Corrompimento de variável global compartilhada usando linguagem Java

Matheus S. Heck1

1Universidade do Vale do Itajaí - UNIVALI

mtequinho@gmail.com / matheus.heck@edu.univali.br / matheus@qualitys.com.br

**Abstract.** This report aims to try and prove the need to use thread protections to avoid data loss and inconsistency at the end of the program's execution.

**Resumo.** Este relatório visa experimentar e provar a necessidade de utilização de proteções em threads para que não ocorra a perda de dados e a inconsistência ao final da execução do programa.

# 1. Apresentação

A atividade proposta pelos professores da disciplina de Sistemas Operacionais procura entender a importância, na utilização de threads, de proteger as variáveis para que não ocorram erros, sobreposições ou leituras erradas do programa e, com isso, resolver as operações de forma correta e devolver para o usuário as informações corretas.

A ideia do projeto era simular um sistema de banco onde seriam feitas 5000 transações de valor 1, tendo o resultado que ser igual a 5000 transações e 5000 unidades de saldo.

O trabalho foi desenvolvido em duas etapas, sendo elas:

-1ª: implementação do sistema sem a proteção dos dados e variáveis, “forçando” o erro do programa;

-2ª: implementação do sistema com proteção de dados e variáveis, deixando o sistema robusto e sem problemas de gravação e leitura dos dados expostos.

# 2. Implementação

Para a implementação desse projeto, foi utilizada a IDE desenvolvida pela empresa *JetBrains,* denominada *IntelliJ*. Foi usada a linguagem JAVA para fazer e o JAVA Development Kit versão 14.0.1.

O projeto foi feito em 4 classes JAVA, sendo elas:

-SafeDepositador(): classe que estende a classe Thread nativa e é responsável por proteger as variáveis e não deixar que ocorra perda de dados.

-UnsafeDepositador(): classe que estende a classe Thread nativa e é responsável por realizar as transações de forma insegura, podendo ocorrer a perda de dados.

-SafeOperation(): classe que implementa o método Main(), e onde são instanciadas 5 instâncias da classe SafeDepositador().

-UnsafeOperation(): classe que implementa o método Main(), e onde são instanciadas 5 instâncias da classe UnsafeDepositador().

A implementação ficou da seguinte forma:

-SafeDepositador():

package desktop.heck.universidade.so;  
  
public class SafeDepositador extends Thread{  
 public static int *balance* = 0;  
 public static int *transactions* = 1000;  
 public static int *totalTransactions* = 0;  
  
  
 public SafeDepositador(){  
 *balance* = 0;  
 }  
  
 public static synchronized void safeIncrement(){  
 *balance*++;  
 *totalTransactions*++;  
 }  
  
 @Override  
 public void run() {  
 for (int i=0; i<*transactions*; i++){  
 *safeIncrement*();  
 }  
 }  
}

-UnsafeDepositador():

package desktop.heck.universidade.so;  
  
public class UnsafeDepositador extends Thread{  
 public static int *balance* = 0;  
 public static int *transactions*=1000;  
 public static int *totalTransactions* = 0;  
  
  
 public UnsafeDepositador(){  
 *balance* = 0;  
 }  
  
 public static void unsafeIncrement(){  
 *balance*++;  
 *totalTransactions*++;  
 }  
  
 @Override  
 public void run() {  
 for (int i=0; i<*transactions*; i++){  
 *unsafeIncrement*();  
 }  
 }  
}

-SafeOperation():

package desktop.heck.universidade.so;  
  
public class SafeOperation {  
 public static void main(String[] args) {  
  
 SafeDepositador d1 = new SafeDepositador();  
 SafeDepositador d2 = new SafeDepositador();  
 SafeDepositador d3 = new SafeDepositador();  
 SafeDepositador d4 = new SafeDepositador();  
 SafeDepositador d5 = new SafeDepositador();  
  
 d1.start();  
 d2.start();  
 d3.start();  
 d4.start();  
 d5.start();  
  
 try {  
 d1.join();  
 d2.join();  
 d3.join();  
 d4.join();  
 d5.join();  
  
 } catch (InterruptedException e) {  
 System.*out*.println("Oops, something get wrong...");  
 }  
  
 System.*out*.println("Operações seguras:");  
 System.*out*.println("Saldo final: " + SafeDepositador.*balance*);  
 System.*out*.println("Total de transações: " + SafeDepositador.*totalTransactions*);  
  
 }  
  
}

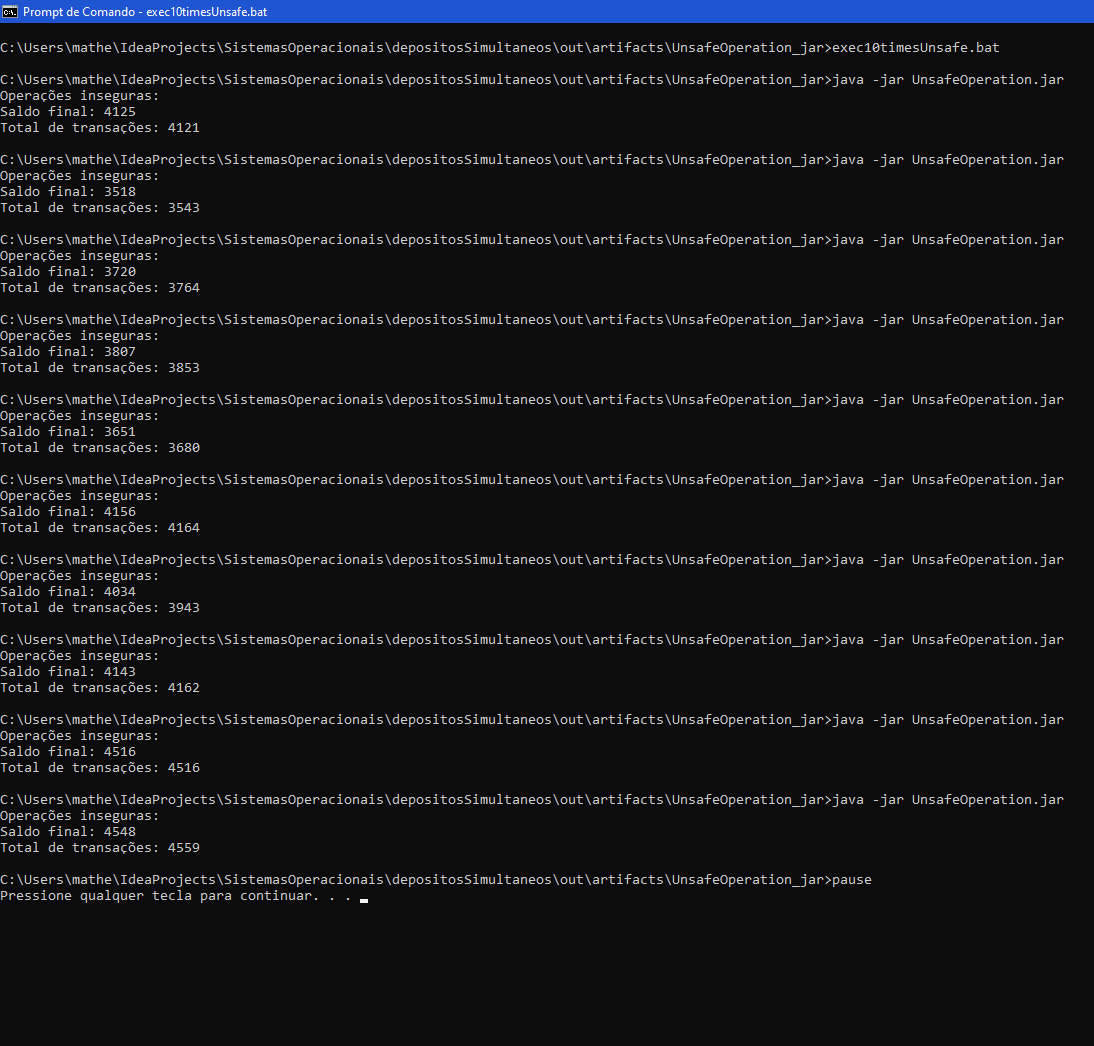
-UnsafeOperation():

package desktop.heck.universidade.so;  
  
public class UnsafeOperation extends Thread {  
  
 public static void main(String[] args) {  
 UnsafeDepositador d1 = new UnsafeDepositador();  
 UnsafeDepositador d2 = new UnsafeDepositador();  
 UnsafeDepositador d3 = new UnsafeDepositador();  
 UnsafeDepositador d4 = new UnsafeDepositador();  
 UnsafeDepositador d5 = new UnsafeDepositador();  
  
 d1.start();  
 d2.start();  
 d3.start();  
 d4.start();  
 d5.start();  
  
 try {  
 d1.join();  
 d2.join();  
 d3.join();  
 d4.join();  
 d5.join();  
  
 } catch (InterruptedException e){  
 System.*out*.println("Oops, something get wrong...");  
 }  
  
 System.*out*.println("Operações inseguras:");  
 System.*out*.println("Saldo final: " + UnsafeDepositador.*balance*);  
 System.*out*.println("Total de transações: " + UnsafeDepositador.*totalTransactions*);  
 }  
  
}

# 3. Resultados

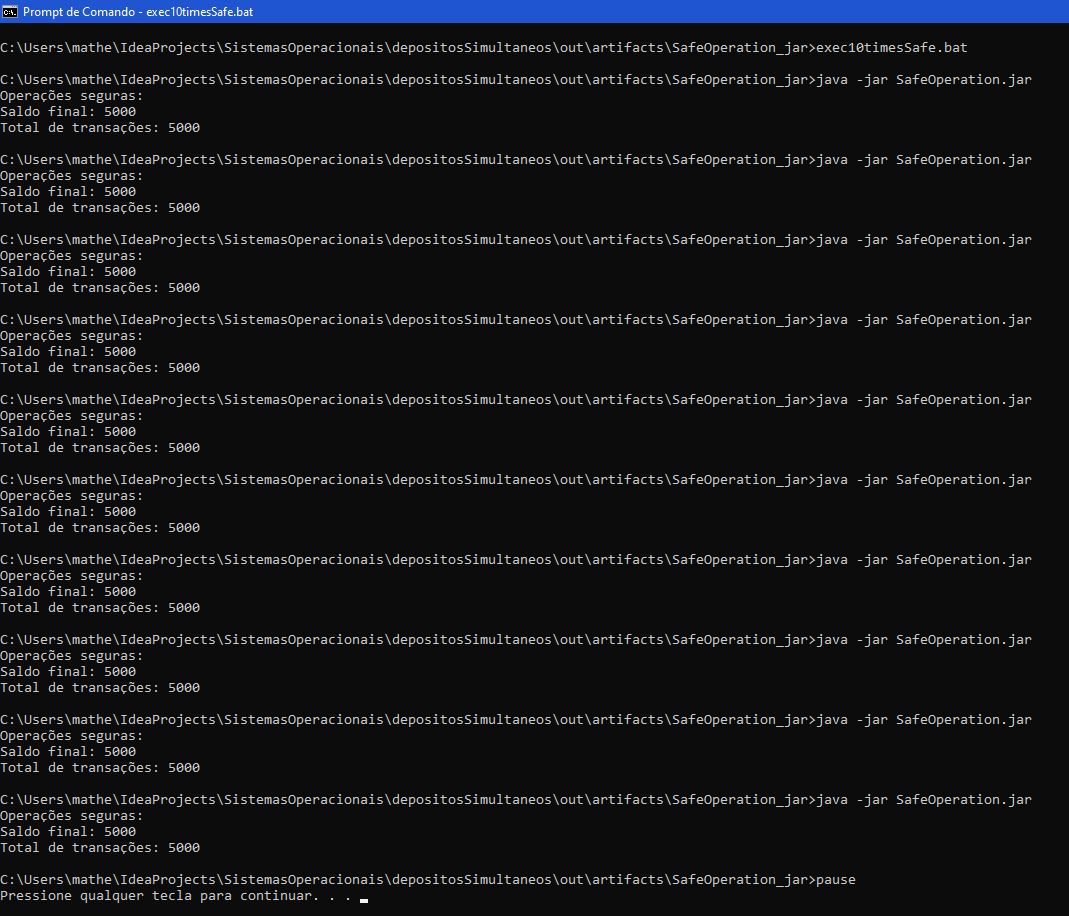
Para termos resultados mais consistentes, foi criado um arquivo .bat para cada arquivo .jar que roda o programa 10 vezes. Os resultados foram os seguintes:

-UnsafeOperation():



Como se pode perceber, em nenhum momento o número de operações e saldo sai como o esperado, pois as threads acabam sendo executadas ao mesmo tempo de clock, fazendo com que se tenha a perda de dados.

-SafeOperation():



Já na execução do programa com a devida proteção das variáveis e das threads, não ocorre nenhum erro, nem na parte do saldo nem na parte do numero de transações, fazendo assim com que se tenha uma segurança das variáveis e dos dados.

# 4. Conclusão

Com a realização deste trabalho, pode-se entender que o acesso a variáveis globais por meio de threads sem a segurança necessária, é um tanto quanto perigoso, pois não tem a total certeza de que os dados acessados e gravados serão corretos. Para resolver esta situação, deve-se utilizar mecanismos de proteção de variáveis, como os utilizados na implementação da classe SafeDepositador().