|  |  |
| --- | --- |
| Object obj = new Object() {  synchronized( obj ) {  // apenas um thread poderá entrar aqui de cada vez  } |  |
| public class Repository {  private StringBuilder buffer = new StringBuilder();  private String prefix;  ...  public void writeData(String text) {  **synchronized(buffer)** { // a trava é obtida para o objeto buffer  buffer.append(this.prefix);  buffer.append(text);  buffer.append(this.suffix);  }  }  } | public class Repository {  private StringBuilder buffer = new StringBuilder(); ...  public void writeData(String text) {  **synchronized(this)** { // a trava é obtida para um objeto Repositorio  buffer.append(this.prefix);  buffer.append(text);  buffer.append(this.suffix);  }  }  } |
| public class Repository {  private StringBuilder buffer = new StringBuilder();  ...  public synchronized void writaData(String prefix, String text, String suffix) {  buffer.append(prefix);  buffer.append(text);  buffer.append(suffix);  }  public synchronized String readData() {  return buffer.toString()  }  } |  |
| class SharedResource {  public synchronized void synchronizedMethod() {  System.out.println("synchronizedMethod(): " + Thread.holdsLock(this));  }  public void method() {  System.out.println("method(): " + Thread.holdsLock(this));  }  } | SharedResource obj = new SharedResource();  synchronized (obj) {  obj.method(); // holdsLock = true  }  obj.method(); // holdsLock = false |
| public class WordFinder implements Runnable {  public volatile boolean done;  public volatile int count;  public List<String> words = new Collections.concurrentList(ArrayList<>());  public void run() {  while(!done) { // leitura feita por thread secundário  words.add(lookForWord("thread")); // alteração feita por thread secundário  count = words.size();  }  }  }  public class SharedObject {  private volatile int value = -1;  public boolean isSet() { return value != -1; }  public synchronized boolean set(int v) {  try {  while(isSet()) { // enquanto houver valor definido, espere  wait();  }  value = v;  System.out.println(Thread.currentThread().getName() + ": PRODUCED: " + value);  notifyAll(); // avisa a produtores e consumidores (notify() avisa a um deles)  return true;  } catch (InterruptedException e) { return false; }  }  public synchronized boolean reset() {  try {  while (!isSet()) { // enquanto não houver valor definido, espere  wait();  }  System.out.println(Thread.currentThread().getName() + ": CONSUMED: " + value);  value = -1;  notifyAll(); // avisa a todos os threads  return true;  } catch (InterruptedException e) { return false; }  }  } | public class TestWordFinder {  public static void main(String[] args) {  WordFinder runnable = new WordFinder(...);  new Thread(runnable).start();  while(runnable.count <= 5) { // leitura feita por thread main  runnable.done = false; // alteração feita por thread main  }  runnable.done = true; // alteração feita por thread main  }  }  public class Producer implements Runnable {  private SharedObject shared;  private static final int TENTATIVAS = 3;  Producer(SharedObject shared) { this.shared = shared; }  @Override public void run() {  for (int i = 0; i < TENTATIVAS; i++) {  if( !shared.set(new Random().nextInt(1000)) )  break; // termina o thread se set() retornar false (foi interrompido)  }  System.out.println(Thread.currentThread().getName() + ": Producer DONE.");  }  } |
| public class Consumer implements Runnable {  private SharedObject shared;  private static final int TENTATIVAS = 3;  Consumer(SharedObject shared) { this.shared = shared; }  @Override public void run() {  for (int i = 0; i < TENTATIVAS; i++) {  if(!shared.reset())  break; // termina thread se retornar false (foi interrompido)  }  System.out.println(Thread.currentThread().getName() + ": Consumer DONE.");  }  } |  |
| public class ProducerConsumerExample {  public static void main(String[] args) {  SharedObject o = new SharedObject();  String[] names = {"C1", "C2", "P1", "P2"};  Thread[] threads = { new Thread(new Consumer(o)), new Thread(new Consumer(o)),  new Thread(new Producer(o)), new Thread(new Producer(o)) };  for(int i = 0; i < threads.length; i++) {  threads[i].setName(names[i]);  threads[i].start();  }  try {  for(Thread t: threads) {  t.join(15000); // will wait up to 15 seconds for each thread to finish  if(t.isAlive()) { t.interrupt(); }  }  } catch (InterruptedException ignored) {}  System.out.println("Main DONE.");  }  } |  |
|  |  |