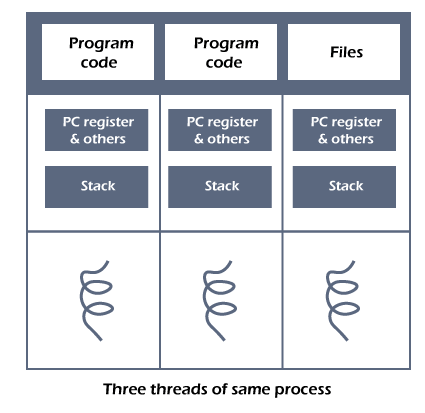
Multithreading in Java

* Multithreading is a Java feature that allows concurrent execution of two or more parts of a Program for maximum utilization of CPU
* Threads are light-weight processes within a process



Threads can be created by using two mechanisms :

1. Extending the Thread class
2. Implementing the Runnable Interface

## Thread creation by extending the Thread class

* Create a class that extends the java.lang.Thread class.
* This class overrides the run() method available in the Thread class
* A thread begins its life inside run() method.
* We create an object of our new class and call start() method to start the execution of a thread
* Start() invokes the run() method on the Thread object.

// Define a class that extends Thread

class MyThread extends Thread {

// Override the run() method to define the thread's behavior

public void run() {

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

try {

Thread.sleep(1000); // Simulate some task

} catch (InterruptedException e) {

System.out.println("Thread interrupted: " + e.getMessage());

}

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

}

}

public class ThreadCreationExample {

public static void main(String[] args) {

// Create multiple instances of MyThread

MyThread thread1 = new MyThread();

MyThread thread2 = new MyThread();

// Start the threads

thread1.start();

thread2.start();

// Main thread continues to run parallel to the created threads

System.out.println("Main thread finished.");

}

}

2. Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.Runnable interface and override run() method.

// Define a class that implements Runnable

class MyRunnable implements Runnable {

// Implement the run() method from the Runnable interface

public void run() {

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

try {

Thread.sleep(1000); // Simulate some task

} catch (InterruptedException e) {

System.out.println("Thread interrupted: " + e.getMessage());

}

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

}

}

public class ThreadCreationExample {

public static void main(String[] args) {

// Create instances of MyRunnable

MyRunnable myRunnable1 = new MyRunnable();

MyRunnable myRunnable2 = new MyRunnable();

// Create Thread objects, passing each MyRunnable instance to the constructor

Thread thread1 = new Thread(myRunnable1);

Thread thread2 = new Thread(myRunnable2);

// Start the threads

thread1.start();

thread2.start();

// Main thread continues to run parallel to the created threads

System.out.println("Main thread finished.");

}

}

**Thread Class vs Runnable Interface**

* If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
* We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.
* Using runnable will give you an object that can be shared amongst multiple threads.

class NumberPrinter implements Runnable {

private int start;

private int end;

public NumberPrinter(int start, int end) {

this.start = start;

this.end = end;

}

public void run() {

for (int i = start; i <= end; i++) {

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

}

}

}

public class PrintNumbersMultiThreaded {

public static void main(String[] args) {

NumberPrinter printer1 = new NumberPrinter(1, 5);

NumberPrinter printer2 = new NumberPrinter(6, 10);

Thread thread1 = new Thread(printer1, "Thread 1");

Thread thread2 = new Thread(printer2, "Thread 2");

thread1.start();

thread2.start();

}

}

—----------------------------------------------------------------------------------------------------------

class BankAccount {

private int balance = 1000;

public synchronized void withdraw(int amount) {

if (balance >= amount) {

balance -= amount;

System.out.println(Thread.currentThread().getName() + " withdrew " + amount + ". Remaining balance: " + balance);

} else {

System.out.println(Thread.currentThread().getName() + " cannot withdraw " + amount + ". Insufficient balance.");

}

}

}

class Transaction implements Runnable {

private BankAccount account;

public Transaction(BankAccount account) {

this.account = account;

}

public void run() {

for (int i = 0; i < 3; i++) {

account.withdraw(300);

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class BankTransactionExample {

public static void main(String[] args) {

BankAccount account = new BankAccount();

Transaction transaction1 = new Transaction(account);

Transaction transaction2 = new Transaction(account);

Thread thread1 = new Thread(transaction1, "Customer 1");

Thread thread2 = new Thread(transaction2, "Customer 2");

thread1.start();

thread2.start();

}

}