Multithreaded Programming in Java

A single threaded program

class ABC {
    ...
    public void main(...) {
        ...
        ..
    }
}

Web/Internet Applications: Serving Many Users Simultaneously

Multithreaded Server: For Serving Multiple Clients Concurrently

Agenda
- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
- Examples

A Multithreaded Program
Modern Applications need Threads (ex1): Editing and Printing documents in background.

Multithreading - Multiprocessors

Multithreading on Uni-processor

Levels of Parallelism

Single and Multithreaded Processes

No of execution process must the number of CPUs

Multithreaded/Parallel File Copy

Concurrency Vs Parallelism

Number of Simultaneous execution units x number of CPUs
What are Threads?

- A piece of code that run in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control – Multithreading or Multithreaded Programming.

Java Threads

- Java has built in thread support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
  - currentThread
  - start
  - setPriority
  - yield
  - run
  - getPriority
  - sleep
  - stop
  - suspend
  - resume
- Java Garbage Collector is a low-priority thread

Threading Mechanisms...

- Create a class that extends the Thread class
- Create a class that implements the Runnable interface

1st method: Extending Thread class

- Threads are implemented as objects that contains a method called run()
- class MyThread extends Thread
  
  public void run()
  {
    // thread body of execution
  }

- Create a thread:
  MyThread thr1 = new MyThread();
- Start Execution of threads:
  thr1.start();

An example

```java
class MyThread extends Thread {
    public void run() {
        System.out.println("this thread is running ... ");
    }
}
class ThreadEx1 {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        // due to extending the Thread class (above)
        // I can call start(), and this will call
        // run(). start() is a method in class Thread.
        t.start();
    }
}
```

2nd method: Threads by implementing Runnable interface

```java
class MyThread implements Runnable {
    public void run() {
        // thread body of execution
    }
}
class ThreadEx1 {
    public static void main(String[] args) {
        MyThread myObject = new MyThread();
        // Creating Thread Object:
        Thread thr1 = new Thread(myObject);
        // Start Execution:
        thr1.start();
    }
}
An example

class MyThread implements Runnable {
    public void run() {
        System.out.println("this thread is running...");
    }
} // end class MyThread

class ThreadEx2 {
    public static void main(String[] args) {
        Thread t = new Thread(new MyThread());
        // due to implementing the Runnable interface
        // I can call start(), and this will call run();
        t.start();
    } // end main()
} // end class ThreadEx2

Life Cycle of Thread

A Program with Three Java Threads

Write a program that creates 3 threads

class A extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("From Thread A: i= " + i);
        }
        System.out.println("Exit from A");
    }
}

class B extends Thread {
    public void run() {
        for (int j = 1; j <= 5; j++) {
            System.out.println("From Thread B: j= " + j);
        }
        System.out.println("Exit from B");
    }
}

class C extends Thread {
    public void run() {
        for (int k = 1; k <= 5; k++) {
            System.out.println("From Thread C: k= " + k);
        }
        System.out.println("Exit from C");
    }
}

class ThreadTest {
    public static void main(String args) {
        new A().start();
        new B().start();
        new C().start();
    }
}

Run 1

[log@mundro] threads [1:76] java ThreadTest
From ThreadA: 1
From ThreadA: 2
From ThreadA: 3
From ThreadA: 4
From ThreadA: 5
Exit from A
From ThreadB: 1
From ThreadB: 2
From ThreadC: 3
From ThreadC: 4
From ThreadC: 5
Exit from C
From ThreadB: 1
From ThreadB: 2
From ThreadB: 3
From ThreadB: 4
From ThreadB: 5
Exit from B
Run2

- int add (int a, int b, int & result)
  // function stuff
- int sub (int a, int b, int & result)
  // function stuff

From Thread A:
- i = 1
- i = 2
- i = 3
- i = 4
- i = 5

From Thread C:
- k = 1
- k = 2
- k = 3
- k = 4
- k = 5

Exit from C

Exit from B

Exit from A

Process Parallelism

- int add (int a, int b, int & result)
- int sub (int a, int b, int & result)

Data Parallelism

- sort (int *array, int count)
  // function stuff

Next Class

- Thread Synchronisation
- Thread Priorities

Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - Printer (two person jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account

Online Bank: Serving Many Customers and Operations
Shared Resources

- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to data.
- In Java: “Synchronized” method:
  - Synchronised void update()
  - {
    "...
  }

Program with 3 threads and shared object

```java
class MyThread implements Runnable {
    Account account;
    public MyThread (Account s) { account = s; }
    public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable {
    Account account;
    public YourThread (Account s) { account = s; }
    public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable {
    Account account;
    public HerThread (Account s) { account = s; }
    public void run() { account.enquire(); }
} // end class HerThread
```

Thread Priority

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (ORM_PRIORITY) and they are served using FCFS policy.
- Java allows users to change priority:
  - ThreadName.setPriority(intNumber)
    - MIN_PRIORITY = 1
    - NORM_PRIORITY=5
    - MAX_PRIORITY=10

Monitor (shared object) example

```java
class Account { // the 'monitor'
    // DATA Members
    int balance;
    // if 'synchronized' is removed, the outcome is unpredictable
    public synchronized void deposit() {
        // METHOD BODY: balance += deposit_amount;
    }
    public synchronized void withdraw() {
        // METHOD BODY: balance -= deposit_amount;
    }
    public synchronized void enquire() {
        // METHOD BODY: display balance.
    }
}
```

Thread Priority Example

```java
class ExtendedThread {
    public void run() {
        System.out.println("Thread A started");
        for (int i = 1 ; i <= 4 ; i++)
            System.out.println(" From Thread A: i = " + i);
        System.out.println(" Exit from A");
    }
}
```
Thread Priority Example

class Thread

```java
public void run()
{
    System.out.println("Thread C started");
    for (int k = 1; k <= 4; k++)
    {
        System.out.println("From Thread C: k = " + k);
    }
    System.out.println("Exit from C");
}
```

class ThreadPriority

```java
public static void main(String args[])
{
    A thread A = new A();
    B thread B = new B();
    C thread C = new C();
    thread C.setPriority(Thread.MAX_PRIORITY);
    thread B.setPriority(thread A.getPriority() + 1);
    thread A.setPriority(Thread.MIN_PRIORITY);
    System.out.println("Started Thread A");
    thread A.start();
    System.out.println("Started Thread B");
    thread B.start();
    System.out.println("Started Thread C");
    thread C.start();
    System.out.println("End of main thread");
}
```