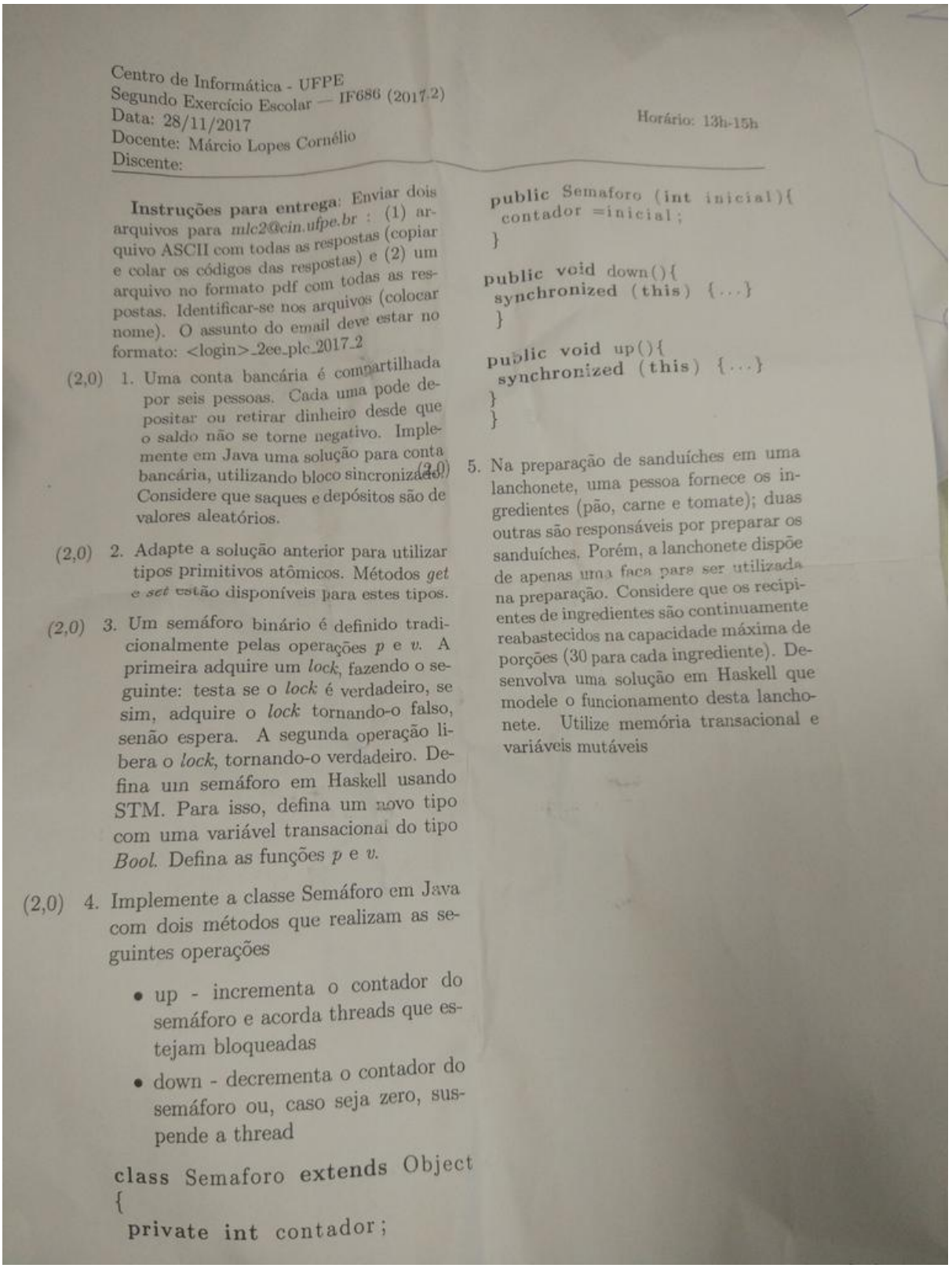
Exercícios, listas e provas EE2 e final

Outros conteúdos



Centro de Informática - UFPE
Segundo Exercício Escolar — IF686 (2017.2)
Data: 28/11/2017 Horário: 13h-15h
Docente: Márcio Lopes Cornélio
Discente:

Instruções para entrega: Enviar dois arquivos para *mlc2@cin.ufpe.br* : (1) arquivo ASCII com todas as respostas (copiar e colar os códigos das respostas) e (2) um arquivo no formato pdf com todas as respostas. Identificar-se nos arquivos (colocar nome). O assunto do email deve estar no formato: <login>_2ee_plc_2017_2

(2,0) 1. Uma conta bancária é compartilhada por seis pessoas. Cada uma pode depositar ou retirar dinheiro desde que o saldo não se torne negativo. Implemente em Java uma solução para conta bancária, utilizando bloco sincronizado. Considere que saques e depósitos são de valores aleatórios.

(2,0) 2. Adapte a solução anterior para utilizar tipos primitivos atômicos. Métodos *get* e *set* estão disponíveis para estes tipos.

(2,0) 3. Um semáforo binário é definido tradicionalmente pelas operações *p* e *v*. A primeira adquire um *lock*, fazendo o seguinte: testa se o *lock* é verdadeiro, se sim, adquire o *lock* tornando-o falso, senão espera. A segunda operação libera o *lock*, tornando-o verdadeiro. Defina um semáforo em Haskell usando STM. Para isso, defina um novo tipo com uma variável transacional do tipo *Bool*. Defina as funções *p* e *v*.

(2,0) 4. Implemente a classe Semáforo em Java com dois métodos que realizam as seguintes operações

- up - incrementa o contador do semáforo e acorda threads que estejam bloqueadas
- down - decrementa o contador do semáforo ou, caso seja zero, suspende a thread

```
class Semaforo extends Object
{
 private int contador;
 public Semaforo (int inicial){
  contador =inicial;
 }
 public void down(){
  synchronized (this) {...}
  }
 public void up(){
  synchronized (this) {...}
 }
}
```

(3,0) 5. Na preparação de sanduíches em uma lanchonete, uma pessoa fornece os ingredientes (pão, carne e tomate); duas outras são responsáveis por preparar os sanduíches. Porém, a lanchonete dispõe de apenas uma faca para ser utilizada na preparação. Considere que os recipientes de ingredientes são continuamente reabastecidos na capacidade máxima de porções (30 para cada ingrediente). Desenvolva uma solução em Haskell que modele o funcionamento desta lanchonete. Utilize memória transacional e variáveis mutáveis

Prova transcrita abaixo:

1) Uma conta bancária é compartilhada por seis pessoas. Cada uma pode depositar ou retirar dinheiro desde que o saldo não se torne negativo. Implemente em Java uma solução para conta bancária, utilizando bloco sincronizado. Considere que saques e depósitos são de valores aleatórios.

```
public class Principal {

	public static void main(String[] args) {
		Conta conta = new Conta();
		for(int i = 0; i<6; i++) {
			MinhaThread threadR = new MinhaThread("@"+i, conta);
			Thread tr = new Thread(threadR);
			tr.start();
		}
	}

}
```

```
public class Conta {
	private int saldo = 0;

	public void depositar (int saldo) {
		synchronized(this){
			this.saldo += saldo;
			System.out.println("Foi depositado "+ saldo);
			System.out.println("Saldo atual pós deposito igual a " +
this.saldo);
		}
	}
	public void retirar (int saldo) {
		synchronized(this) {
			if(this.saldo - saldo >= 0) {
				this.saldo -= saldo;
				System.out.println("Foi retirado "+ saldo);
				System.out.println("Saldo atual pós retirada igual a
" + this.saldo);
			}
```

```
            }
      }
}
```

```
import java.util.Random;

public class MinhaThread implements Runnable {
      private String nome;
      private Conta conta;

      public MinhaThread(String nome, Conta conta) {
            this.nome = nome;
            this.conta = conta;
      }

      public void run() {
            System.out.println("Startando a Thread: "+ this.nome);
            Random random = new Random();
            conta.depositar(random.nextInt(11));
            conta.retirar(random.nextInt(11));
      }
}
```

2) Adapte a solução anterior para utilizar tipos primitivos atômicos. Métodos get e set estão disponíveis para estes tipos.

```
package ContaBancaria;

public class Principal {

      public static void main(String[] args) {
            Conta conta = new Conta();
            for(int i = 0; i<6; i++) {
                  MinhaThread threadR = new MinhaThread("@"+i, conta);
                  Thread tr = new Thread(threadR);
                  tr.start();
            }
      }

}
```

```java
package ContaBancaria;

import java.util.Random;


public class MinhaThread implements Runnable {
	private String nome;
	private Conta conta;

	public MinhaThread(String nome, Conta conta) {
		this.nome = nome;
		this.conta = conta;
	}

	public void run() {
	  Random random = new Random();
		conta.depositar(random.nextInt(11));
		conta.retirar(random.nextInt(11));

	}
}
```

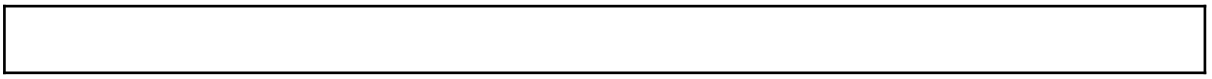
```java
package ContaBancariaAtomica;

import java.util.concurrent.atomic.AtomicInteger;

public class Conta {
	private AtomicInteger saldo = new AtomicInteger(0);

	public void depositar (int saldo) {

			int valor = this.saldo.addAndGet(saldo);
			System.out.println("Foi depositado "+ saldo);
			System.out.println("Saldo atual pós deposito igual a " +
valor);

	}

	public void retirar (int saldo) {
			if(this.saldo.get() - saldo >= 0) {
				int valor = this.saldo.addAndGet(saldo*-1);
				System.out.println("Foi retirado "+ saldo);
				System.out.println("Saldo atual pós retirada igual a
" + valor);
			}

	}
}
```

3) Um semáforo binário é definido tradicionalmente pelas operações "p" e "v". A primeira adquire um lock, fazendo o seguinte: testa se o lock é verdadeiro, se sim, adquire o lock tornando-o falso, senão espera. A segunda operação libera o lock, tornando-o verdadeiro. Defina um semáforo em Haskell usando STM. Para isso, defina um novo tipo com uma variável transacional do tipo Bool. Defina as duas funções "p" e "v".

4) Implemente a classe Semáforo em Java com dois métodos que realizam as seguintes operações:
up: incrementa o contador do semáforo e acorda threads que estejam bloqueadas.
down: decrementa o contador do semáforo ou, caso seja zero, suspende a thread.

```
class Semaforo extends Object
{
     private int contador;

     public Semaforo (int inicial)
     {
     contador = inicial;
     }

     public void down()
     {
     synchronized (this) {...}
     }
```

```
    public void up()
    {
    synchronized (this) {...}
    }
}
```



```
class Semaforo extends Object
{
    private int contador;

    public Semaforo (int inicial)
    {
        contador = inicial;
    }

    public void up()
    {
        synchronized (this) {
            this.contador++;
            notifyAll();
        }
    }

    public void down()
    {
        synchronized (this) {
            if(contador == 0){
                interrupt();
            } else {
                this.contador--;
            }
        }
    }
}
```

5) Na preparação de sanduíches em uma lanchonete chamada Pé de Fava, uma pessoa fornece os ingredientes (pão, carne e tomate); duas outras são responsáveis por preparar os sanduíches. Porém, a lanchonete dispõe de apenas uma faca para ser utilizada na preparação. Considere que os recipientes de ingredientes são

continuamente reabastecidos na capacidade máxima de porções (30 para cada ingrediente). Desenvolva uma solução me Haskell que modele o funcionamento desta lanchonete. Utilize memória transacional e variáveis mutáveis.

// Botei uns comentários, tá bem explicado:
https://github.com/vss-2/IP-LC/blob/master/Lanchonete.hs

1. Implemente, em Java, um vetor que seja seguro para uso com threads. Os métodos
get, set e swap devem ser implementados.

```
class SecureVector{
   int arranjo[] = null;
   public SecureVector(int i){
       this.arranjo = new int[i];
   }
   private int get(int i){
       synchronized (this){
           return this.arranjo[i];
       }
   }
   private void set(int i, int j){
       synchronized (this){
           this.arranjo[i] = j;
       }
   }
   private void swap(int i, int j){
       synchronized (this){
           int k = this.arranjo[i];
           this.arranjo[i] = this.arranjo[j];
           this.arranjo[j] = k;
       }
   }
}
```

2. Implemente uma fila bloqueante em Java. Não usar funções da biblioteca de Java.

```
public SecureQueue(int i){
    this.arranjo = new int[i+1];
```

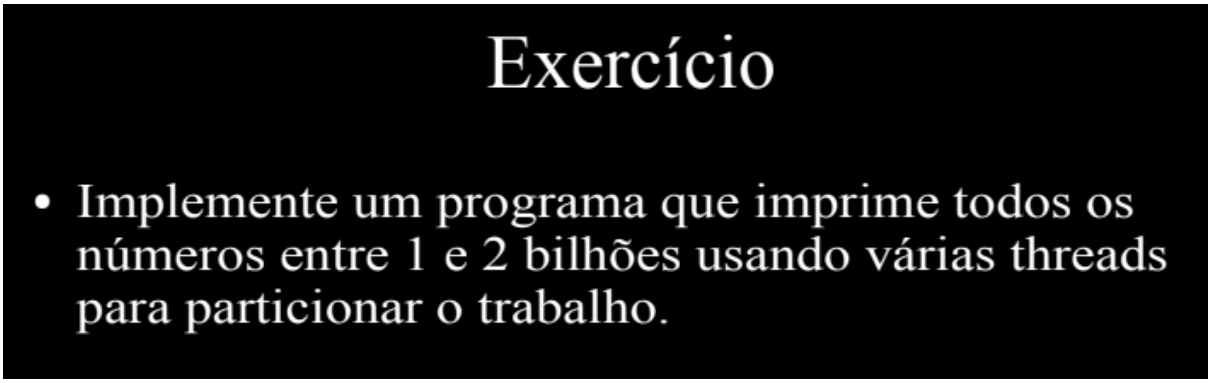
```
    maxSize = i;
  }
  synchronized public int take(){
    int retorn = 0;
    if(size == 0){
      try{
        wait();
      } catch (InterruptedException e) {}
    }
    retorn = this.arranjo[0];
    for(int i = 0;i<=maxSize;i++){
      this.arranjo[i] = this.arranjo[i+1];
    }
    size--;
    notifyAll();
    return retorn;
  }
  synchronized public void put(int i){
    if(size == maxSize){
      try{
        wait();
      } catch (InterruptedException e) {}
    }
    this.arranjo[size] = i;
    size++;
    notifyAll();
  }
}
```

3. O forno de uma padaria tem capacidade para assar 50 pães simultaneamente. A medida que pães ficam prontos, são retirados do forno. O abastecimento, que acontece apenas após o forno ser completamente esvaziado, é feito de maneira que 10 pães são colocados no forno por vez, até a capacidade do forno. Assuma que o primeiro lote de pães colocados no forno é também o primeiro a ser retirado. Utilizando uma fila bloqueante, implemente, em Java, o comportamento dessa padaria. Considere que retirar os pães do forno é um processo mais lento que o abastecimento dele, devido ao tempo necessário para assar os pães.

4. Utilizando memória transacional, implemente um buffer em Haskell. Defina o tipo

Buffer que pode guardar valores de um tipo qualquer, uma função para cria ̧c ̃ao de
um buffer (tipo: IO (Buffer a)) (utilize a fun ̧c ̃ao newTVarIO), a fun ̧c ̃ao put (tipo:
Buffer a −> a −> STM()) e a fun ̧c ̃ao get (tipo: Buffer a −> STM a). A fun ̧c ̃ao
put sempre coloca um novo dado no buffer; a fun ̧c ̃ao get retorna um valor do buffer,
por ́em aguarda se o buffer estiver vazio.

newBuffe r : : [a] −> IO (B uff e r a)
newBuffe r = newTVarIO




```
public class Exercicio {
  public static void main(String[] args){
     SharedNumber sNumber = new SharedNumber();
     Thread a = new Thread(new PrintSNumber("batato",sNumber));
     Thread b = new Thread(new PrintSNumber("cenoro",sNumber));
     a.start();
     b.start();

     System.out.println("Main Done!");
  }
}
class SharedNumber{
  public int number = 0;

  public SharedNumber(){
     this.number = 0;
  }
  public void add(){
     this.number+=1;
```

```
  }
}
class PrintSNumber implements Runnable{
  public SharedNumber number = null;
  String name;
  public PrintSNumber(String name,SharedNumber aux){
     this.number = aux;
     this.name = name;
  }

  public void run(){
     while (this.number.number <= 2000){
        System.out.println(name+" Have printed the number: "+this.number.number+".");
        this.number.add();
     }
  }
}
```

```
public class PrintNumber {

	public static void main(String[] args) {
		int x = 1;
		for(int j=0; j<50; j++) {
			PrintNumberThread threadR = new
PrintNumberThread(""+j,x,x+40000000);
			Thread tr = new Thread(threadR);
			tr.start();
			x+=40000000;
		}

	}

}

public class PrintNumberThread implements Runnable {
	private String nome;
	private int numberi;
	private int numberf;


	public PrintNumberThread(String nome, int numberi, int numberf) {
		this.nome = nome;
		this.numberi = numberi;
		this.numberf = numberf;
	}
```

```
        public void run() {
                for(int i = this.numberi; i<this.numberf; i++) {
                        System.out.println("Thread "+this.nome+" printando numero "+i);
                }
                System.out.println("Finalizado, thread de número "+this.nome);
        }
}
```

Exercício

- Implemente um programa que calcula todos os números primos entre 1 e um valor N fornecido como argumento. Seu programa deve dividir o trabalho a ser realizado entre X threads (onde X também é uma entrada do programa) para tentar realizar o trabalho de maneira mais rápida que uma versão puramente sequencial.
- A thread principal do programa deve imprimir os números primos identificados apenas quando as outras threads terminarem.

```
import java.util.Scanner;
public class Exercício2 {
   public static void main(String[] args) throws InterruptedException{
       Scanner leia = new Scanner(System.in);
       System.out.println("Informe em um unicco numero o range de procura de
primos.");
       Crivo arranjo = new Crivo(leia.nextInt()+1);
       Thread Crivo = new Thread(new VisitCrivo(arranjo));
       Crivo.start();
       try{
           Crivo.join();
       } catch(InterruptedException ie){}
   }
}
class Crivo{
```

```
    public boolean[] Crivo = null;
    public Crivo(int i){
        this.Crivo = new boolean[i];
    }
    public void setTrue(int i){
        this.Crivo[i] = true;
    }
    public void setFalse(int i){
        this.Crivo[i] = false;
    }
}
class VisitCrivo implements Runnable{
    public Crivo table = null;
    public VisitCrivo(Crivo table){
        this.table = table;
    }
    public void run(){
        int printAux = 0;
        for(int i = 2; i<this.table.Crivo.length;i++){
            if(!this.table.Crivo[i]){
                printAux++;
                System.out.print(i+", ");
                if((printAux)%10 == 0){ //Só pra printar de 10 em 10
                    System.out.println();
                }
                (new Thread(new MarkCrivo(this.table,i))).start();
            }
        }
    }
}
class MarkCrivo implements Runnable{
    public Crivo table = null;
    public int multOf = 0;
    public MarkCrivo(Crivo table,int i){
        this.table = table;
        this.multOf = i;
    }
    public void run(){
        for(int i = this.multOf*2;i<this.table.Crivo.length;i+=this.multOf){
            this.table.setTrue(i);
        }
    }
}
```

Exercícios

- Construa um programa onde *N* threads incrementam contadores locais (não-compartilhados) em um laço. Cada thread imprime o valor de seu contador a cada incremento. As threads param quando seus contadores chegam a um valor limite *X*, recebido como entrada pelo programa.
  - Por que as saídas das threads se misturam?
- Agora mude seu programa para que as threads modifiquem um contador global compartilhado entre elas.
  - O que acontece com os resultados?

```java
import java.util.Scanner;

public class Exercício3 {
   public static void main(String[] args){
       Scanner leia = new Scanner(System.in);
       System.out.println("Informe o valor limite dos contadores.");
       int x = leia.nextInt();
       Thread batata = new Thread(new Incrementar(x,"batata"));
       Thread cenoura = new Thread(new Incrementar(x,"cenoura"));
       Thread chuchu = new Thread (new Incrementar(x,"chuchu"));
       batata.start();
       cenoura.start();
       chuchu.start();
       try{
           batata.join();
           cenoura.join();
           chuchu.join();
       }catch (InterruptedException ie){
           System.out.println("Deu pau");
       }
       System.out.println("Acabou a salada");
   }
}
class Incrementar implements Runnable{
   public int X = 0;
   public String nome = null;
   public Incrementar(int i,String nome){
       this.X = i;
       this.nome = nome;
   }
   public void run(){
       for(int i = 0; i<=this.X;i++){
```

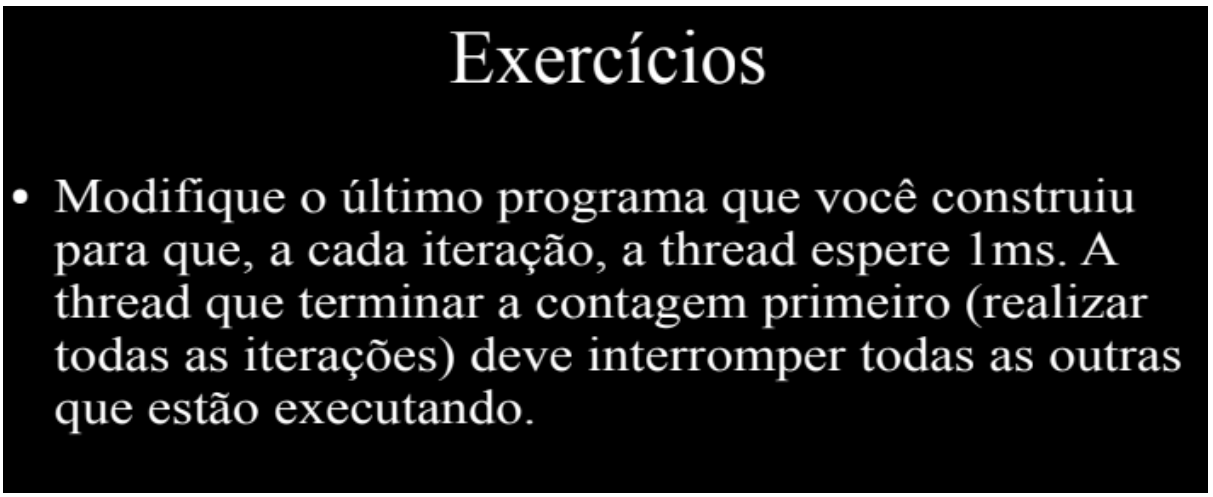
```
            System.out.println("Thread "+this.nome+" has printed: "+i);
        }
    }
}
```

Thread é não determinístico, não há garantia de qual está sendo executada, o processador escalona as threads.

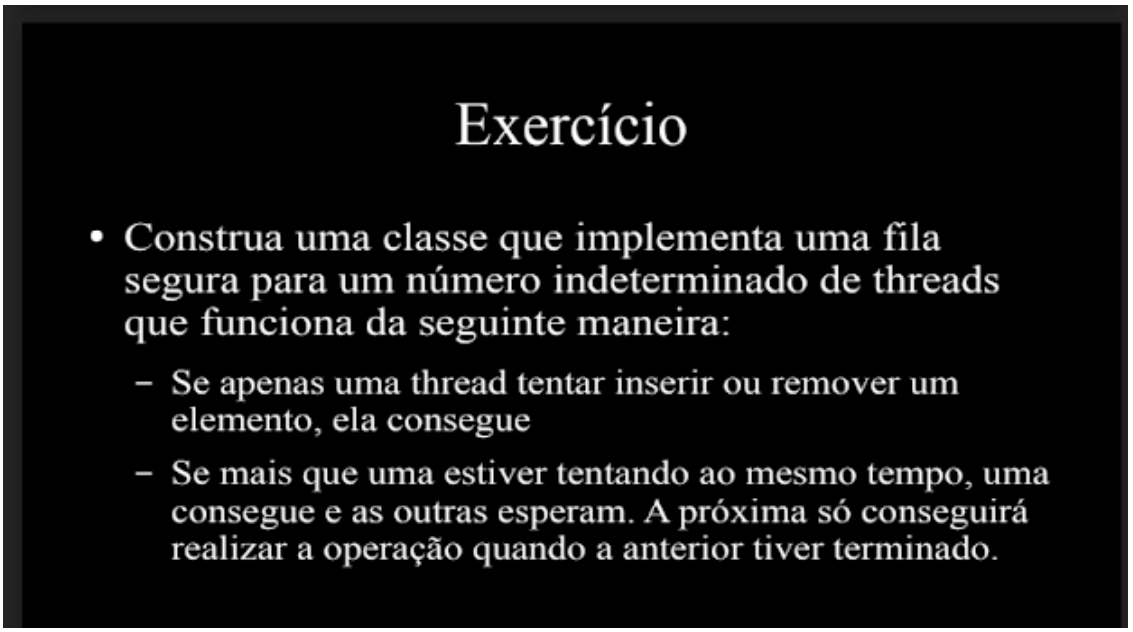
```
import java.util.Scanner;

public class Exercício4 {
    public static void main(String[] args){
        Scanner leia = new Scanner(System.in);
        System.out.println("Informe o valor limite dos contadores.");
        int x = leia.nextInt();
        SharedInt si = new SharedInt();
        Thread batata = new Thread(new IncrementarPub(x,"batata",si));
        Thread cenoura = new Thread(new IncrementarPub(x,"cenoura",si));
        Thread chuchu = new Thread (new IncrementarPub(x,"chuchu",si));
        batata.start();
        cenoura.start();
        chuchu.start();
        try{
            batata.join();
            cenoura.join();
            chuchu.join();
        }catch (InterruptedException ie){
            System.out.println("Deu pau");
        }
        System.out.println("Acabou a salada");
    }
}
class SharedInt{
    public int x = 0;
    public SharedInt(){
        this.x = 0;
    }
    public void plus(){
        this.x++;
    }
}
class IncrementarPub implements Runnable{
    public SharedInt sInt= null;
    public int X = 0;
    public String nome = null;
    public IncrementarPub(int i,String nome, SharedInt si){
        this.sInt = si;
        this.X = i;
        this.nome = nome;
```

```
    }
    public void run(){
        for(; this.sInt.x <=this.X;){
            System.out.println("Thread "+this.nome+" has printed: "+this.sInt.x);
            this.sInt.plus();
        }
    }
}
```

Digamos que a ordem ficou zoada, e alguns números se repetem.



Em duvida como fazer, pq o método interrupt() só é declarado no main



```java
package Filas;
import PrintNumberExercicio.PrintNumberThread;

public class Principal {

	public static void main(String[] args) {

		Fila fila = new Fila ();
		for(int i = 0; i<30 ;i++) {
			MinhaThread threadR = new MinhaThread("@"+i,fila);
			Thread tr = new Thread(threadR);
			tr.start();
		}
		System.out.println("Fim");
	}
}
```

```java
package Filas;

public class Fila {

	private int valor = 0;
	private Fila proximo = null;

	public Fila () {
		this.valor = 0;
		this.proximo = null;
	}

	public synchronized void Inserir (int valor) {
		if(this.valor == 0) {
			this.valor = valor;
			this.proximo = new Fila ();
		} else if (this.valor != 0) {
			this.proximo.Inserir(valor);
		}
	}

	public synchronized void Remover() {
		if(this.valor!=0) {
			if(this.proximo.valor == 0) {
				this.valor = 0;
			} else {
				this.proximo.Remover();
			}
		}else {
			System.out.println("Fila vazia");
		}
	}
```

```
}
```

```
package Filas;

public class MinhaThread implements Runnable {
	private String nome;
	private static Fila fila;



	public MinhaThread(String nome, Fila fila) {
		this.nome = nome;
		this.fila = fila;
	}

	public void run() {
		fila.Inserir(1);
		System.out.println("Thread "+this.nome+" inseriu!");
		fila.Inserir(2);
		System.out.println("Thread "+this.nome+" inseriu!");
		fila.Inserir(1);
		System.out.println("Thread "+this.nome+" inseriu!");
		fila.Remover();
		System.out.println("Thread "+this.nome+" removeu!");
	}
}
```

PROVA 2017.1

1. Leia a descrição abaixo e, em seguida, faça o que se pede.

Uma fábrica de lâmpadas utiliza máquinas de fabricação distintas que produzem bulbos, soquetes e embalagens que são colocados em caixas separadas. Após a produção destes elementos, duas máquinas produzem lâmpadas, juntando bulbos e soquetes e colocando cada lâmpada em uma embalagem. Cada lâmpada embalada é colocada em uma única caixa até que seja completamente preenchida com 50 lâmpadas. Quando preenchida, a caixa é transportada para um depósito, sendo substituída por uma caixa vazia.
(a) Implemente, em Haskell, o que foi descrito acima, utilizando mutable variables (MVar). envolvendo tipos primitivos atômicos).

2. Implemente em Java as classes Produtor, Consumidor e ProdutorConsumidor. Esta última possui o método main(). O dados produzidos são valores do tipo inteiro. Não se pode utilizar classes da API de Java como, por exemplo, interface BlockingQueue, que possui métodos put e take, utilizando a implementação ArrayBlockingQueue.

3. Modifique a questão anterior para utilizar tipos primitivo atômicos ou a interface BlockingQueue.

Se você fez a de cima, essa 3 é só você trocar todos os lugares que chama ArrayBlockingQueue por simplesmente BlockingQueue, e usufruir dos métodos put e take no lugar da sua implementação.

4. Defina um tipo chamado conta Conta constituído de uma variável transacional do tipo inteiro. Defina as seguintes funções em Haskell:
(a) saque :: Conta− > Int− > STM() que realiza retirada de uma quantia de uma conta
(b) deposito :: Conta− > Int− > STM() que realiza um depósito em uma conta. Utilize a função saque para definir deposito
(c) saque2, uma modificação da função saque, que bloqueia, caso o saldo da conta vá se tornar negativo.
(d) Suponha que você possa retirar dinheiro de uma conta A se esta tiver saldo suficiente, senão, retira de uma conta B. Defina a função saque3, utilizando a função orElse e a função saque2.

```haskell
module Main where
import Control.Concurrent
import Control.Concurrent.MVar
import Control.Concurrent.STM

type Conta = TVar Int
```

```haskell
saque :: Conta -> Int -> STM()
saque a b = do
 x <- readTVar a
 writeTVar a (x - b)

deposito::Conta-> Int ->STM()
deposito a b = do
 saque a ((-) 0 b)

saque2:: Conta -> Int -> STM()
saque2 a b = do
 x <- readTVar a
 if(x - b >= 0)
 then do { writeTVar a (x-b)}
 else return ()

main :: IO()
main = do
 putStrLn("aa")
 x <- atomically(newTVar 10)
 atomically(deposito x 10)
 atomically(saque2 x 21)
 z <- atomically(readTVar x)
 print z
 return()
```

Lista 2
Aula de Hoje

```lua
function f1 (a)
	print("f1", a)
	coroutine.yield(2*a)
	f2(a)
	return a*7
end

function f2 (a)
	print("f2", a)
	return coroutine.yield(3*a)
end

co1 = coroutine.create(function (a,b)
	print("co1", a, b)
	local r = f1(a+1)
	print(r)
	print(a,b)
	local r, s = coroutine.yield(a+b, a-b)
	print("co1", coroutine.resume(co2, a+1,b+1))
	return b, "fim co1"
end)

co2 = coroutine.create(function (a,b)
	print("co2", a, b)
	print("co2", coroutine.resume(co3, a+1, b+1))
	return b, "fim co2"
end)

co3 = coroutine.create(function (a,b)
	print("co3", a, b)
	coroutine.yield(b*57)
	local r, s = coroutine.yield(a+b, a-b)
	print("co3", r, s)
	print("co3", a+1, b+1)
	end)

print("main", coroutine.resume(co1, 6, 14))
print("main", coroutine.resume(co1))
print("main", coroutine.resume(co1, "x", "y"))
print("main", coroutine.resume(co1, "x", "y"))
```

Passos:
Obs: esses “true” são printados por Lua como se fosse um aviso dizendo que deu certo.

1. Executa: `print("main", coroutine.resume(co1, 6, 14))` mas não printa, vai pra co1
2. Dentro de co1, printa `co1 6 14`
3. Ainda em co1, muda o contexto para f1(6+1) em, salva em r: `local r = f1(a+1)`
4. Em f1, printa `f1 7` encontra o primeiro yield
5. Esse yield faz (7*2) e retorna para o resume mais recente que ele rodou: o do passo 1
6. Volta para a linha do passo 1, dessa vez printando: `main true 14` vai para linha debaixo
7. Ele está nessa linha `print("main", coroutine.resume(co1))` esse resume te leva de volta ao último yield, que é o `coroutine.yield(2*a)`
8. O código voltou para f1, executa a linha embaixo de do yield `f2(a)`: o contexto vai pra f2
9. Chegando em f2 `print("f2", a)` printa `f2 7` é chamado um yield: `coroutine.yield(3*a)`
10. Como de costume, o yield chama o último resume executado, que é o do passo 7.
11. Será printado `main true 21` vai para linha debaixo
12. Ele vê esse resume `print("main", coroutine.resume(co1, "x", "y"))` volta para o último yield encontrado.
13. Ele voltou pro f2, está no `end`
14. Ele volta pro f1, e roda `return a*7` é retornado 49, depois `end` voltando para o passo 3
15. O `print(r)` produz `49` , `print(a,b)` o valor de a e b é o mesmo que no passo 2 `6 14`
16. É encontrado um yield em `local r, s = coroutine.yield(a+b, a-b)` Lua busca o último resume executado é o do passo 12, (a+b) = 20, (a-b) = -8; isso será printado na linha do passo 12, note que x e y serão substituídos por 20 e -8. `main true 20 -8`
17. Vamos para o último `print("main", coroutine.resume(co1, "x", "y"))` Lua busca o último yield, no caso, ele volta para a linha debaixo do passo 16.
18. É executado `print("co1", coroutine.resume(co2, a+1,b+1))` os valores: (co2, 7,15)
19. Em co2, é printado `co2 7 15`.
20. Como não há yields para retornar, executamos co3 de: `print("co2", coroutine.resume(co3, a+1, b+1))`.
21. Em co3, é printado `co3 8 16`. Encontramos o yield `coroutine.yield(b*57)`, voltamos para o último resume, levando o retorno 16*57 = 912
22. Voltamos para resume do passo 18: é printado o status e o retorno `co2 true 912`
23. É printado `co2 true 912`. co2 retorna: `b`, que era `15` (passo 19), e `fim co2`
24. O co2 que havia sido chamado por co1 no passo 18, leva seu retorno, então é printado:
25. É printado `co1 true 15 fim co2` no retorno leva `b = 14` e `fim co1` chegou no `end`
26. Volta para o último print, printando o valor de b em co1 (não mudou desde o passo 2), que é `main true 14 fim co1`

PROVA 2019.1

t

Centro de Informática - UFPE
Segundo Exercício Escolar — IF686 (2019.1)
Data: 13/06/2019
Docente: Márcio Lopes Cornélio
Discente:

Turma: 15
Horário: 8h-10h

Instruções para entrega: Enviar os seguintes arquivos para *mlc2@cin.ufpe.br* : (1) arquivos .hs e .java, e (2) um arquivo no formato pdf com **todas** as respostas, incluindo aquelas que não envolvem código apenas. Escrever o nome na primeira linha dos arquivos. O assunto do email deve estar no formato: Nome completo_2ee_plc_2019_1.

(1,5) 1. (**Java**) Considere a classe abaixo

```
public class C extends Thread{
 private Object a;
 private Object b;

 public C(Object x, Object y){
  a = x;
  b = y;
 }

 public void run(){
  synchronized(b){
   synchronized(a){   ... } } }
}
```

Considere também o seguinte código, em que a1 e b1 são do tipo Object.

```
C t1 = new C(a1, b1);
C t2 = new C(a1, b1);
t1.start();
t2.start();
```

Há possibilidade de haver *deadlock* ao executarmos o código acima? Justifique. Se sim, explique em detalhes como pode acontecer, descrevendo uma execução intercalada problemática e os *locks* utilizados.

(3.0) 2. (**Java**) Implemente em Java um produtor-consumidor de valores inteiros.

(3.0) 3. (**Haskell**) Na preparação de sanduíches em uma lanchonete, uma pessoa fornece os ingredientes (pão, carne e tomate); duas outras são responsáveis por preparar os sanduíches. Porém, a lanchonete dispõe de apenas uma faca para ser utilizada na preparação. Considere que os recipientes de ingredientes são continuamente reabastecidos na capacidade máxima de porções (30 para cada ingrediente). Desenvolva uma solução em Haskell que modele o funcionamento desta lanchonete.

(2.5) 4. (**Lua**) Escreva uma função que recebe os coeficientes de um polinômio e retorna uma função que, quando chamada com um valor para x, retorna o valor do polinômio para aquele x. Por exemplo,

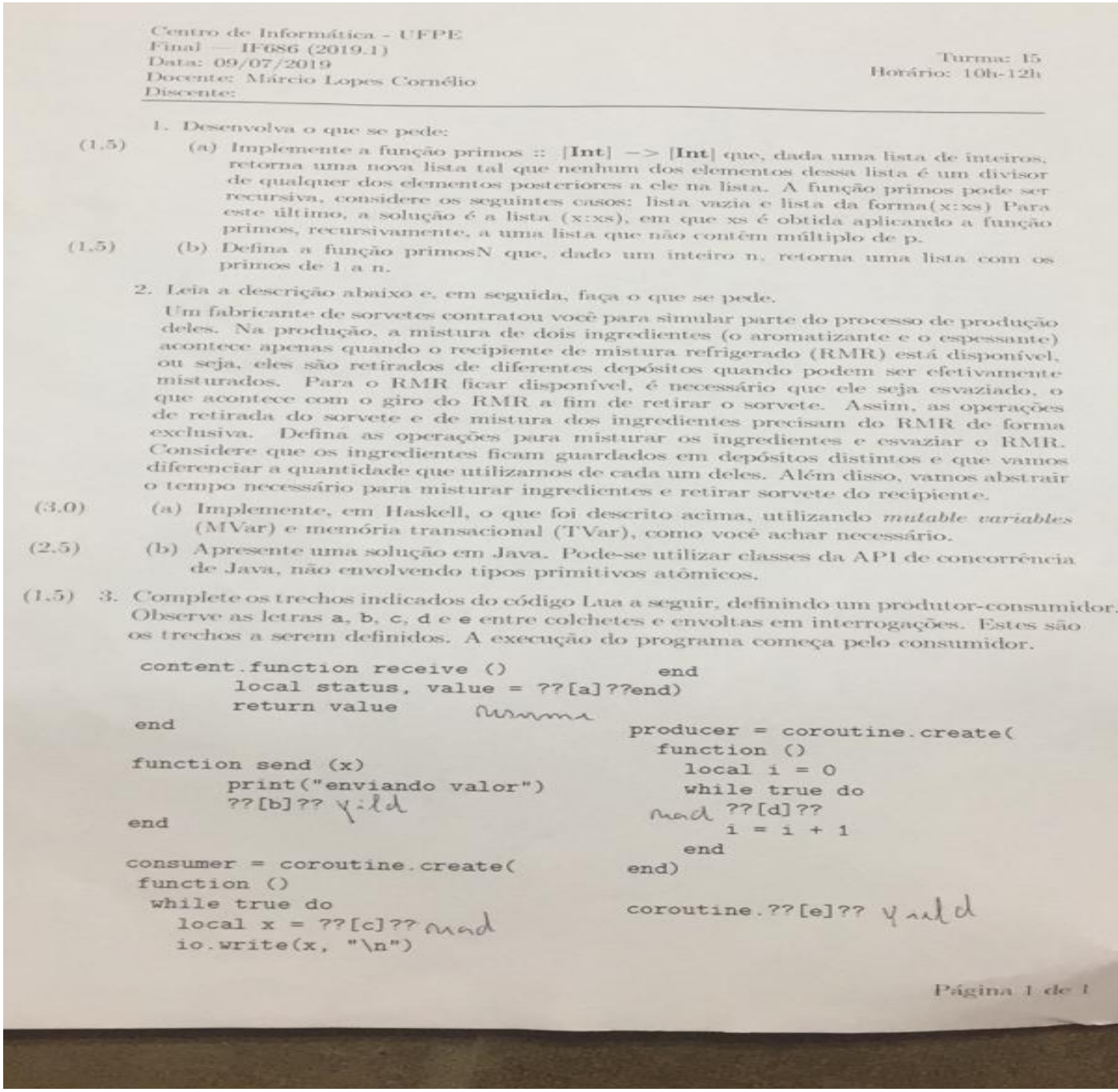
```
f = newpoly({3,0,1})
    (f(0))   - - > 1
    (f(5))   - - > 76
    (f(10))  - - > 301
```

1. Não, as variáveis chamadas “a” e “b” não estão sendo chamadas por métodos distintos de forma que a execução de ambas cause um deadlock, do jeito que está, b será “lockado” e depois “a” será lockado, sem nenhuma execução de outra thread, podemos afirmar que o código é safe.

4.

```
function poli (a,b,c)
    return function (x)
        print(a*x*x + b*x + c)
    end
end

f = poli(3,0,1)
print(f(5))
```

FINAL 2019.1



Centro de Informática - UFPE
Final — IF686 (2019.1)
Data: 09/07/2019
Docente: Márcio Lopes Cornélio
Discente:

Turma: 15
Horário: 10h-12h

1. Desenvolva o que se pede:

(1.5) (a) Implemente a função primos :: [Int] -> [Int] que, dada uma lista de inteiros, retorna uma nova lista tal que nenhum dos elementos dessa lista é um divisor de qualquer dos elementos posteriores a ele na lista. A função primos pode ser recursiva, considere os seguintes casos: lista vazia e lista da forma(x:xs) Para este último, a solução é a lista (x:xs), em que xs é obtida aplicando a função primos, recursivamente, a uma lista que não contém múltiplo de p.

(1.5) (b) Defina a função primosN que, dado um inteiro n, retorna uma lista com os primos de 1 a n.

2. Leia a descrição abaixo e, em seguida, faça o que se pede.

Um fabricante de sorvetes contratou você para simular parte do processo de produção deles. Na produção, a mistura de dois ingredientes (o aromatizante e o espessante) acontece apenas quando o recipiente de mistura refrigerado (RMR) está disponível, ou seja, eles são retirados de diferentes depósitos quando podem ser efetivamente misturados. Para o RMR ficar disponível, é necessário que ele seja esvaziado, o que acontece com o giro do RMR a fim de retirar o sorvete. Assim, as operações de retirada do sorvete e de mistura dos ingredientes precisam do RMR de forma exclusiva. Defina as operações para misturar os ingredientes e esvaziar o RMR. Considere que os ingredientes ficam guardados em depósitos distintos e que vamos diferenciar a quantidade que utilizamos de cada um deles. Além disso, vamos abstrair o tempo necessário para misturar ingredientes e retirar sorvete do recipiente.

(3.0) (a) Implemente, em Haskell, o que foi descrito acima, utilizando *mutable variables* (MVar) e memória transacional (TVar), como você achar necessário.

(2.5) (b) Apresente uma solução em Java. Pode-se utilizar classes da API de concorrência de Java, não envolvendo tipos primitivos atômicos.

(1.5) 3. Complete os trechos indicados do código Lua a seguir, definindo um produtor-consumidor. Observe as letras a, b, c, d e e entre colchetes e envoltas em interrogações. Estes são os trechos a serem definidos. A execução do programa começa pelo consumidor.

```
content.function receive ()
      local status, value = ??[a]??
      return value
end

function send (x)
      print("enviando valor")
      ??[b]??
end

consumer = coroutine.create(
 function ()
  while true do
    local x = ??[c]??
    io.write(x, "\n")
  end
end)

producer = coroutine.create(
  function ()
    local i = 0
    while true do
      ??[d]??
      i = i + 1
    end
end)

coroutine.??[e]??
```

Página 1 de 1

1)

```
primos [] = []
primos (x:xs) =  x : primos (retiraMultiplos x xs)

retiraMultiplos a [] = []
```

```haskell
retiraMultiplos a (x:xs) =  if mod x a == 0 then retiraMultiplos a  xs else  x
: retiraMultiplos a xs

--caso o professor fale para nao usar compreensao de lista
geraLista 0 _ = []
geraLista n m = m : geraLista (n-1) (m+1)

primosN 0 = []
primosN n = primos (geraLista n 2)


--Utilizando compreensao de lista
geraLista2 n = [x | x <- [2..n]]


primosN2 n = primos (geraLista2 n)

--sem a funcao auxiliar
primosN3 n = primos [x | x <- [2..n]]
```

```haskell
takeRep:: [Int] -> [Int]
takeRep[] = []
takeRep(x:xs) = (x: (takeRep(filter (/=x) xs)))


primos::[Int]->[Int]
primos [] = []
primos (x:xs) = takeRep(x : (filter(\n -> not (mod n x == 0)) (xs)) ++
primos (filter(\n -> not (mod n x == 0)) (xs)))
```

3)

```lua
consumer = coroutine.create(function()
    while true do

        local x = receive() -- recebe do produtor
        io.write(x, "\n") -- consome novo valor
```

```
    end
end)

function receive ()
    local status, value = coroutine.resume(producer)
    return value
end
producer = coroutine.create(function()
    local i = 0
    while true do
    local x = io.read() -- produz novo valor
    send(x) -- envia para consumidor
    end
end)

function send (x)
    print("enviando valor")
    coroutine.yield(x)
end

coroutine.resume(consumer)
```

Desafio de fazer um contador usando apenas uma funcionalidade de Thread

1 - synchronized
2 - reentrant lock
3 - notify
4 - atomic
5 - blockingQueue

2)

```
package Desafio;

public class Principal {

	public static void main(String[] args) {
		Contador cont = new Contador();
		for(int i=0; i<5;i++) {
			MinhaThread threadR = new MinhaThread("@Lucas", cont);
```

```
                    Thread tr = new Thread(threadR);
                    tr.start();
              }

       }

}
```

```
package Desafio;

public class MinhaThread implements Runnable {
       private String nome;
       private Contador contador;

       public MinhaThread(String nome, Contador contador) {
              this.nome = nome;
              this.contador = contador;

       }

       public void run() {
              System.out.println("Startando uma thread");
         for(int i =0; i<10; i++) {
              this.contador.IncrementContador();
         }
       }
}
```

```
package Desafio;

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

public class Contador {
       private int valor;
       private Lock l;

       public Contador() {
              this.valor = 0;
              this.l = new ReentrantLock();
       }

       public void IncrementContador() {
              boolean permitir = l.tryLock();
              try {
                     while(!permitir) {
                            permitir = this.l.tryLock();
                     }
              } finally {
```

```
                this.valor++;
                System.out.println("Contador: "+this.valor);
                l.unlock();

            }

        }

}
```

1. Usando Synchronized de bloco ou Synchronized

```
import java.util.concurrent.*;
public class Contador {
   private int valor;

   public Contador(int inicio){
       this.valor = inicio;
   }


   // Jeito 1
   public synchronized void incrementarBloco(){
       this.valor++;
       System.out.print(Thread.currentThread());
       System.out.printf(" incrementou para: %d\n", this.valor);
       return;
   }

   // Jeito 2
   public void incrementarThis(){
       synchronized(this){
           this.valor++;
           System.out.print(Thread.currentThread());
           System.out.printf(" incrementou para: %d\n", this.valor);
       }
       return;
   }
```

```
}

public class Auxiliar implements Runnable {
   private Contador c;

   public Auxiliar(){
       this.c = new Contador(0);
   }

   public void run() {
       for(int f = 0; f < 5; f++){
           c.incrementarBloco();
       }
   }
}
class Main {
   public static void main(String args[]){
       Thread t1 = new Thread(new Auxiliar());
       Thread t2 = new Thread(new Auxiliar());
       Thread t3 = new Thread(new Auxiliar());
       t1.start(); t2.start(); t3.start();
   }
}
```

4) AtomicInteger

```
import java.util.concurrent.atomic.AtomicInteger;

class Atomico{
   private AtomicInteger contador;
   public Atomico(){
       this.contador = new AtomicInteger(0);
   }

   public void IncrementContador(){
       contador.getAndIncrement();
   }
   public void DecrementContador(){
       contador.getAndDecrement();
   }
```

```
}

class MinhaThreadd implements Runnable{
   private int op;
   private Atomico a;
   public MinhaThreadd(int op){
       this.op = op;
       this.a = new Atomico();
   }
   public void run(){
       if(op == 0){
           a.IncrementContador();
           //System.out.println(this.op);
       }else{
           a.DecrementContador();
           //System.out.println(this.op);
       }
   }
}

public class Desafio4 {
   public static void main(String[] args) {
       for(int i = 0; i < 100; i++){
           Thread t = new Thread(new MinhaThreadd((int) (Math.random() *
2)));
           t.start();
       }
   }
}
```

PROVA 2019.2 2EE

Centro de Informática - UFPE
Segugimdo Exercício Escolar — IF686 (2019.2)
Data: 21/11/2019
Docente: Márcio Lopes Cornélio
Discente:

Turma: I5
Horário: 15h-17h

Instruções para entrega: Enviar arquivos para *mlc2@cin.ufpe.br* : (1) arquivos com os fontes dos programas e (2) um arquivo no formato pdf também com todas as respostas. Escrever o nome na primeira linha dos arquivos. O assunto do email deve estar no formato: Nome completo_2ee_plc_2019_2.

(2,5) 1. Implemente em Java um produtor-consumidor de valores inteiros.

(2,5) 2. Implemente uma fila bloqueante em Java. Não usar funções da biblioteca de Java.

(2,5) 3. Leia a descrição abaixo e, em seguida, faça o que se pede.

Uma fábrica de porcas e parafusos funciona com o uso de uma máquina de fabricação de porcas e de uma outra máquina que fabrica parafusos. Após a produção, porcas e parafusos são colocados em caixas separadas de onde são retirados por duas máquinas que montam pares porca-parafuso. Cada par porca-parafuso é colocado em uma única caixa. Implemente, em Haskell, o que foi descrito. As máquinas são definidas por funções executadas em threads.

(1,5) 4. Escreva uma função que recebe os coeficientes de um polinômio e retorna uma função que, quando chamada com um valor para x, retorna o valor do polinômio para aquele x. Por exemplo,

```
f = newpoly({3,0,1})
print(f(0))   -- -> 1
print(f(5))   -- -> 76
print(f(10))  -- -> 301
```

(1,0) 5. Complete os trechos indicados do código Lua a seguir, definindo um produtor-consumidor. Observe as letras 'a', 'b', 'c', 'd' e 'e' entre colchetes e envoltas em interrogações. Estes são os trechos a serem definidos.

```
coroutine.receive ()
    local status, value = ??[a]??
    return value
end

function send (x)
    print("enviando valor")
    ??[b]??
end

consumer = coroutine.create(
  function ()
    while true do
      local x = ??[c]??
      io.write(x, "\n")
    end
  end)

producer = coroutine.create(
  function ()
    local i = 0
    while true do
      ??[d]??
      i = i + 1
    end
  end)

coroutine.??[e]??
```