Escopos Sincronizados e Travas

Fernando Castor

Professor Adjunto

Centro de Informática

Universidade Federal de Pernambuco

castor@cin.ufpe.br

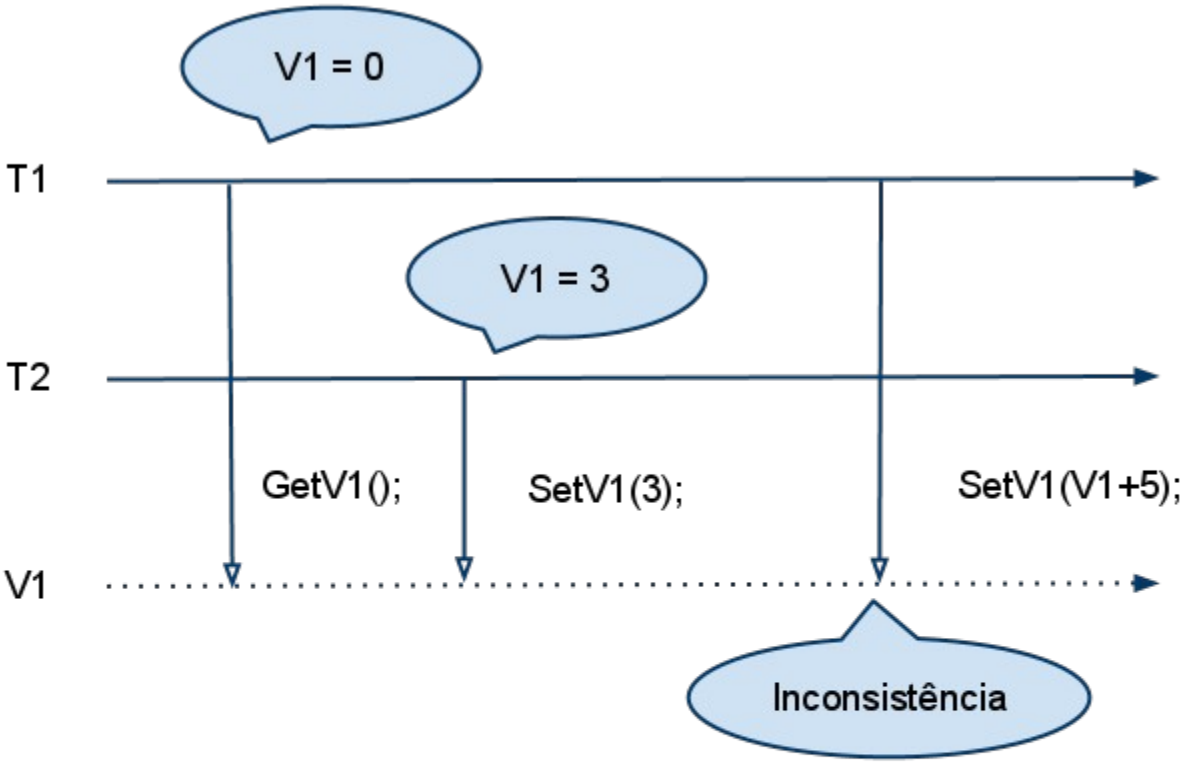
Elaborado conjuntamente por:

Benito Fernandes, João Paulo Oliveira e Weslley Torres

Exclusão mútua

- Java oferece duas abordagens básicas de exclusão mútua:
  - Métodos sincronizados
  - Blocos sincronizados

Problemas da falta de sincronização



Monitores

- Objetos que garantem a exclusão mútua na execução dos procedimentos associados a eles.
  - apenas um procedimento associado ao monitor pode ser executado em um determinado momento

- Em Java todo objeto possui um monitor associado.

Métodos sincronizados

- Estratégia simples de prevenção de interferência entre `threads`
- Também previnem erros de consistência de memória
- Podem ser caros, dependendo da aplicação

Métodos sincronizados – exemplos

```
public class ContadorSincronizado {

        private static int c = 0;

        public synchronized void incrementar() {
            c++;
        }

        public synchronized void decrementar() {
            c--;
        }

        public synchronized int getC() {
            return c;
        }
}
```

Métodos sincronizados

```
public class ThreadSincronizada extends Thread {
    private ContadorSincronizado contador = new ContadorSincronizado();
    private int repeticao;
    private String nome;

    public void run(){
        if(nome=="thread1")
            while (repeticao>0){
                this.contador.incrementar();
                System.out.println("thread1 valor do contador " + this.contador.getC());
                this.repeticao--;
            }
        else
            while (repeticao>0){
                this.contador.decrementar();
                System.out.println("thread2 valor do contador " + this.contador.getC());
                this.repeticao--;
            }
    }

    public void setRepeticao(int repeticao){
        this.repeticao = repeticao;
    }

    public void setNome(String nome){
        this.nome = nome;
    }
}
```

Métodos sincronizados

```
public class testarSincronizacao {
    public static void main(String[] args) {

        ThreadSincronizada task1 = new ThreadSincronizada();
        ThreadSincronizada task2 = new ThreadSincronizada();

        task1.setRepeticao(10);
        task2.setRepeticao(10);
        task1.setNome("thread1");
        task2.setNome("OutraThread");

        task1.start();
        task2.start();
    }
}
```

Resultados

```
thread2 valor do contador -1
thread2 valor do contador -2
thread2 valor do contador -3
thread2 valor do contador -4
thread2 valor do contador -5
thread2 valor do contador -6
thread2 valor do contador -7
thread2 valor do contador -8
thread2 valor do contador -9
thread2 valor do contador -10
thread1 valor do contador -9
thread1 valor do contador -8
thread1 valor do contador -7
thread1 valor do contador -6
thread1 valor do contador -5
thread1 valor do contador -4
thread1 valor do contador -3
thread1 valor do contador -2
thread1 valor do contador -1
thread1 valor do contador 0
```

```
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
thread1 valor do contador 1
thread2 valor do contador 0
```

```
thread1 valor do contador 1
thread1 valor do contador 1
thread1 valor do contador 2
thread1 valor do contador 3
thread1 valor do contador 4
thread1 valor do contador 5
thread1 valor do contador 6
thread1 valor do contador 7
thread1 valor do contador 8
thread1 valor do contador 9
thread2 valor do contador 0
thread2 valor do contador 8
thread2 valor do contador 7
thread2 valor do contador 6
thread2 valor do contador 5
thread2 valor do contador 4
thread2 valor do contador 3
thread2 valor do contador 2
thread2 valor do contador 1
thread2 valor do contador 0
```

Blocos sincronizados

- Reduzir o custo de sincronização associado ao método.

- Podem combinar mais de um objeto no momento da sincronização.

Exemplo de blocos sincronizados

```
class Parenteses {

  public void exibir(String s) {
    System.out.print("(" + s);
    try {
      Thread.sleep(1000);
    }
    catch(InterruptedException e) {
        System.out.println("Interrupted");
    }
    System.out.println(")");
  }
}
```

Exemplo de blocos sincronizados

```
class MinhaThread implements Runnable {

  String s1;
  Parenteses p1;
  public MinhaThread (Parenteses p2, String s2) {
    p1 = p2;
    s1 = s2;
  }
  public void run() {
    synchronized(p1) {
      p1.exibir(s1);
    }
  }
}
```

Exemplo de blocos sincronizados

```
public class Demo {
  public static void main(String args[]) {
    Parenteses p3 = new Parenteses();
    Thread n1 = new Thread(new MinhaThread(p3, "Bob"));
    Thread n2 = new Thread(new MinhaThread(p3, "Maria"));
    n1.start();
    n2.start();
    try {
      n1.join();
      n2.join();
    } catch(InterruptedException ie) {
      System.out.println("Interrupted");
    }
  }
}
```

Exemplo de blocos sincronizados

```
private void metodo(Object parametro) {
        //Validação não sincronizada
        if (parametro == null)
            throw new IllegalArgumentException();

        //Parte duplamente sincronizada
        synchronized (chave){
            synchronized (this) {
                    //Método com dupla sincronização
            }
        }
}
```

Como corrigir o exemplo a seguir?

```
public abstract class ContadorErrado
    implements Runnable {
  private static int contador = 0;

  public void incrementar() {
    contador++;
  }
  public void decrementar(){
    contador--;
  }
  public static void main(String args[]) {
    ContadorErrado c1 = new ContadorErrado() {
      public void run() {
        for(int i = 0; i < 10000; i++) {
          this.incrementar(); }}};

  //(continua...)
```

```
    ContadorErrado c2 = new ContadorErrado() {
      public void run() {
        for(int i = 0; i < 10000; i++) {
          this.decrementar();
        }
      }
    };
    Thread t1 = new Thread(c1);
    Thread t2 = new Thread(c2);
    t1.start(); t2.start();
    try {
      t1.join();
      t2.join();
    } catch(InterruptedException e) {}
    System.out.println(c1.contador);
  }
}
```

Travas (*locks*) Explícitas

Locks

- A interface Lock pode ser usada no lugar de blocos sincronizados
- Oferecem maior flexibilidade.
  - Tratamento e destravamento explícitos
- Mais cuidado é necessário, porém
- Para criar um lock explícito, você instancia uma implementação da interface Lock.
  - Normalmente, instancia-se `ReentrantLock`

Locks

- Para obter o lock, usa-se o método `lock()`
- Para liberar o lock, usa-se `unlock()`

Locks

- Já que o `lock` não é liberado automaticamente, é essencial envolver o corpo do método com um bloco `try/finally`

– garante que o `lock` é liberado

```
public void put(Object x) throws InterruptedException {
  lock.lock();
  try {
    while (count == items.length)
      notFull.await();
    items[putptr] = x;
    if (++putptr == items.length) putptr = 0;
    ++count;
    notEmpty.signal();
  } finally {
    lock.unlock();
  }
}
```

Sintaxe básica

```
Lock lck = ...

lck.lock();
try {
   ... Acessar recursos compartilhados
} finally {
  lck.unlock();
}
```

Exemplo 1

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ConcurrentIntegerArray {
      private final Lock lock = new ReentrantLock();
      private final int[] arr;

      public ConcurrentIntegerArray(int size) {
        arr = new int[size];
      }
      public void set(int index, int value) {
        lock.lock();
        try {
          arr[index] = value;
        } finally {
          lock.unlock();
        }
      }
      public int get(int index) {
        lock.lock();
        try {
          return arr[index];
        } finally {
          lock.unlock();
        }
      }
}
```

Exemplo 1

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ConcurrentIntegerArray {
      private final Lock lock = new ReentrantLock();
      private final int[] arr;

      public ConcurrentIntegerArray(int size) {
        arr = new int[size];
      }
      public void set(int index, int value) {
        lock.lock();
        try {
          arr[index] = value;
        } finally {
          lock.unlock();
        }
      }
      public int get(int index) {
        lock.lock();
        try {
          return arr[index];
        } finally {
          lock.unlock();
        }
      }
}
```

Bloqueia a thread caso não obtenha o *lock*

Exemplo 2

```
public class Safelock {
    static class Friend {
        private final String name;
        private final Lock lock = new ReentrantLock();

        public Friend(String name) {
            this.name = name;
        }

        public String getName() {
            return this.name;
        }

        public boolean impendingBow(Friend bower) {
            Boolean myLock = false;
            Boolean yourLock = false;
            try {
                myLock = lock.tryLock();
                yourLock = bower.lock.tryLock();
            } finally {
                if (! (myLock && yourLock)) {
                    if (myLock) {
                        lock.unlock();
                    }
                    if (yourLock) {
                        bower.lock.unlock();
                    }
                }
            }
            return myLock && yourLock;
        }
```

Continua

Exemplo 2

```
public class Safelock {
    static class Friend {
        private final String name;
        private final Lock lock = new ReentrantLock();

        public Friend(String name) {
            this.name = name;
        }

        public String getName() {
            return this.name;
        }

        public boolean impendingBow(Friend bower) {
            Boolean myLock = false;
            Boolean yourLock = false;
            try {
                myLock = lock.tryLock();
                yourLock = bower.lock.tryLock();
            } finally {
                if (! (myLock && yourLock)) {
                    if (myLock) {
                        lock.unlock();
                    }
                    if (yourLock) {
                        bower.lock.unlock();
                    }
                }
            }
            return myLock && yourLock;
        }
```

A thread não é bloqueada caso o lock não esteja disponível

Continua

Exemplo 2

```
    public void bow(Friend bower) {
        if (impendingBow(bower)) {
            try {
                System.out.format("%s: %s has bowed to me!%n",
                        this.name, bower.getName());
                bower.bowBack(this);
            } finally {
                lock.unlock();
                bower.lock.unlock();
            }
        } else {
            System.out.format("%s: %s started to bow to me, but" +
                    " saw that I was already bowing to him.%n",
                    this.name, bower.getName());
        }
    }

    public void bowBack(Friend bower) {
        System.out.format("%s: %s has bowed back to me!%n",
                this.name, bower.getName());
    }
}
```

Continua

Exemplo 2

```
    static class BowLoop implements Runnable {
        private Friend bower;
        private Friend bowee;

        public BowLoop(Friend bower, Friend bowee) {
            this.bower = bower;
            this.bowee = bowee;
        }

        public void run() {
            Random random = new Random();
            for (;;) {
                try {
                    Thread.sleep(random.nextInt(10));
                } catch (InterruptedException e) {}
                bowee.bow(bower);
            }
        }
    }


    public static void main(String[] args) {
        final Friend alphonse = new Friend("Alphonse");
        final Friend gaston = new Friend("Gaston");
        new Thread(new BowLoop(alphonse, gaston)).start();
        new Thread(new BowLoop(gaston, alphonse)).start();
    }
}
```

Resultado

```
Alphonse: Gaston has bowed to me!
Gaston: Alphonse has bowed back to me!
Gaston: Alphonse has bowed to me!
Alphonse: Gaston has bowed back to me!
Gaston: Alphonse has bowed to me!
Alphonse: Gaston has bowed back to me!
Alphonse: Gaston has bowed to me!
Gaston: Alphonse has bowed back to me!
Gaston: Alphonse has bowed to me!
Alphonse: Gaston has bowed back to me!
Alphonse: Gaston started to bow to me, but saw that I was already bowing to him.
Gaston: Alphonse has bowed to me!
Alphonse: Gaston has bowed back to me!
Alphonse: Gaston has bowed to me!
Gaston: Alphonse has bowed back to me!
Alphonse: Gaston has bowed to me!
Gaston: Alphonse has bowed back to me!
Gaston: Alphonse has bowed to me!
```