스레드

2016-11-21

이승진

**학습 목표**

스레드 생성

공유 변수 잠금

공유 객체 잠금

**목차**

[1. 스레드 (thread) 2](#_Toc467535324)

[1) 스레드란? 2](#_Toc467535325)

[2) 멀티 스레드 (multi thread) 2](#_Toc467535326)

[3) 스레드의 수 2](#_Toc467535327)

[4) 메인 스레드(main thread) 3](#_Toc467535328)

[5) 실행 흐름 생성 3](#_Toc467535329)

[6) 구현 방법1 3](#_Toc467535330)

[7) 구현 방법2 3](#_Toc467535331)

[2. 스레드 생성 예제 4](#_Toc467535332)

[1) 스레드 생성 방법1 4](#_Toc467535333)

[2) 스레드 생성 방법2 5](#_Toc467535334)

[3. 스레드 종료 시점 6](#_Toc467535335)

[1) Example03.java 6](#_Toc467535336)

[2) Example04.java 8](#_Toc467535337)

[3) Example05.java 10](#_Toc467535338)

[4. 공유 변수 12](#_Toc467535339)

[1) Example06.java 12](#_Toc467535340)

[2) Example07.java 13](#_Toc467535341)

[5. 공유 객체 14](#_Toc467535342)

[1) Example08.java 14](#_Toc467535343)

[2) Example09.java 15](#_Toc467535344)

[6. 실습 16](#_Toc467535345)

# 스레드 (thread)

## 스레드란?

Java 컴파일러는 Java 소스 코드를 바이트코드(bytecode)로 컴파일 한다.



Java 가상 기계(virtual machine)가

Java 바이트코드 파일을 메모리에 읽어들여서 실행한다.

메모리에 올라와 있는 바이트 코드를 도로의 보도 블럭에 비유하자.

보도 블럭 위를 걸어가는 사람을 CPU에 비유하자.

사람이 보도 블럭 위를 걸어가는 것을, CPU가 바이트 코드를 실행하는 것에 비유할 수 있다.

CPU가 바이트 코드를 하나씩 순서대로 실행하고 있는 것을, 그 실행 흐름을 스레드(thread)라고 부른다.

스레드란 사람이 보도 블럭 위를 걸어가고 있는 것에 비유할 수 있다.

## 멀티 스레드 (multi thread)

요즘 CPU는 멀티 코어 형태이다.

CPU의 코어는 하나의 독립적인 CPU 이다.

즉 요즘 CPU는 작은 CPU 여러 개가 결합된 형태이다.

CPU가 멀티 코어이기 때문에, 각각의 코어가 독립적으로 바이트 코드를 실행할 수 있다.

즉 여러 개의 스레드가 동시에 실행될 수 있다.

이것을 멀티 스레드(multi thread)라고 부른다.

멀티 스레드는, 보도 블럭 위에서 사람 여러 명이 동시에 걸어가고 있는 것에 비유할 수 있다.

주의할 점은 보도 블럭이 스레드인 것이 아니고, 보도 블럭 위를 걸어가는 사람이 스레드이라는 점이다.

즉 바이트 코드가 스레드인 것이 아니고, 바이트 코드를 실행하는 CPU 코어가 스레드이다.

실행 흐름을 선으로 표현 한다면, 싱글 스레드는 선이 한 가닥인 것이고, 멀티 스레드는 선이 여러 가닥인 것이다.

## 스레드의 수

메모리를 책에 비유하자.

책에 적힌 문장들을 바이트 코드에 비유할 수 있다.

CPU가 바이트 코드를 하나씩 실행하는 것을, 책에 적힌 문장들을 사람이 하나씩 읽는 것에 비유하자.

스레드란 책에 적힌 문장들을 사람이 하나씩 읽어 나가는 것에 비유할 수 있다.

책을 읽는 사람의 수는 CPU 코어 수에 비유할 수 있다.

4 코어 CPU는 4명의 사람이 동시에 책을 읽을 수 있다는 말이다.

그런데 사람들이 책을 한 권씩만 읽고 있는 것이 아니고,

책을 여러 권 펼쳐 넣고, 잠시 이 책을 읽다가 중단하고 또 저 책을 읽고,

책 여러 권을 번갈아 읽는다고 생각해 보자.

책에 적힌 문장들을 하나씩 순서대로 읽어 나가는 것이 스레드이다.

4명의 사람이 각각 책 3권씩 펼쳐 넣고, 책을 번갈아 읽는다면,

스레드의 수는 4명 x 3권 = 12 개 이다.

물론 어느 한 순간 실행 중인 스레드는 4개 뿐일 것이다. (사람이 4명이므로)

## 메인 스레드(main thread)

Java 프로그램을 실행하면 main 메소드부터 실행이 시작된다.

main 메소드부터 실행이 시작되는 실행 흐름을 메인 스레드라고 부른다.

## 실행 흐름 생성

Thread 클래스의 start 메소드를 호출하면 새 실행 흐름이 생성된다.

Thread 객체를 생성하는 것은 객체만 생성할 뿐이고, 새 실행 흐름이 생성되는 것은 아니다.

start 메소드를 호출할 때 새 실행 흐름이 생성된다.

## 구현 방법1

Thread 클래스를 상속하여 자식 클래스 ChildThread 클래스를 구현한다.

ChildThread 클래스에서 run 메소드를 재정의한다.

ChildThread 클래스의 객체를 생성한다.

ChildThread 객체의 start 메소드를 호출하면 새 실행 흐름이 생성되어 ChildThread 객체의 run 메소드를 실행하게 된다.

메인 스레드는 start 메소드 아래로 실행을 계속한다.

## 구현 방법2

Runnable 인터페이스를 구현한 ChildRunnable 클래스를 구현한다.

ChildRunnable 클래스에서 run 메소드를 재정의한다.

ChildRunnable 클래스 객체를 생성한다.

ChildRunnable 객체를 파라미터로 Thread 객체를 생성한다.

Thread 객체의 start 메소드를 호출하면 새 실행 흐름이 생성되어 ChildRunnable 객체의 run 메소드를 실행하게 된다.

메인 스레드는 start 메소드 아래로 실행을 계속한다.

# 스레드 생성 예제

## 스레드 생성 방법1

Example01.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | package net.skhu.thread1;  import java.util.Random;  class ThreadA extends Thread {  String name;  Random random;  public ThreadA(String name) {  this.name = name;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 0; i < 100; ++i) {  System.out.printf("%s: %d\n", name, i);  int mili\_second = random.nextInt(100);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  }  }  public class Example01 {  public static void main(String args[]) {  Thread threadOne = new ThreadA("one");  Thread threadTwo = new ThreadA("two");  Thread threadThree = new ThreadA("three");  threadOne.start();  threadTwo.start();  threadThree.start();  }  } |

줄36까지 실행 흐름은 메인 스레드 하나 뿐이다.

줄37에서 start 메소드를 호출하면 실행 흐름이 하나 더 생성된다. 새로 생성된 실행 흐름은 threadOne 객체의 run 메소드를 실행한다. run 메소드에서 리턴되면 새 실행 흐름은 종료된다.

새 실행 흐름이 run 메소드를 실행하고 있을 때, 메인 스레드는 그 다음 줄38로 넘어간다.

줄38에서 start 메소드를 호출하면 실행 흐름이 하나 더 생성된다. 새로 생성된 실행 흐름은 threadTwo 객체의 run 메소드를 실행한다. run 메소드에서 리턴되면 새 실행 흐름은 종료된다.

새 실행 흐름이 run 메소드를 실행하고 있을 때, 메인 스레드는 그 다음 줄39로 넘어간다.

줄39에서 start 메소드를 호출하면 실행 흐름이 하나 더 생성된다. 새로 생성된 실행 흐름은 threadThtree 객체의 run 메소드를 실행한다. run 메소드에서 리턴되면 새 실행 흐름은 종료된다.

줄40에서 메인 스레드는 종료된다.

## 스레드 생성 방법2

Example02.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | package net.skhu.thread1;  import java.util.Random;  class RunnableA implements Runnable {  String name;  Random random;  public RunnableA(String name) {  this.name = name;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 0; i < 100; ++i) {  System.out.printf("%s: %d\n", name, i);  int mili\_second = random.nextInt(100);  Thread.sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  }  }  public class Example02 {  public static void main(String args[]) {  Thread threadOne = new Thread(new RunnableA("one"));  Thread threadTwo = new Thread(new RunnableA("two"));  Thread threadThree = new Thread(new RunnableA("three"));  threadOne.start();  threadTwo.start();  threadThree.start();  }  } |

Example01 과 Example02 의 구현은 아무 차이가 없다.

둘 중의 어떤 방식으로 구현해도 된다.

# 스레드 종료 시점

## Example03.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package net.skhu.thread1;  import java.util.Random;  class ThreadB extends Thread {  String name;  int count;  int sum;  Random random;  public ThreadB(String name, int count) {  this.name = name;  this.count = count;  this.sum = 0;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  sum = sum + i;  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: %d\n", name, sum);  System.out.printf("%s: stoped\n", name);  }  }  public class Example03 {  public static void main(String args[]) {  System.out.println("main started");  Thread threadOne = new ThreadB("one", 100);  Thread threadTwo = new ThreadB("two", 100);  Thread threadThree = new ThreadB("three", 100);  threadOne.start();  threadTwo.start();  threadThree.start();  System.out.println("main stoped");  }  } |

1에서 100까지 더하면 5050 이다.

스레드 3 개를 만들어서 이 계산을 3번 반복하여 출력한다.

실행 결과 출력

|  |
| --- |
| main started  main stoped  three: started  one: started  two: started  one: 5050  one: stoped  two: 5050  two: stoped  three: 5050  three: stoped |

## Example04.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | package net.skhu.thread1;  import java.util.Random;  class ThreadC extends Thread {  String name;  int count;  int sum;  Random random;  public ThreadC(String name, int count) {  this.name = name;  this.count = count;  this.sum = 0;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  sum = sum + i;  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  }  }  public class Example04 {  public static void main(String args[]) {  System.out.println("main started");  ThreadC threadOne = new ThreadC("one", 100);  ThreadC threadTwo = new ThreadC("two", 100);  ThreadC threadThree = new ThreadC("three", 100);  threadOne.start();  threadTwo.start();  threadThree.start();  System.out.printf("one: %d\n", threadOne.sum);  System.out.printf("two: %d\n", threadTwo.sum);  System.out.printf("three: %d\n", threadThree.sum);  System.out.println("main stoped");  }  } |

실행 결과 출력

|  |
| --- |
| main started  one: 0  two: 0  three: 0  main stoped  three: started  two: started  one: started  three: stoped  one: stoped  two: stoped |

메인 스레드가 줄45~47을 실행할 때, 계산 스레드는 아직 시작도 하지 않았다.

그래서 sum 멤버 변수 값이 0 이다.

계산 결과를 main 스레드가 출력하려면, 계산 스레드가 종료될 때까지 기다렸다가 출력해야 한다.

## Example05.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | package net.skhu.thread1;  import java.util.Random;  class ThreadD extends Thread {  String name;  int count;  int sum;  Random random;  public ThreadD(String name, int count) {  this.name = name;  this.count = count;  this.sum = 0;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  sum = sum + i;  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  }  }  public class Example05 {  public static void main(String args[]) {  System.out.println("main started");  ThreadD threadOne = new ThreadD("one", 100);  ThreadD threadTwo = new ThreadD("two", 100);  ThreadD threadThree = new ThreadD("three", 100);  threadOne.start();  threadTwo.start();  threadThree.start();  try {  threadOne.join();  threadTwo.join();  threadThree.join();  } catch (InterruptedException e) {  System.out.println("interrupted");  }  System.out.printf("one: %d\n", threadOne.sum);  System.out.printf("two: %d\n", threadTwo.sum);  System.out.printf("three: %d\n", threadThree.sum);  System.out.println("main stoped");  }  } |

**Thread 클래스의 join 메소드**

메인 스레드가 스레드 객체의 join 메소드를 호출하면, 그 스레드의 실행흐름이 run 메소드를 리턴하여 종료될 때까지 main 메소드는 대기하다가 리턴한다.

실행 결과 출력

|  |
| --- |
| main started  one: started  three: started  two: started  three: stoped  one: stoped  two: stoped  one: 5050  two: 5050  three: 5050  main stoped |

# 공유 변수

## Example06.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | package net.skhu.thread1;  import java.util.Random;  class ThreadE extends Thread {  String name;  int count;  static int sum = 0;  Random random;  public ThreadE(String name, int count) {  this.name = name;  this.count = count;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  sum = sum + i;  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  System.out.printf("%s: %d\n", name, sum);  }  }  public class Example06 {  public static void main(String args[]) {  ThreadE threadOne = new ThreadE("one", 100);  ThreadE threadTwo = new ThreadE("two", 100);  ThreadE threadThree = new ThreadE("three", 100);  threadOne.start();  threadTwo.start();  threadThree.start();  }  } |

sum 변수는 static 변수이다. 그래서 계산 스레드들이 sum 변수를 공유한다.

여러 스레드가 공유 변수를 동시에 읽고 쓰는 것은 문제가 발생할 수 있다.

그래서 계산 결과 값이 5050 x 3 = 15150 이어야 하지만, 그보다 작은 값이 될 것이다.

## Example07.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | package net.skhu.thread1;  import java.util.Random;  class ThreadF extends Thread {  String name;  int count;  static int sum = 0;  Random random;  public ThreadF(String name, int count) {  this.name = name;  this.count = count;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  synchronized(ThreadF.class) {  sum = sum + i;  }  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  System.out.printf("%s: %d\n", name, sum);  }  }  public class Example07 {  public static void main(String args[]) {  ThreadF threadOne = new ThreadF("one", 100);  ThreadF threadTwo = new ThreadF("two", 100);  ThreadF threadThree = new ThreadF("three", 100);  threadOne.start();  threadTwo.start();  threadThree.start();  }  } |

여러 스레드가 공유하는 변수를 읽고 쓸 때에는 락(lock)이 필요하다.

sum 변수가 ThreadF 클래스의 클래스 변수이기 때문에 아래의 코드로 락을 걸면 된다.

synchronized (ThreadF.class) {

. . .

}

락이 걸리면 다른 스레드는 그 부분을 실행할 수 없기 때문에 안전하다.

클래스 멤버 변수는 인스턴스 객체에 들어있지 않고, 그 클래스 객체에 들어있다.

즉 ThreadF.class 객체에 들어있다고 보면 된다.

그래서 ThreadF.class 객체에 락을 걸어야 한다.

실행결과 출력

|  |
| --- |
| one: started  three: started  two: started  one: stoped  one: 14286  three: stoped  three: 14378  two: stoped  two: 15150 |



# 공유 객체

## Example08.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package net.skhu.thread1;  import java.util.Random;  class SharedObjA {  int sum = 0;  void add(int i) {  sum = sum + i;  }  }  class ThreadH extends Thread {  String name;  int count;  SharedObjA sharedObj;  Random random;  public ThreadH(String name, int count, SharedObjA sharedObj) {  this.name = name;  this.count = count;  this.sharedObj = sharedObj;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  sharedObj.add(i);  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  System.out.printf("%s: %d\n", name, sharedObj.sum);  }  }  public class Example08 {  public static void main(String args[]) {  SharedObjA sharedObj = new SharedObjA();  ThreadH threadOne = new ThreadH("one", 100, sharedObj);  ThreadH threadTwo = new ThreadH("two", 100, sharedObj);  ThreadH threadThree = new ThreadH("three", 100, sharedObj);  threadOne.start();  threadTwo.start();  threadThree.start();  }  } |

줄47에서 생성된 SharedObjA 클래스 객체를 3 개의 계산 스레드가 공유한다.

공유되는 객체의 멤버 변수를 동시에 읽고 쓰는 것은 문제가 발생할 수 있다.



## Example09.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | package net.skhu.thread1;  import java.util.Random;  class SharedObjB {  int sum = 0;  synchronized void add(int i) {  sum = sum + i;  }  }  class ThreadJ extends Thread {  String name;  int count;  SharedObjB sharedObj;  Random random;  public ThreadJ(String name, int count, SharedObjB sharedObj) {  this.name = name;  this.count = count;  this.sharedObj = sharedObj;  this.random = new Random();  }  @Override  public void run() {  System.out.printf("%s: started\n", name);  try {  for (int i = 1; i <= count; ++i) {  sharedObj.add(i);  int mili\_second = random.nextInt(10);  sleep(mili\_second);  }  } catch (InterruptedException e) {  System.out.printf("%s: interrupted\n", name);  }  System.out.printf("%s: stoped\n", name);  System.out.printf("%s: %d\n", name, sharedObj.sum);  }  }  public class Example09 {  public static void main(String args[]) {  SharedObjB sharedObj = new SharedObjB();  ThreadJ threadOne = new ThreadJ("one", 100, sharedObj);  ThreadJ threadTwo = new ThreadJ("two", 100, sharedObj);  ThreadJ threadThree = new ThreadJ("three", 100, sharedObj);  threadOne.start();  threadTwo.start();  threadThree.start();  }  } |

줄47에서 생성된 SharedObjB 클래스 객체를 3 개의 계산 스레드가 공유한다.

공유되는 객체의 멤버 변수를 동시에 읽고 쓰는 것은 문제가 발생할 수 있다.

공유되는 객체의 메소드에 synchronized 키워드를 붙이면, 이 메소드를 호출할 때, 이 객체에 락(lock)이 걸리게 된다. 그래서 어느 한 순간 하나의 스레드만 이 객체의 메소드를 호출할 수 있어서 안전하다.

# 실습

SharedObject 클래스를 구현하라.

sum 멤버 변수

add 메소드: 파라미터 값을 sum 변수에 더하기

sub 메소드: 파라미터 값을 sum 변수에서 빼기

스레드를 두 개 만들라.

두 스레드가 SharedObject 객체 하나를 공유하라.

스레드 하나는 SharedObject 객체의 add 메소드를 호출하여 1부터 100까지 값을 더하라.

다른 스레드는 SharedObject 객체의 sub 메소드를 호출하여 1부터 100까지 값을 빼라.

두 스레드가 종료한 뒤, SharedObject 객체의 sum 변수 값이 0 임을 확인하라.

### 구현#1

|  |
| --- |
| package net.skhu.thread.v2;  class SharedObjD {  int sum = 0;  synchronized void add(int i) { sum = sum + i; }  synchronized void sub(int i) { sum = sum - i; }  }  class ThreadK1 extends Thread {  String name;  int count;  SharedObjC sharedObj;  public ThreadK1(String name, int count, SharedObjC sharedObj) {  this.name = name;  this.count = count;  this.sharedObj = sharedObj;  }  @Override  public void run() {  for (int i = 1; i <= count; ++i)  sharedObj.add(i);  }  }  class ThreadK2 extends Thread {  String name;  int count;  SharedObjC sharedObj;  public ThreadK2(String name, int count, SharedObjC sharedObj) {  this.name = name;  this.count = count;  this.sharedObj = sharedObj;  }  @Override  public void run() {  for (int i = 1; i <= count; ++i)  sharedObj.sub(i);  }  }  public class Example10 {  public static void main(String args[]) {  SharedObjC sharedObj = new SharedObjC();  ThreadK1 threadOne = new ThreadK1("one", 100, sharedObj);  ThreadK2 threadTwo = new ThreadK2("two", 100, sharedObj);  threadOne.start();  threadTwo.start();  try {  threadOne.join();  threadTwo.join();  } catch (InterruptedException e) {  }  System.out.println(sharedObj.sum);  }  } |

### 구현#2

|  |
| --- |
| package net.skhu.thread.v2;  class SharedObjC {  int sum = 0;  synchronized void add(int i) {  sum = sum + i;  }    synchronized void sub(int i) {  sum = sum - i;  }  }  class ThreadK extends Thread {  public enum Operation { ADD, SUB };  String name;  int count;  SharedObjD sharedObj;  Operation operation;  public ThreadK(String name, int count, SharedObjD sharedObj, Operation operation) {  this.name = name;  this.count = count;  this.sharedObj = sharedObj;  this.operation = operation;  }  @Override  public void run() {  for (int i = 1; i <= count; ++i)  if (operation == Operation.ADD)  sharedObj.add(i);  else  sharedObj.sub(i);  }  }  public class Example11 {  public static void main(String args[]) {  SharedObjD sharedObj = new SharedObjD();  ThreadK threadOne = new ThreadK("one", 100, sharedObj, ThreadK.Operation.ADD);  ThreadK threadTwo = new ThreadK("two", 100, sharedObj, ThreadK.Operation.SUB);  threadOne.start();  threadTwo.start();  try {  threadOne.join();  threadTwo.join();  } catch (InterruptedException e) {  }  System.out.println(sharedObj.sum);  }  } |