LISTA DE EXERCÍCIO THREAD

1. Assinale a alternativa que corresponde ao código correto para deixar uma thread dormir por 5 segundos.

A) Thread.sleep(5);

B) thread.wait(5000);

C) thread.sleep(5);

D) Thread.sleep(5000);

Resposta: Alternativa D.

2. Analise o código a seguir:

```
TarefaMultiplicacao tarefa = new TarefaMultiplicacao(...);

Thread threadMultiplicador = new Thread(tarefa);
```

Sobre a tarefa que a thread recebe, selecione a alternativa correta.

A) É um Runnable.

B) É uma classe qualquer.

C) A tarefa deve ter o método main.

D) É um Comparable.

Resposta: Alternativa A.

3. Em relação a um programa com várias Threads, marque a alternativa correta sobre a ordem de execução.

A) A ordem de execução será diferente em máquinas diferentes, mas na mesma máquina será sempre a mesma.

B) Não é possível determinar a ordem de execução, que pode ser sempre diferente inclusive na mesma máquina.

C) A ordem de execução será a mesma somente para máquinas com mesmo sistema operacional.

D) A ordem de execução pode ser determinada, mas isso depende da implementação da máquina virtual.

Resposta: Alternativa B.

4. Considere que um programa que crie e chame o método start() em três threads que respectivamente imprimem os valores 1, 2 e 3. Qual a ordem que esses valores serão impressos?

A) 3,2,1

B) Nada será impresso.

C) 1, 2, 3

D) Não é possível determinar a ordem.

Resposta: Alternativa D.

5. Assinale a alternativa que apresenta a maneira correta de pegar a instância da Thread atual (aquela que está sendo executada).

A) Thread atual = Object.currentThread();

B) Thread atual = Thread.getThread();

C) Thread atual = (Thread) this;

D) Thread atual = Thread.currentThread();

Resposta: Alternativa D.

6. Qual o nome do bloco ou modificador que deve ser colocado em um método para que não possa ser executado por duas Threads ao mesmo tempo? Selecione a alternativa correta.

A) static

B) strictfp

C) synchronized

D) final

Resposta: Alternativa C.

7. Assinale a alternativa que apresenta o significado de operação atômica.

A) Cuja execução não pode ser interrompida na metade.

B) Que está associada a apenas uma Thread

C) Cuja execução é feita na memória principal sem o uso de cache.

D) Cuja execução por várias Threads é alternada.
Resposta: Alternativa A.

8. assinale a alternativa que faz uso correto da palavra chave synchronized:

A) public void metodo() {
}

B) public void metodo() {
synchronized(this){
}
}

C) public class TarefaBuscaNome {
public synchronized TarefaBuscaNome() {
}
}

D) public class TarefaBuscaNome {
public synchronized String nome;
}

Resposta: Alternativa B.

9. Assinale a alternativa que representa a maneira correta de fazer com que uma thread A espere a execução da thread B.

A) Coloque um wait() na Thread B.

B) Coloque um wait() na Thread A e um notify() na Thread B.

C) Coloque o modificador synchronized na thread B.

D) Coloque um notify() na Thread A

Resposta: Alternativa B.

10. Crie um programa na linguagem Java para ler dois números e qual operação matemática deve ser utilizada. Em seguida o programa deverá apresentar o resultado do cálculo. Toda a operação matemática deverá ser executada via thread.

Classe OperacaoMatematica:

```
public class OperacaoMatematica implements Runnable {

	private Integer numA;
	private Integer numB;
	private String operacao;

	public OperacaoMatematica(Integer numA, Integer numB, String operacao) {
		this.numA = numA;
		this.numB = numB;
		this.operacao = operacao;

		Thread t = new Thread(this);
		t.start();
	}

	@Override
	public void run() {
	  // TODO Auto-generated method stub
	  switch(operacao) {
		case "+":
		  System.out.println(numA+" + "+numB+" = "+(numA+numB));
		  break;
		case "-":
		  System.out.println(numA+" - "+numB+" = "+(numA-numB));
		  break;
		case "/":
		  System.out.println(numA+" / "+numB+" = "+(numA/numB));
		  break;
		case "*":
		  System.out.println(numA+" * "+numB+" = "+(numA*numB));
		  break;
		default:
		   System.out.println("Operacao nao efetuada, verifique as entradas." );
		   break;
		}
	}

}
```

Classe principal (App):

```
import java.util.Scanner;

public class App {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Integer numA;
		Integer numB;
		String operacao;

		Scanner in = new Scanner(System.in);


		System.out.print("Digite o primeiro número: ");
		numA = Integer.valueOf( in.nextLine() );

		System.out.print("\n\nDigite o segundo número: ");
		numB = Integer.valueOf( in.nextLine() );

		System.out.print("\n\nDigite uma das 4 opecacoes basicas [+-/*]: ");
		operacao = in.nextLine();

		OperacaoMatematica o = new OperacaoMatematica(numA, numB, operacao);

		in.close();
	}

}
```

11. Implemente uma thread para contar a quantidade de consoantes e vogais de uma frase.

Classe ConsoantesVogais:

```
public class ConsoantesVogais implements Runnable {

	private boolean vogal = false;
	private String palavra;
	private Integer qtdVogais = 0;
	private Integer qtdConsoantes = 0;
	private char c;
	private Integer resultado = 0;

	public ConsoantesVogais(boolean vogal, String palavra) {
		this.vogal = vogal;
		this.palavra = palavra.toLowerCase();
	}

	@Override
	public void run() {
	  // TODO Auto-generated method stub
	  for(int x=0; x < palavra.length(); x++) {
	    c = palavra.charAt(x);
	    if( c == 97 || c == 101 || c == 105 || c == 111 || c == 117 ) {
	       qtdVogais++;
	    } else if ( (c >= 98) && (c <=122) ){
	       qtdConsoantes++;
	    }
	  } //fim do for
	  resultado = (vogal ? qtdVogais : qtdConsoantes);
	  if(vogal)
	    System.out.println("\nHa "+resultado+" vogais na palavra '"+palavra+"'.");
	  else
	    System.out.println("\nHa "+resultado+" consoantes na palavra
	                        '"+palavra+"'.");
	} //fim de run

	public void quantidade() {
		Thread t = new Thread(this);
		t.start();
	}

}
```

Classe principal (App):

```
import java.util.Scanner;

public class App {

   public static void main(String[] args) {
      // TODO Auto-generated method stub
      Scanner in = new Scanner( System.in );

      String palavra;

      System.out.print("Digite uma palavra: ");
      palavra = in.nextLine();

      ConsoantesVogais consoantes = new ConsoantesVogais(false, palavra);
      ConsoantesVogais vogais = new ConsoantesVogais(true, palavra);

      consoantes.quantidade();
      vogais.quantidade();

      in.close();
   }

}
```

12. Escreva um programa para ler um valor X e um valor Z (se Z for menor que X deve ser lido um novo valor para Z). Crie uma thread para contar quantos números inteiros devemos somar em sequência (a partir do X inclusive) para que a soma ultrapasse o valor de Z o mínimo possível. Escrever o valor final da contagem.

Exemplo:

X Z Reposta

3 20 5 (3+4+5+6+7=25)

2 10 4 (2+3+4+5=14)

30 40 2 (30+31=61)

Classe SomaDeNumeros:

```
public class SomaDeNumeros implements Runnable {

	private Integer x;
	private Integer z;
	private Integer soma = 0;

	public SomaDeNumeros(Integer x, Integer z) {
		this.x = x;
		this.z = z;
	}

	public void calcularSequencia() {
		Thread t = new Thread(this);
		t.start();
	}

	@Override
	public void run() {
	   // TODO Auto-generated method stub
	   Integer qtd=1;
	   Integer vlInicial = x;
	   soma = vlInicial;
	   while(soma < z) {
		vlInicial++;
		soma = soma + vlInicial;
		qtd++;
	   }
	   System.out.print("\n\nX\tZ\tResposta\n"+x+"\t" + z + "\t" + qtd + " (" + x);
	   for(int aux=(x+1); aux <=vlInicial; aux++) {
		System.out.print("+");
		System.out.print(aux);
	   }
	   System.out.print("=" + soma + ")\n");
	}

}
```

Classe principal (App):

```
import java.util.Scanner;

public class App {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner in = new Scanner(System.in);

		Integer x;
		Integer z;

		System.out.print("Digite um numero: ");
		x = Integer.parseInt( in.nextLine() );

		System.out.print("Digite um numero maior que o anterior: ");
		z = Integer.parseInt( in.nextLine() );

		while( z < x ) {
			System.out.print("\n\nO valor digitado nao e maior que "+
				x+", digite um novo valor: ");
			z = Integer.parseInt( in.nextLine() );
		}

		SomaDeNumeros soma = new SomaDeNumeros(x, z);

		soma.calcularSequencia();

		in.close();

	}

}
```

13. Uma agência bancária possui vários clientes, todavia a agência possui apenas um caixa eletrônico em funcionamento, para a realização de saques e transferências. Para realizar a operação de saque o cliente gasta 8 segundos para finalizar seu saque e para a transferência o cliente do banco gasta 5 segundos. Implemente um sistema em que o caixa eletrônico será o nosso objeto e os clientes serão threads que tentaram realizar as operações de saque e transferência no caixa eletrônico.

Classe Cliente:

```
public class Cliente implements Runnable {

	private String nome;
	private Boolean saque;
	private Boolean transferencia;

	private static CaixaEletronico cx = new CaixaEletronico();

	public Cliente(String nome) {
		super();
		this.nome = nome;
	}

	public void realizarOperacao(Boolean saque, Boolean transferencia) {
		this.saque = saque;
		this.transferencia = transferencia;

		new Thread(this, nome).start();
	}

	@Override
	public void run() {
		// TODO Auto-generated method stub
		System.out.println("Cliente " + nome + " solicitou uso do caixa.");

		cx.operacao(nome, saque, transferencia);

		System.out.println("Cliente " + nome + " finalizou o uso do caixa.");
	}

}
```

Classe CaixaEletronico:

```
public class CaixaEletronico {

	public synchronized void operacao(String cliente, Boolean saque, Boolean
transferencia) {

		if( saque ) {
			System.out.println("Cliente " + cliente +
				" iniciou operacao de saque!");
			try {
				Thread.sleep(8000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}

		if( transferencia ) {
			System.out.println("Cliente " + cliente +
				" iniciou operacao de transferencia!");
			try {
				Thread.sleep(5000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}

}
```

Classe principal (App):

```
public class App {

	public static void main(String[] args) {

		Cliente ana = new Cliente("Ana");
		Cliente bob = new Cliente("Bob");

		//                  saque  transferencia
		ana.realizarOperacao(true, true);
		bob.realizarOperacao(true, true);
	}

}
```

14. Implemente um mecanismo que verifica se o caixa eletrônico do exercício anterior (13) está sem cédulas de saque, em caso aformativo, deverá ser inicializada uma thread com o objetivo de alimentar novas cédulas no caixa eletrônico. Durante esse processo o caixa eletrônico ficará insidponível, esperando o terminado de alimentação das cédulas, apenas ao final do processo os clientes podem voltar a utilizar o caixa eletrônico.

Classe Cliente:

```
public class Cliente implements Runnable {

	private String nome;
	private Boolean saque;
	private Boolean transferencia;

	private static CaixaEletronico cx = new CaixaEletronico();

	public Cliente(String nome) {
		super();
		this.nome = nome;
	}

	public void realizarOperacao(Boolean saque, Boolean transferencia) {
		this.saque = saque;
		this.transferencia = transferencia;

		new Thread(this, nome).start();
	}

	@Override
	public void run() {
		// TODO Auto-generated method stub
		System.out.println("Cliente " + nome + " solicitou uso do caixa.");

		cx.operacao(nome, saque, transferencia);

		System.out.println("Cliente " + nome + " finalizou o uso do caixa.");
	}

}
```

Classe CaixaEletronico:

```
public class CaixaEletronico {

	private Integer totalCedulas = 300;

	public synchronized void operacao(String cliente, Boolean saque, Boolean
transferencia) {

		if( saque ) {
			System.out.println("Cliente " + cliente +
				" iniciou operacao de saque!");
			for(int aux=0; aux < 8; aux++) {
				try {
					System.out.println("[" + cliente +
					    "] Realizando saque...\t[total disponivel: " +
					    totalCedulas + "]");
					totalCedulas -= 100;
					Thread.sleep(1000);
				} catch (InterruptedException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}

				// Verificar se há cédulas disponíveis...
				if( totalCedulas <= 0 ) {
					System.out.println("Acabaram as cedulas!");

					Cedulas c = new Cedulas();
					Thread t = new Thread(c);
					c.abastecer(t);

					try {
						t.join();
					} catch (InterruptedException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}

					totalCedulas = 500;

					System.out.println("Novas cedulas inseridas!");
				}
			}
		}

		if( transferencia ) {
			System.out.println("Cliente " + cliente +
				" iniciou operacao de transferencia!");
			for(int aux=0; aux < 5; aux++) {
				try {
					System.out.println("[" + cliente +
						"] Realizando transferencia...");
					Thread.sleep(1000);
				} catch (InterruptedException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
			}
		}
	}

}
```

Classe Cedulas:

```
public class Cedulas implements Runnable{

	public void abastecer(Thread t) {

		t.start();

	}

	@Override
	public void run() {
		// TODO Auto-generated method stub

		try {
			for(int auxB=0; auxB< 4; auxB++) {
				System.out.println("Alimentando com mais cedulas...");
				Thread.sleep(1000);
			}
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}

}
```

Classe principal (App):

```
public class App {

	public static void main(String[] args) {

		Cliente ana = new Cliente("Ana");
		Cliente bob = new Cliente("Bob");

		//                  saque  transferencia
		ana.realizarOperacao(true, true);
		bob.realizarOperacao(true, true);
	}

 }
```