for deposit

thread2.start(); // Start thread for withdrawal

}

}

Output :

Thread-0 is processing: Deposit

Thread-1 is processing: Withdraw

Thread-0 completed: Deposit

Thread-1 completed: Withdraw

**Multiple Thread :**

**Example :**

**class** NewThread **implements** Runnable {

String name;

Thread t;

NewThread(String threadname) {

name=threadname;

t=**new** Thread(**this**,name);

System.***out***.println("New thread : "+t);

t.start();

}

**public** **void** run() {

**try** {

**for**(**int** i=5;i>0;i--) {

System.***out***.println(name+":"+i);

Thread.*sleep*(1000);

}

}**catch**(InterruptedException e) {

System.***out***.println(name+ "Interrupted");

}

System.***out***.println(name +" exiting");

}

}

**public** **class** MultiThread {

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

// **TODO** Auto-generated method stub

NewThread ob1=**new** NewThread("One");

NewThread ob2=**new** NewThread("Two");

NewThread ob3=**new** NewThread("Three");

System.***out***.println("Thread one is alive: "+ob1.t.isAlive());

System.***out***.println("Thread two is alive: "+ob2.t.isAlive());

System.***out***.println("Thread three is alive: "+ob3.t.isAlive());

**try** {

ob1.t.join();

ob2.t.join();

ob3.t.join();

}**catch**(InterruptedException e) {

System.***out***.println("main thread Interrupted");

}

System.***out***.println("Thread one is alive: "+ob1.t.isAlive());

System.***out***.println("Thread two is alive: "+ob2.t.isAlive());

System.***out***.println("Thread three is alive: "+ob3.t.isAlive());

System.***out***.println("main thread exiting");

}

}