Index

[Exercise 1: Use the Thread class 3](#_Toc65834754)

[Task 1 - Create RunThread class extending Thread class 3](#_Toc65834755)

[Task 2 - Main program 3](#_Toc65834756)

[Task 3 - Execute your program 3](#_Toc65834757)

[Exercise 2: Implement the Runable Interface 4](#_Toc65834758)

[Task 1 - Create RunThread class Implement Runable Interface 4](#_Toc65834759)

[Task 2 - Main program 4](#_Toc65834760)

[Task 3 - Execute your program 4](#_Toc65834761)

[Exercise 3: Using join() to wait for threads to finish 5](#_Toc65834762)

[Task 1 - Create MyThread class Implement Runable Interface 5](#_Toc65834763)

[Task 2 - Main program 5](#_Toc65834764)

[Task 3 - Execute your program 6](#_Toc65834765)

[Exercise 4: Thread priority 6](#_Toc65834766)

[Task 1 - Create clicker class Implement Runable Interface 6](#_Toc65834767)

[Task 2 - Main program 7](#_Toc65834768)

[Task 3 - Execute your program 8](#_Toc65834769)

[Exercise 5: Synchronization 8](#_Toc65834770)

[Task 1 - Create CountPrimesThread class extending Thread class 8](#_Toc65834771)

[Task 2 - Create a running Program 9](#_Toc65834772)

[Task 3 - Execute your program 10](#_Toc65834773)

[Task 4 - Change your program to make speed up your program: 11](#_Toc65834774)

[Exercise 6: Dining Philosopher 12](#_Toc65834775)

[Task 1 - Create ChopStick class 12](#_Toc65834776)

[Task 2 - Create Philosopher class 13](#_Toc65834777)

[Task 3 - Dining class to test y our philosophers 13](#_Toc65834778)

[Task 4 - Execute your program 14](#_Toc65834779)

[Task 5 - Modify your code to resolve this problem, your output will be: 15](#_Toc65834780)

1. Use the Thread class

## Create RunThread class extending Thread class

### Write code for RunThread Class

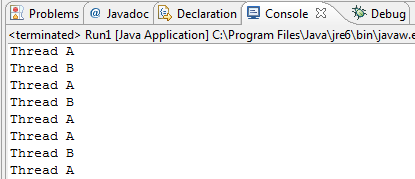
* 1. **private** String toSay;
  2. **private** **int** Sleep;
  3. **public** RunThread(String st,**int** sl){
  4. toSay = st;
  5. Sleep=sl;
  6. }
  7. **public** **void** run(){
  8. **try**{
  9. **for**(;;){
  10. System.*out*.println(toSay);
  11. *sleep*(Sleep);
  12. }
  13. } **catch**(InterruptedException e){
  14. }
  15. }

## Main program

### Write code for main() method

* 1. **public** **static** **void** main(String [] args){
  2. RunThread thr1 = **new** RunThread("Thread A",1200);
  3. RunThread thr2 = **new** RunThread("Thread B",2000);
  4. thr1.start();
  5. thr2.start();
  6. }

## Execute your program



1. Implement the Runable Interface

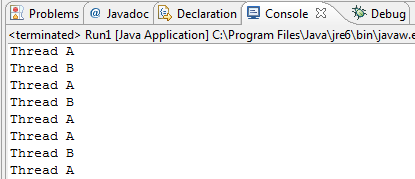
## Create RunThread class Implement Runable Interface

* 1. **private** String toSay;
  2. **int** sleep;
  3. **public** RunThread1(String st,**int** sl){
  4. toSay = st;
  5. sleep=sl;
  6. }
  7. **public** **void** run() {
  8. **try** {
  9. **for**(;;) {
  10. System.*out*.println(toSay);
  11. Thread.*sleep*(sleep);
  12. }
  13. }**catch**(InterruptedException e){
  15. }
  16. }

## Main program

* 1. **public** **static** **void** main(String [] args) {
  2. RunThread1 out1 = **new** RunThread1("Thread A",2000);
  3. RunThread1 out2 = **new** RunThread1("Thread B",1200);
  4. Thread thr1 = **new** Thread(out1);
  5. Thread thr2 = **new** Thread(out2);
  6. thr1.start();
  7. thr2.start();
  8. }

## Execute your program



1. Using join() to wait for threads to finish

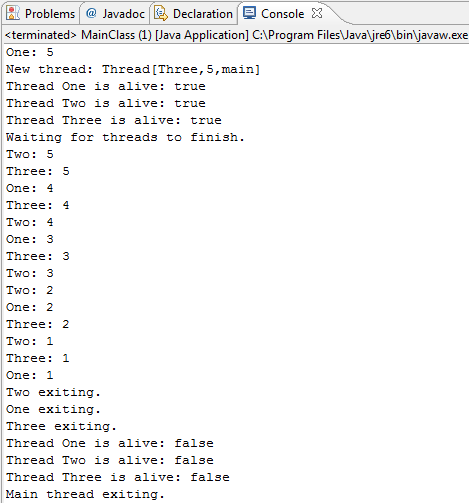
## Create MyThread class Implement Runable Interface

* 1. **class** MyThread **implements** Runnable {
  2. String name; // name of thread
  3. Thread t;
  4. MyThread(String threadname) {
  5. name = threadname;
  6. t = **new** Thread(**this**, name);
  7. System.*out*.println("New thread: " + t);
  8. t.start();
  9. }
  10. **public** **void** run() {
  11. **try** {
  12. **for** (**int** i = 5; i > 0; i--) {
  13. System.*out*.println(name + ": " + i);
  14. Thread.*sleep*(1000);
  15. }
  16. } **catch** (InterruptedException e) {
  17. System.*out*.println(name + " interrupted.");
  18. }
  19. System.*out*.println(name + " exiting.");
  20. }
  21. }

## Main program

* 1. MyThread ob1 = **new** MyThread("One");
  2. MyThread ob2 = **new** MyThread("Two");
  3. MyThread ob3 = **new** MyThread("Three");
  4. System.*out*.println("Thread One is alive: " + ob1.t.isAlive());
  5. System.*out*.println("Thread Two is alive: " + ob2.t.isAlive());
  6. System.*out*.println("Thread Three is alive: " + ob3.t.isAlive());
  7. **try** {
  8. System.*out*.println("Waiting for threads to finish.");
  9. ob1.t.join();
  10. ob2.t.join();
  11. ob3.t.join();
  12. } **catch** (InterruptedException e) {
  13. System.*out*.println("Main thread Interrupted");
  14. }
  15. System.*out*.println("Thread One is alive: " + ob1.t.isAlive());
  16. System.*out*.println("Thread Two is alive: " + ob2.t.isAlive());
  17. System.*out*.println("Thread Three is alive: " + ob3.t.isAlive());
  18. System.*out*.println("Main thread exiting.");
  19. }

## Execute your program



1. Thread priority

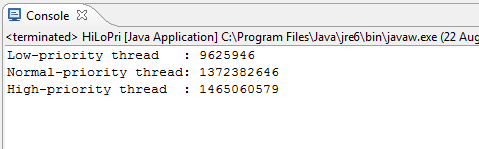
## Create clicker class Implement Runable Interface

* 1. **long** click = 0;
  2. Thread t;
  3. **private** **volatile** **boolean** running = **true**;
  4. **public** clicker(**int** p) {
  5. t = **new** Thread(**this**);
  6. t.setPriority(p);
  7. }
  8. **public** **void** run() {
  9. **while** (running) {
  10. click++;
  11. }
  12. }
  13. **public** **void** stop() {
  14. running = **false**;
  15. }
  16. **public** **void** start() {
  17. t.start();
  18. }

## Main program

* 1. **public** **static** **void** main(String args[]) {
  2. Thread.*currentThread*().setPriority(Thread.*MAX\_PRIORITY*);
  3. clicker hi = **new** clicker(Thread.*NORM\_PRIORITY* + 2);
  4. clicker no = **new** clicker(Thread.*NORM\_PRIORITY*);
  5. clicker lo = **new** clicker(Thread.*NORM\_PRIORITY* - 2);
  6. lo.start();
  7. no.start();
  8. hi.start();
  9. **try** {
  10. Thread.*sleep*(5000);
  11. } **catch** (InterruptedException e) {
  12. System.*out*.println("Main thread interrupted.");
  13. }
  15. lo.stop();
  16. no.stop();
  17. hi.stop();
  18. // Wait for child threads to terminate.
  19. **try** {
  20. hi.t.join();
  21. no.t.join();
  22. lo.t.join();
  23. } **catch** (InterruptedException e) {
  24. System.*out*.println("InterruptedException caught");
  25. }
  26. System.*out*.println("Low-priority thread : " + lo.click);
  27. System.*out*.println("Normal-priority thread: " + no.click);
  28. System.*out*.println("High-priority thread : " + hi.click);
  29. }

## Execute your program



### Set other priority for each thread and see the output result.

1. Synchronization

The program in this exercise will demonstrate the usage of synchronization to count the number of prime integers between 0 and 1000000. The work can be divided among one to five threads, where the number of threads required to search the prime numbers will be decided by the user.

## Create CountPrimesThread class extending Thread class

### Class variables

//Variable to store the number of primes in a specified range of

**static** **int** *count* = 0;

// Variable to store the range of minimum and maximum integer value

**int** min, max;

### Create constructors

* 1. **public** CountPrimesThread(**int** min, **int** max) {
  2. **this**.min = min;
  3. **this**.max = max;
  4. }

### isPrime() method is invoked to test whether x is a prime number or not.

* 1. **private** **static** **boolean** isPrime(**int** x) {
  2. **int** top = (**int**)Math.*sqrt*(x);
  3. **for**(**int** i = 2;i <= top;i++)
  4. **if**(x % i == 0)
  5. **return** **false**;
  6. **return** **true**;
  7. }

### countPrimes() method is invoked to count the primes between min and max, inclusive.

**private** **static** **void** countPrimes(**int** min, **int** max) {

**for**(**int** i = min;i <= max;i++)

**if**(*isPrime*(i))

*count*++;

}

### Run() method is invoked to output a message about the number of primes that it has found, and it adds its count.

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

*countPrimes*(min, max);

System.*out*.println("There are " + *count* + " primes between " +

"" + min + " and " + max);

}

## Create a running Program

### Write a CountPrimesWithThreads() to create thread to count prime numbers

**public** **static** **void** countPrimesWithThreads(**int** numberOfThreads) {

**int** start =0;

**int** to =1000000;

**int** increment = to/numberOfThreads;

System.*out*.println("\nCounting primes between " + (start+1) + " and "

+ to + " using " + numberOfThreads + " threads. " +

"Please wait...\n");

**long** startTime = System.*currentTimeMillis*();

CountPrimesThread[] worker = **new** CountPrimesThread[numberOfThreads];

**for**(**int** i = 0;i < numberOfThreads;i++)

worker[i] = **new** CountPrimesThread(start+i\*increment+1,

start+(i+1)\*increment);

**for**(**int** i = 0;i < numberOfThreads;i++)

worker[i].start();

**for**(**int** i = 0;i < numberOfThreads;i++) {

**while**(worker[i].isAlive()) {

**try** {

worker[i].join();

} **catch** (InterruptedException e) {

}

}

}

**long** elapsedTime = System.*currentTimeMillis*() - startTime;

System.*out*.println("\nTotal elapsed time: " + (elapsedTime/1000.0) +

"seconds.\n");

}

### Write code for main() method

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

**int** numberOfThreads = 0;

**do**

{

System.*out*.println("How many threads do you want to use (from 1 " + "to 5)?");

Scanner input = **new** Scanner(System.*in*);

numberOfThreads = input.nextInt();

**if** (numberOfThreads < 1 || numberOfThreads > 5)

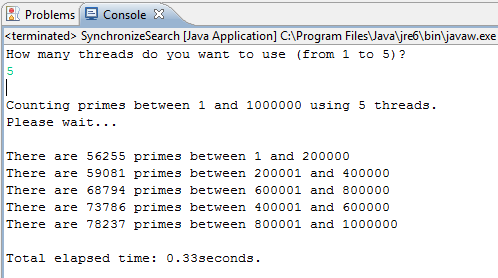
System.*out*.println("Please enter 1, 2, 3, 4, or 5!");

} **while** (numberOfThreads < 1 || numberOfThreads > 5);

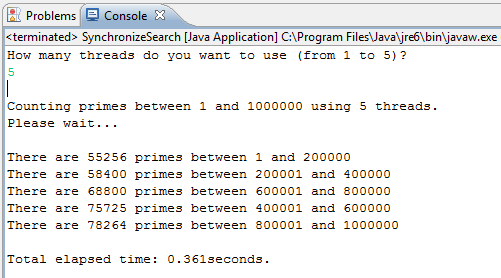
*countPrimesWithThreads*(numberOfThreads);

}

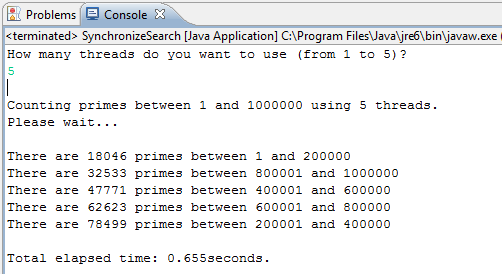
## Execute your program



### And Try Again



### Add **synchronized** keyword to countPrimes method and run again

* 1. 

## Change your program to make speed up your program:

### Add **addcount** method to CountPrimesThread class

**synchronized** **private** **static** **void** addcount(**int** x)

{

*count*=*count*+x;

}

### Change CountPrimes() method

**private** **static** **void** countPrimes(**int** min, **int** max) {

**int** x=0;

**for**(**int** i = min;i <= max;i++)

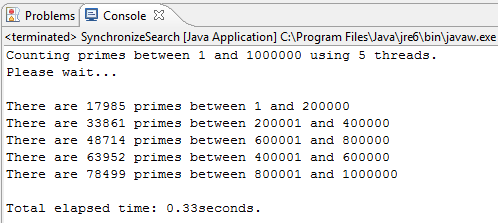
**if**(*isPrime*(i))

x++;

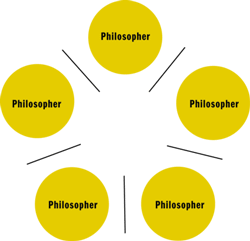
*addcount*(x);

}

### Run your program again:

* 1. 

1. Dining Philosopher



## Create ChopStick class

**public** **class** ChopStick {

**public** ChopStick() {

available = **true**;

}

**public** **synchronized** **void** takeup() {

**while**(!available) {

System.*out*.println("Philosopher is waiting for the other " +

"chopstick");

}

available = **false**;

}

**public** **synchronized** **void** putdown() {

available = **true**;

}

**boolean** available;

}

## Create Philosopher class

**public** **class** Philosopher **extends** Thread {

**public** Philosopher(**int** num, ChopStick chop1,ChopStick chop2) {

philo\_num = num;

left = chop1;

right = chop2;

}

**public** **void** eat() {

left.takeup();

right.takeup();

System.*out*.println("Philosopher "+(philo\_num+1)+" is eating");

}

**public** **void** think() {

left.putdown();

right.putdown();

System.*out*.println("Philosopher "+(philo\_num+1)+" is thinking");

}

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

**while**(**true**) {

eat();

**try** {

*sleep*(2000);

} **catch** (InterruptedException ex) {

ex.printStackTrace();

}

think();

**try** {

*sleep*(2000);

} **catch** (InterruptedException ex) {

ex.printStackTrace();

}

}

}

ChopStick left, right;

**int** philo\_num;

}

## Dining class to test y our philosophers

**public** **class** Dining {

**public** Dining() {

}

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

ChopStick[] chopsticks = **new** ChopStick[5];

Philosopher[] philos = **new** Philosopher[5];

**for**(**int** count = 0;count <= 4;count++) {

chopsticks[count] = **new** ChopStick();

}

**for**(**int** count = 0;count <= 4;count++) {

philos[count] = **new** Philosopher(count, chopsticks[count],

chopsticks[(count+1)%5]);

}

**for**(**int** count = 0;count <= 4;count++) {

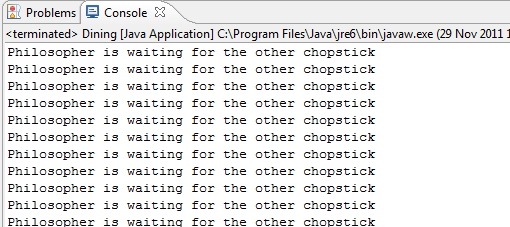
philos[count].start( );

}

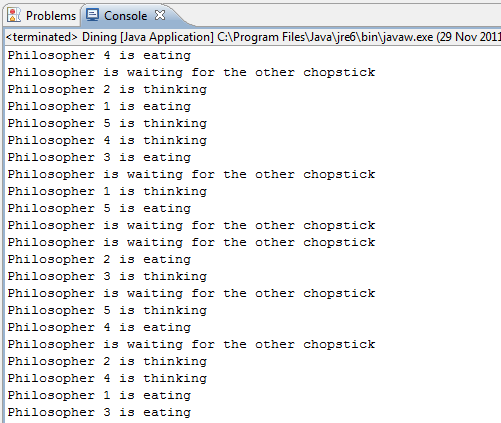
}

}

## Execute your program



## Modify your code to resolve this problem, your output will be:



Just for fun



And other solution



But

