Write a multi-threaded Java program to print all numbers below 100,000 that are both prime and fibonacci number (some examples are 2, 3, 5, 13, etc.). Design a thread that generates prime numbers below 100,000 and writes them into a pipe. Design another thread that generates fibonacci numbers and writes them to another pipe. The main thread should read both the pipes to identify numbers common to both by using runnable interface as well as thread class.

To generate prime numbers and Fibonacci series using java.(Multithreaded)

import java.io.\*;

import java.util.\*;

class MyThread1 extends Thread {

private PipedReader pr;

private PipedWriter pw;

MyThread1(PipedReader pr, PipedWriter pw) {

this.pr = pr;

this.pw = pw;

}

public void run() {

try {

int i;

for (i=1;i<10;i++){

int j;

for (j=2; j<i; j++){

int n = i%j;

if (n==0){

break;}}

if(i == j){

pw.write(i+"\n");}

}pw.close();}

catch (IOException e) {

}}}

class MyThread2 extends Thread {

private PipedReader pr;

private PipedWriter pw;

MyThread2(PipedReader pr, PipedWriter pw) {

this.pr = pr;

this.pw = pw;}

public void run() {

try {

int f1, f2 = 1, f3 = 1;

for (int i = 1; i <10; i++) {

pw.write(f3+"\n");

f1 = f2;

f2 = f3;

f3 = f1 + f2;}

pw.close();

} catch (IOException e) {

}}}

class MultithreadedProgram {

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

ArrayList list1=new ArrayList();

ArrayList list2=new ArrayList();

PipedWriter pw1 = new PipedWriter();

PipedReader pr1 = new PipedReader(pw1);

MyThread1 mt1 = new MyThread1(pr1, pw1);

System.out.println("Prime Numbers: ");

mt1.start();

int item1;

while ((item1 = pr1.read()) != -1){

char ch1=((char) item1);

System.out.print(Character.toString(ch1));

list1.add(Character.toString(ch1));

}

pr1.close();

PipedWriter pw2 = new PipedWriter();

PipedReader pr2 = new PipedReader(pw2);

MyThread2 mt2 = new MyThread2(pr2, pw2);

System.out.println("Fibonacci Numbers: ");

mt2.start();

int item2;

while ((item2 = pr2.read()) != -1){

char ch2=((char) item2);

System.out.print(Character.toString(ch2));

list2.add(Character.toString(ch2));

}

pr2.close();

System.out.println("Elements common to both lists are:");

list1.retainAll(list2);

for(int i=0;i<list1.size();i++){

System.out.print(list1.get(i));

}}}

**OUTPUT:**

E:\SQ>java MultithreadedProgram

Prime Numbers:

2

3

5

7

Fibonacci Numbers:

1

2

3

5

8

13

21

34

55

Elements common to both lists are:

2

3

5

**Multithreading For Prime and Fibonacci numbers using pipes in Java (Another solution)**

import java.util.\*;

import java.io.\*;

class Fibonacci extends Thread

{

private PipedWriter out = new PipedWriter();

public PipedWriter getPipedWriter()

{

return out;

}

public void run()

{

Thread t = Thread.currentThread();

t.setName("Fibonacci");

System.out.println(t.getName() + " thread started");

int fibo1=0,fibo2=1,fibo=0;

while(true)

{

try

{

fibo = fibo1 + fibo2;

if(fibo>100000)

{

out.close();

break;

}

out.write(fibo);

sleep(100);

}

catch(Exception e)

{

System.out.println("Fibonacci:"+e);

}

fibo1=fibo2;

fibo2=fibo;

}

System.out.println(t.getName() + " thread exiting");

}

}

class Prime extends Thread

{

private PipedWriter out1 = new PipedWriter();

public PipedWriter getPipedWriter()

{

return out1;

}

public void run()

{

Thread t= Thread.currentThread();

t.setName("Prime");

System.out.println(t.getName() + " thread Started...");

int prime=1;

while(true)

{

try

{

if(prime>100000)

{

out1.close();

break;

}

if(isPrime(prime))

out1.write(prime);

prime++;

sleep(0);

}

catch(Exception e)

{

System.out.println(t.getName() + " thread exiting.");

System.exit(0);

}

}

}

public boolean isPrime(int n)

{

int m=(int)Math.round(Math.sqrt(n));

if(n==1 || n==2)

return true;

for(int i=2;i<=m;i++)

if(n%i==0)

return false;

return true;

}

}

public class PipedIo

{

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

{

Thread t=Thread.currentThread();

t.setName("Main");

System.out.println(t.getName() + " thread Started...");

Fibonacci fibonacci = new Fibonacci();

Prime prime = new Prime();

PipedReader fpr = new PipedReader(fibonacci.getPipedWriter());

PipedReader ppr = new PipedReader(prime.getPipedWriter());

fibonacci.start();

prime.start();

int fib=fpr.read(), prm=ppr.read();

System.out.println("The numbers common to PRIME and FIBONACCI:");

while((fib!=-1) && (prm!=-1))

{

while(prm<=fib)

{

if(fib==prm)

System.out.println(prm);

prm=ppr.read();

}

fib=fpr.read();

}

System.out.println(t.getName() + " thread exiting");

}

}

**Producer Consumer Multithreading Example**

import java.util.Vector;

class Producer extends Thread {

static final int MAXQUEUE = 5;

private Vector messages = new Vector();

public void run() {

try {

while ( true ) {

putMessage();

sleep( 1000 );

}

}

catch( InterruptedException e ) { }

}

private synchronized void putMessage()

throws InterruptedException {

while ( messages.size() == MAXQUEUE )

wait();

messages.addElement( new java.util.Date().toString() );

notify();

}

// Called by Consumer

public synchronized String getMessage()

throws InterruptedException {

notify();

while ( messages.size() == 0 )

wait();

String message = (String)messages.firstElement();

messages.removeElement( message );

return message;

}

}

class Consumer extends Thread {

Producer producer;

String name;

Consumer(String name, Producer producer) {

this.producer = producer;

this.name = name;

}

public void run() {

try {

while ( true ) {

String message = producer.getMessage();

System.out.println(name + " got message: " + message);

sleep( 2000 );

}

}

catch( InterruptedException e ) { }

}

public static void main(String args[]) {

Producer producer = new Producer();

producer.start();

// Start two this time

new Consumer( "One", producer ).start();

new Consumer( "Two", producer ).start();

}

}

**OUTPUT:**

E:\SQ>java Consumer

Two got message: Wed Sep 01 11:57:54 GMT+05:30 2010

One got message: Wed Sep 01 11:57:55 GMT+05:30 2010

Two got message: Wed Sep 01 11:57:56 GMT+05:30 2010

One got message: Wed Sep 01 11:57:57 GMT+05:30 2010

Two got message: Wed Sep 01 11:57:58 GMT+05:30 2010

One got message: Wed Sep 01 11:57:59 GMT+05:30 2010

Two got message: Wed Sep 01 11:58:00 GMT+05:30 2010

One got message: Wed Sep 01 11:58:01 GMT+05:30 2010

Two got message: Wed Sep 01 11:58:02 GMT+05:30 2010

One got message: Wed Sep 01 11:58:03 GMT+05:30 2010

One got message: Wed Sep 01 11:58:04 GMT+05:30 2010

Two got message: Wed Sep 01 11:58:18 GMT+05:30 2010

One got message: Wed Sep 01 11:58:19 GMT+05:30 2010

Two got message: Wed Sep 01 11:58:20 GMT+05:30 2010

One got message: Wed Sep 01 11:58:21 GMT+05:30 2010

Two got message: Wed Sep 01 11:58:22 GMT+05:30 2010

#### Example:using isAlive() and join()

**// Using join() to wait for threads to finish.**

**class NewThread implements Runnable**

**{**

**String name; // name of thread**

**Thread t;**

**NewThread(String threadname) {**

**name = threadname;**

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

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

**t.start(); // Start the thread**

**}**

**// This is the entry point for thread.**

**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.");**

**}**

**}**

**class DemoJoin**

**{**

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

**{**

**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());**

**// wait for threads to finish**

**try**

**{**

**System.out.println("Waiting for threads to finish.");**

**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.");**

**}**

**}**

Sample output from this program is shown here. (Your output may vary based on processor speed and task load.)  
  
**New thread: Thread[One,5,main]**  
**New thread: Thread[Two,5,main]**  
**New thread: Thread[Three,5,main]**  
**Thread One is alive: true**  
**Thread Two is alive: true**  
**Thread Three is alive: true**  
**Waiting for threads to finish.**  
**One: 5**  
**Two: 5**  
**Three: 5**  
**One: 4**  
**Two: 4**  
**Three: 4**  
**One: 3**  
**Two: 3**  
**Three: 3**  
**One: 2**  
**Two: 2**  
  
**Three: 2**

To develop a multithreaded GUI application in java

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.util.\*;

import java.awt.geom.\*;

class Ball{

public void move(Rectangle2D bounds)

{

x += dx;

y += dy;

if (x < bounds.getMinX())

{

x = bounds.getMinX();

dx = -dx; }

if (x + XSIZE >= bounds.getMaxX())

{

x = bounds.getMaxX() - XSIZE;

dx = -dx; }

if (y < bounds.getMinY())

{

y = bounds.getMinY();

dy = -dy; }

if (y + YSIZE >= bounds.getMaxY())

{

y = bounds.getMaxY() - YSIZE;

dy = -dy; } }

public Ellipse2D getShape()

{

return new Ellipse2D.Double(x, y, XSIZE, YSIZE);

}

private static final int XSIZE = 15;

private static final int YSIZE = 15;

private double x = 0;

private double y = 0;

private double dx = 1;

private double dy = 1;

}

class BallComponent extends JComponent

{

public void add(Ball b)

{

balls.add(b);}

public void paintComponent(Graphics g)

{

Graphics2D g2 = (Graphics2D) g;

for (Ball b : balls)

{

g2.fill(b.getShape()); } }

private ArrayList<Ball> balls = new ArrayList<Ball>();

}

public class BounceThread

{

public static void main(String[] args)

{

EventQueue.invokeLater(new Runnable()

{

public void run() {

JFrame frame = new BounceFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);}

}); }}

class BallRunnable implements Runnable

{

public BallRunnable(Ball aBall, Component aComponent)

{

ball = aBall;

component = aComponent;

}

public void run(){

try {

for (int i = 1; i <= STEPS; i++)

{

ball.move(component.getBounds());

component.repaint();

Thread.sleep(DELAY); } }

catch (InterruptedException e)

{

}}

private Ball ball;

private Component component;

public static final int STEPS = 1000;

public static final int DELAY = 5;

}

class BounceFrame extends JFrame

{

public BounceFrame() {

setSize(DEFAULT\_WIDTH, DEFAULT\_HEIGHT);

setTitle("BounceThread");

comp = new BallComponent();

add(comp, BorderLayout.CENTER);

JPanel buttonPanel = new JPanel();

addButton(buttonPanel, "Start", new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

addBall();}

});

addButton(buttonPanel, "Close", new ActionListener()

{

public void actionPerformed(ActionEvent event)

{

System.exit(0);}

});

add(buttonPanel, BorderLayout.SOUTH);

}

public void addButton(Container c, String title, ActionListener listener)

{

JButton button = new JButton(title);

c.add(button);

button.addActionListener(listener);

}

public void addBall()

{

Ball b = new Ball();

comp.add(b);

Runnable r = new BallRunnable(b, comp);

Thread t = new Thread(r);

t.start();

}

private BallComponent comp;

public static final int DEFAULT\_WIDTH = 450;

public static final int DEFAULT\_HEIGHT = 350;

public static final int STEPS = 1000;

public static final int DELAY = 3;

}

Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of

the number.