

INE 5410

PROGRAMAÇÃO CONCORRENTE

Prof. Vitorio Bruno Mazzola
mazzola@inf.ufsc.br



Capítulo 5 CONCORRÊNCIA EM JAVA



Introdução

- A Concorrência em Java

- Diferente de algumas linguagens de programação mais clássicas, a linguagem Java possui, de forma nativa, um mecanismo para a implementação de programas concorrentes, através do uso de múltiplas *threads*;
- Mesmo quando o programa não é concorrente, o fato de existir um programa *main()* de Java já provoca a criação de uma *thread*.

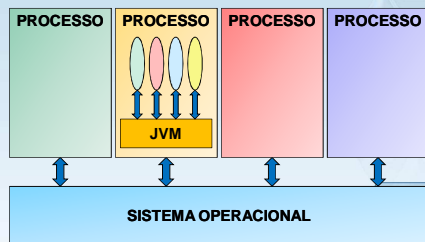
Introdução

- A Concorrência em Java

- A *Máquina Virtual Java (JVM)* executa numa plataforma de sistema operacional na forma de um processo único, gerenciado pelo núcleo do processo;
- Dentro da máquina virtual, as diversas *threads* executam, compartilhando e competindo pelos recursos do sistema;

Introdução

- A Concorrência em Java



Threads em Java

- A Classe Thread

- Java oferece uma classe `Thread` que faz parte do package `java.lang`
- Esta classe oferece um conjunto de métodos que podem ser utilizados para gerenciar um programa composto de diversas threads concorrentes
- Cada thread vai possuir um ciclo de vida que é caracterizado por diversos estados

Threads em Java

- Ciclo de vida de uma thread em Java



Threads em Java

- Métodos da Classe Thread

- O método `run` corresponde ao código que vai determinar a execução da thread
- Quando uma thread é criada num programa em Java, este método deverá ser reescrito para que se possa ter o comportamento específico desta da thread
- Uma thread é ativada a partir do método `start`, ativado a partir de alguma outra parte de código (o programa principal, por exemplo)

Threads em Java

• Métodos da Classe Thread

- O método `start` é utilizado para ativar uma thread
- Após sua execução, o método `start` retorna ao código de onde ele foi ativado, sendo que o código que o ativou passa a executar de modo concorrente com a thread ativada por este método
- O método `start` pode resultar numa `IllegalThreadStateException`, caso a thread que se tentou ativar já estava no estado *ready*

Threads em Java

• Métodos da Classe Thread

- O método `sleep` é utilizado para adormecer uma thread por uma quantidade de tempo (dada em *ms*)
- Ao ser colocada no estado *sleeping*, a thread não concorre pelo processador, o que significa que outras threads podem assumir o controle daquele recurso;
- Ao finalizar o tempo estabelecido na chamada deste método, a thread retorna ao estado *ready*, passando a concorrer com as demais threads

Threads em Java

• Métodos da Classe Thread

- O método `interrupt` é utilizado para interromper uma thread
- Um programa pode ativar o método `interrupted` para verificar se uma thread está interrompida;
- O método `isAlive` pode ser ativado para verificar se uma thread está em execução

Threads em Java

• Métodos da Classe Thread

- O método `join` é utilizado por uma thread para aguardar a finalização de outra para poder executar
- Os métodos `wait`, `notify` e `notifyAll` são utilizados no manuseio de monitores em Java
- `wait` faz com que a thread entre num estado de espera num dado objeto... `notify` ou `notifyAll` liberam a thread deste estado de espera (*waiting*)

Threads em Java

- Compondo um programa com Threads

- Um programa Java pode ser composto de diversas threads, as quais irão concorrer pelos diversos recursos do sistema e poderão comunicar-se para a realização de uma ou mais tarefas
- A linguagem oferece basicamente duas formas de criar *threads* no contexto de um programa, sendo uma delas a adoção da classe `Thread` disponível para extensão

Threads em Java

- Compondo um programa com Threads

- A primeira forma de criar *threads* é a criação de uma subclasse da classe nativa `Thread`, e sobrescrever o método `run` para implementar o comportamento da nova classe
- A outra forma é o uso da interface **`Runnable`**, a qual vai permitir criar *threads* que possam derivar de outras classes no contexto do programa de aplicação

Threads em Java

- Criando uma subclasse thread

```
public class ExemploThread extends Thread {  
    public void run() {  
        int total = 0;  
        while ((int)(Math.random()*100) != 50)  
            total++;  
        System.out.println("Sou a " + this.getName() + " e  
            tentei " + total + " vezes.");  
    }  
}
```

```
ExemploThread t = new ExemploThread();  
t.start();
```

Threads em Java

- Criando uma subclasse thread

- No exemplo anterior, foi criada uma subclasse da classe *thread* a qual executa uma adição de números gerados aleatoriamente, parando de executar quando o número gerado for igual a 50;
- Na segunda parte do código, é mostrado a instanciação da nova thread e sua ativação a partir da chamada do método `start`;

Threads em Java

- Outra forma de criação de Threads

- Como indicado anteriormente, é possível criar threads num programa em Java através da implementação da interface **Runnable**;
- As interfaces são um mecanismo criado em Java que permite que uma dada classe possa herdar atributos e métodos de mais de uma classe, servindo de meio de implementar a herança múltipla;

Threads em Java

- Outra forma de criação de Threads

- A vantagem da criação de threads desta segunda forma é que torna-se possível à classe criada estender uma outra classe que não seja a classe **Thread**;
- Implementando a interface **Runnable**, a classe tem o benefício de tornar-se um objeto executável e, ao mesmo tempo, herdar atributos e métodos de alguma outra classe (que não seja a classe **Thread**);

Threads em Java

- Outra forma de criação de Threads

```
public class Exec extends SuperClasse implements Runnable
{
    public void run()
    { . . .
        // código do método run()
    }
}
```

```
Runnable runnable = new mThread();
Thread thread1 = new Thread(runnable);
thread1.start();
```

Threads em Java

- Comparando as duas abordagens

```
public class MyThread extends Thread {
    public void run()
    {
        System.out.println("A thread executou!");
    }
}
```

```
public class MyRunnable implements Runnable {
    public void run()
    {
        System.out.println("A thread executou!");
    }
}
```

Prioridade entre threads

- Níveis de Prioridade

- Numa aplicação Java, threads podem assumir diferentes níveis de prioridade, indo de um nível (`Thread.MIN_PRIORITY`) mínimo a um nível máximo (`Thread.MAX_PRIORITY`);
- O nível mínimo corresponde a uma constante igual a 1 e o nível máximo, a 10;
- Por default, threads criadas assumem o nível “normal” de prioridade, que corresponde à constante igual a 5 (`Thread.NORM_PRIORITY`)

Prioridade entre threads

- Métodos relacionados à prioridades

- O nível de prioridade das threads pode ser definido através do método `set_priority`, que faz uso de um argumento inteiro que deve estar entre 1 e 10;
- O método `get_priority` é utilizado para obter o nível de prioridade de uma dada thread;
- O método `yield` permite que uma thread ofereça a possibilidade de outras threads do mesmo nível de prioridade assumam o controle do processador

Escalonamento de threads

- Escalonador

- Algumas plataformas Java suportam um mecanismo conhecido por *timeslicing*, o qual permite que threads possam alternar-se na execução, por uma janela de tempo denominada *quantum*;
- Se o escalonador não suporta *timeslicing*, uma thread em execução só deixa o estado **running** quando:
 - Finaliza sua execução
 - Vai para um dos estados **waiting**, **sleeping** ou **blocked**
 - É interrompida por uma thread de mais alta prioridade

Escalonamento de threads

- Escalonador

- Numa plataforma que suporte *timeslicing*, uma thread dispõe de um *quantum* para executar;
- Ao final deste período, mesmo que a thread não tenha finalizado sua execução, ela irá ceder lugar a outra thread de mesmo nível de prioridade (se houver pelo menos uma no estado **ready**)
- A preocupação do escalonador deverá ser a de manter sempre a thread de maior prioridade com o controle do processador

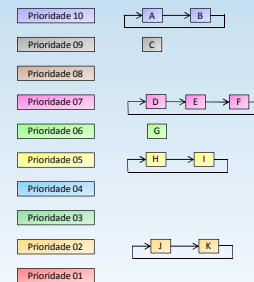
Escalonamento de threads

• Resumindo o escalonamento...

- Uma thread pode sair do estado **running** nas seguintes circunstâncias:
 - finaliza sua execução;
 - chama os métodos **wait**, **sleep** ou **yield**
 - é interrompida por uma thread de maior prioridade
 - numa plataforma que suporte timeslicing, seu quantum expirou.

Escalonamento de threads

• Resumindo o escalonamento...



Threads em Java

• Exemplo 1 – Múltiplas Threads

```
public class MinhasThreads extends Thread {
    private int sleepTime;

    // Construtor de MinhasThreads atribui um nome para a thread
    // chamando o construtor da superclasse Thread
    public MinhasThreads( String name )
    {
        super( name );
        // Thread dorme entre 0 e 5 s
        sleepTime = (int) ( Math.random() * 5000 );
        // Imprime o nome e o sleeptime
        System.err.println("Name: " + getName() + "; sleep: " + sleepTime );
    }
}
```

Threads em Java

• Exemplo 1 – Múltiplas Threads

```
public void run()
{
    // adormece a thread por um intervalo randômico entre 0 e 5 s
    try { System.err.println( getName() + " indo dormir" );
        // coloca a thread para dormir
        Thread.sleep( sleepTime );
    }
    // se a thread é interrompida durante o sono, trata a exceção
    // e imprime mensagem de erro
    catch ( InterruptedException interruptedException ) {
        System.err.println( interruptedException.toString() );
    }
    // impressão
    System.err.println( getName() + " acordou" );
}
// fim MinhasThreads
```


Threads em Java

• Exemplo 1 – Múltiplas Threads

```
public class Main {
    // cria e inicializa as threads
    public static void main( String args[] )
    {
        MinhasThreads thread1, thread2, thread3, thread4;
        // cria as quatro threads
        thread1 = new MinhasThreads( "thread1" );
        thread2 = new MinhasThreads( "thread2" );
        thread3 = new MinhasThreads( "thread3" );
        thread4 = new MinhasThreads( "thread4" );
        System.err.println( "\nIniciando as threads" );
        // inicia a execução das threads
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        System.err.println( "Threads iniciadas\n" );
    } // fim da classe Main
}
```

Threads em Java

• Exemplo 1 – Múltiplas Threads

```
Name: thread1; sleep: 517
Name: thread2; sleep: 1046
Name: thread3; sleep: 3712
Name: thread4; sleep: 3273
```

```
Iniciando as threads
thread1 indo dormir
thread3 indo dormir
Threads iniciadas
```

```
thread2 indo dormir
thread4 indo dormir
thread1 acordou
thread2 acordou
thread4 acordou
thread3 acordou
```

Threads em Java

• Exemplo 2 – Prod/Cons sem sincronização

```
public class SharedCell {

    public static void main(String[] args) {
        HoldIntegerUnsynchronized sharedObject = new
        HoldIntegerUnsynchronized();
        Produtor produtor1 = new Produtor( sharedObject );
        Consumidor consumidor1 = new Consumidor( sharedObject );

        produtor1.start();
        consumidor1.start();
    }
}
```

Threads em Java

• Exemplo 2 – Prod/Cons sem sincronização

```
public class SharedCell {

    public static void main(String[] args) {
        HoldIntegerUnsynchronized sharedObject = new
        HoldIntegerUnsynchronized();
        Produtor produtor1 = new Produtor( sharedObject );
        Consumidor consumidor1 = new Consumidor( sharedObject );

        produtor1.start();
        consumidor1.start();
    }
}
```


Threads em Java

• Exemplo 2 – Prod/Cons sem sincronização

```
public class HoldIntegerUnsynchronized {
    private int sharedInt = -1;
    public void setSharedInt( int value ) {
        System.err.println( Thread.currentThread().getName() + "
ajustando valor para " + value );
        sharedInt = value;
    }

    public int getSharedInt() {
        System.err.println( Thread.currentThread().getName() + "
recuperando valor " + sharedInt );
        return sharedInt;
    }
}
```

Threads em Java

• Exemplo 2 – Prod/Cons sem sincronização

```
public class Produtor extends Thread {
    private HoldIntegerUnsynchronized sharedObject;
    public Produtor( HoldIntegerUnsynchronized shared )
    {
        super( "Produtor" );
        sharedObject = shared;
    }

    public void run() {
        for ( int count = 1; count <= 10; count++ ) {
            try { Thread.sleep( ( int ) ( Math.random() * 3000 ) ); }
            catch( InterruptedException exception ) { System.err.println(
exception.toString() ); }
            sharedObject.setSharedInt( count );
        }
        System.err.println( getName() + " finished producing values" +
"\nTerminating " + getName() );
    }
}
```

Threads em Java

• Exemplo 2 – Prod/Cons sem sincronização

```
public class Consumidor extends Thread {
    private HoldIntegerUnsynchronized sharedObject;

    public Consumidor( HoldIntegerUnsynchronized shared ) {
        super( "Consumidor" );
        sharedObject = shared;
    }

    public void run() {
        int value, sum = 0;
        do { try { Thread.sleep( (int) ( Math.random() * 3000 ) ); }
            catch( InterruptedException exception )
            { System.err.println( exception.toString() ); }

            value = sharedObject.getSharedInt();
            sum += value;
        } while ( value != 10 );
        System.err.println( getName() + " retrieved values totaling: " + sum +
"\nTerminating " + getName() );
    }
}
```

Threads em Java

• Exemplo 2 – Prod/Cons sem sincronização

```
Consumidor Recuperando valor -1
Consumidor Recuperando valor -1
Consumidor Recuperando valor -1
Produtor Ajustando valor 1
Produtor Ajustando valor 2
Consumidor Recuperando valor 2
Produtor Ajustando valor 3
Consumidor Recuperando valor 3
Consumidor Recuperando valor 3
Consumidor Recuperando valor 3
Produtor Ajustando valor 4
Produtor Ajustando valor 5
Consumidor Recuperando valor 5
Consumidor Recuperando valor 5
Consumidor Recuperando valor 5
Consumidor Recuperando valor 5
Produtor Ajustando valor 6
Consumidor Recuperando valor 6

Produtor Ajustando valor 7
Consumidor Recuperando valor 7
Produtor Ajustando valor 8
Produtor Ajustando valor 9
Consumidor Recuperando valor 9
Produtor Ajustando valor 10
Produtor Definindo valor
Finalizando Produtor
Consumidor Recuperando valor 10
Consumidor Total recuperado: 55
Finalizando Consumidor
```

Sincronizando as threads

- Monitores

- Java faz uso de monitores (conceito criado por C. A. Hoare) para realizar a sincronização das threads
- O programador pode criar tantos monitores quantos sejam necessários para controlar a concorrência entre as threads, dependendo dos recursos que estas compartilham e pelas quais concorrem
- Os monitores controlam a execução de **métodos sincronizados** para controlar o acesso ao monitor

Sincronizando as threads

- Monitores e Métodos Sincronizados

- Quando uma thread quer acessar um monitor para entrar em sua seção crítica, ela ativa um dos métodos sincronizados do monitor
- Se o monitor estiver livre, permite a execução do código do método no contexto da thread
- Caso contrário, a thread é bloqueada e ficará aguardando sua vez para acessar o mesmo
- A liberação das threads pelo monitor obedece ao esquema de prioridades suportado em Java

Sincronizando as threads

- Monitores e Métodos Sincronizados

- Um método sincronizado é definido com o auxílio do modificador de métodos **synchronized**
- Ao modificar um método para “sincronizado”, o sistema irá gerenciar o acesso ao método de modo que apenas uma thread possa acessá-lo a cada vez
- Se uma classe contiver mais de um método sincronizado, o acesso aos métodos é feito de modo que apenas um método sincronizado da classe pode ser acessado a cada vez

Sincronizando as threads

- Monitores e Métodos Sincronizados

- Se, no código de um método sincronizado ativado por uma dada thread, existe a chamada ao método **wait()**, a execução da thread será interrompida, passando a thread ao estado **waiting**;
- Por outro lado, quando, num método sincronizado, existe a chamada ao método **notify()** ou **notifyAll()**, uma ou diversas threads bloqueadas no estado waiting são sinalizadas e retornam ao estado **ready**, sendo que uma delas pode assumir o controle do monitor;

Sincronizando as threads

• Monitores e Métodos Sincronizados

- Nos monitores em Java, não existe o conceito de *variável condição*... as threads não estarão associadas a filas específicas de uma ou diversas variáveis;
- Elas apenas irão posicionar-se no estado *waiting* e, ao serem notificadas, retornam ao estado *ready*;

Sincronizando as threads

• Implementando um Semáforo em Java

- Em casos mais simples, ou no caso de código legado, onde o mecanismo adotado para controlar a concorrência foi o semáforo, pode ser interessante se poder manter o uso deste mecanismo;
- Sendo assim, é possível considerar a implementação de semáforos com o auxílio do conceito de métodos sincronizados.

Sincronizando as threads

• Implementando um Semáforo em Java

```
public class Mutex {
    private int valor = 0;

    public Mutex (int inicial)
    { valor = inicial; }

    public synchronized void P()
    throws InterruptedException {
        while ( valor == 0 ) wait();
        valor--;
    }

    public synchronized void V()
    throws InterruptedException {
        if ( valor == 0 ) notify();
        valor++;
    }
} // Fim Classe Mutex
```

Exemplos

• Exemplo 4 – Prod/Cons com monitor

```
public class SharedCell {

    public static void main( String args[] )
    {
        HoldIntegerSynchronized sharedObject = new HoldIntegerSynchronized();

        ProduceInteger producer = new ProduceInteger( sharedObject );
        ConsumerInteger consumer = new ConsumerInteger( sharedObject );

        producer.start();
        consumer.start();
    }
}
```

Exemplos

- Exemplo 4 – Prod/Cons com monitor

```
public class HoldIntegerSynchronized {
    private int sharedInt = -1;
    private boolean writeable = true; // condition variable

    public synchronized void setSharedInt( int value ) {
        while ( !writeable ) { // not the producer's turn
            try {
                wait();
            } catch ( InterruptedException exception ) {
                exception.printStackTrace();
            }
        }
        System.err.println( Thread.currentThread().getName() + "
        setting sharedInt to " + value );
        sharedInt = value;
        writeable = false;
        notify();
    }
}
```

Exemplos

- Exemplo 4 – Prod/Cons com monitor

```
public synchronized int getSharedInt() {
    while ( !writeable ) {
        try { wait(); }
        catch ( InterruptedException exception ) {
            exception.printStackTrace();
        }
    }
    writeable = true;
    notify();
    System.err.println( Thread.currentThread().getName() + "
    retrieving sharedInt value " + sharedInt );
    return sharedInt;
}
}
```

Exemplos

- Exemplo 4 – Prod/Cons com monitor

```
public class ProduceInteger extends Thread {
    private HoldIntegerSynchronized sharedObject;
    public ProduceInteger( HoldIntegerSynchronized shared ) {
        super( "ProduceInteger" );
        sharedObject = shared;
    }

    public void run() {
        {
            for ( int count = 1; count <= 10; count++ ) {
                // sleep for a random interval
                try { Thread.sleep( ( int ) ( Math.random() * 3000 ) ); }
            }
            catch( InterruptedException exception ) {
            }
        }
    }
}
```

Exemplos

- Exemplo 4 – Prod/Cons com monitor

```
System.err.println( exception.toString() );
}

sharedObject.setSharedInt( count );
}

System.err.println( getName() + " finished producing values" +
"\nTerminating " + getName() );
}
}
```

Exemplos

• Exemplo 4 – Prod/Cons com monitor

```
public class ConsumerInteger extends Thread {
    private HoldIntegerSynchronized sharedObject;
    public ConsumerInteger( HoldIntegerSynchronized shared ) {
        super( "ConsumerInteger" );
        sharedObject = shared;
    }
    public void run() {
        int value, sum = 0;
        do {
            try { Thread.sleep( (int) ( Math.random() * 3000 ) ); }
            catch( InterruptedException exception ) {
                System.err.println( exception.toString() ); }
            value = sharedObject.getSharedInt();
            sum += value;
        }
        while ( value != 10 );
        System.err.println( getName() + " retrieved values totaling: " + sum
+ "\nTerminating " + getName() );
    }
}
```

Exemplos

• Exemplo 4 – Prod/Cons com monitor

```
Produtor ajustando valor a 1
Consumidor recuperando valor 1
Produtor ajustando valor a 2
Consumidor recuperando valor 2
Produtor ajustando valor a 3
Consumidor recuperando valor 3
Produtor ajustando valor a 4
Consumidor recuperando valor 4
Produtor ajustando valor a 5
Consumidor recuperando valor 5
Produtor ajustando valor a 6
Consumidor recuperando valor 6
Produtor ajustando valor a 7
Consumidor recuperando valor 7
Produtor ajustando valor a 8
Consumidor recuperando valor 8
Produtor ajustando valor a 9
Consumidor recuperando valor 9
Produtor ajustando valor a 10
```

```
Produtor Finalizou geração de valores
Finalizando Produtor
Consumidor recuperando valor 10
Consumidor Total dos valores obtidos: 55
Finalizando Consumidor
```

Exemplos

• Exemplo 5 – Controle da Piscina

```
public class Status {
    String nadador [];
    int numNadador;
    public Status ( int numNadador ){
        this.numNadador = numNadador;
        nadador = new String [ numNadador ];
        for (int i = 0; i < numNadador ; i++){
            nadador [ i ] = "CA";
            System.out.print(" " + nadador [ i ] );
        }
        System.out.println();
    }
}
```

Exemplos

• Exemplo 5 – Controle da Piscina

```
public synchronized void change ( int nid , String newStatus ){
    nadador [ nid ] = newStatus;
    int nadando = 0;
    for (int i = 0; i < numNadador ; i++){
        if ( nadador [ i ] == "NA" ) nadando++;
        System.out.print(" " + nadador [ i ] );
    }
    System.out.print(" NADANDO: " + nadando );
    System.out.println();
}
```

Exemplos

- Exemplo 5 – Controle da Piscina

```
public class Recurso {
    int nrec;

    public Recurso ( int nrec ){
        this.nrec = nrec;
    }

    public synchronized void pega(){
        try{
            while ( nrec == 0 ) wait();
            nrec--;
        } catch ( Exception e ){ }
    }

    public synchronized void larga(){
        nrec++;
        notify();
    }
}
```

Exemplos

- Exemplo 5 – Controle da Piscina

```
public class Nadador extends Thread {
    Recurso cesto, cabine;
    Status estado;
    int nid;

    public Nadador ( Recurso cesto, Recurso cabine, Status estado, int
nid ){
        this.cesto = cesto;
        this.cabine = cabine;
        this.estado = estado;
        this.nid = nid;
    }
}
```

Exemplos

- Exemplo 5 – Controle da Piscina

```
public void run (){
    try{ while ( true ){
        estado.change(nid , "CA");
        sleep ( (int) ( Math.random()* 500 ) );
        cesto.pega();
        cabine.pega();
        estado.change (nid , "TR");
        sleep ( (int) ( Math.random()* 1000 ) );
        cabine.larga();
        estado.change (nid , "NA");
        sleep ( (int) ( Math.random()* 1500 ) );
        cabine.pega();
        estado.change (nid , "CR");
        sleep ( (int) ( Math.random()* 500 ) );
        cesto.larga();
        cabine.larga();
    } } catch(Exception e ){ }
}
```

Exemplos

- Exemplo 5 – Controle da Piscina

```
public class Main {
    public static void main ( String [] s ){
        Recurso cesto , cabine;
        Status estado;
        cesto = new Recurso ( 8 );
        cabine = new Recurso ( 3 );
        estado = new Status ( 15 );
        Nadador nadador[];
        nadador = new Nadador [ 15 ];
        for (int i = 0 ; i < 15 ; i++){
            nadador [ i ] = new Nadador ( cesto, cabine, estado, i );
            nadador [ i ].start();
        }
    }
}
```


Exemplos

- Exemplo 6 – Pombos-Correio

```
public class Main {
    public static void main ( String [] s ){
        CaixaPostal caixa = new CaixaPostal ();
        Gaiola gaiola = new Gaiola();
        Pombo pombo[] = new Pombo [ 12 ];
        Pessoa pessoa[] = new Pessoa [ 20 ];
        for ( int i = 0 ; i < 12; i++ ){
            pombo [ i ] = new Pombo ( gaiola, caixa, i+1);
            pombo [ i ].start();
        }
        for ( int i = 0 ; i < 20; i++ ){
            pessoa [ i ] = new Pessoa ( caixa );
            pessoa [ i ].start();
        }
    }
}
```

Exemplos

- Exemplo 7 – Fumantes

```
public class Mesa {
    private int pedra;
    public Mesa (){ pedra = 0; }
    public synchronized int consulta( int fid ){
        int aux;
        aux = pedra;
        if ( pedra == fid ) pedra = 0;
        return aux;
    }
    public synchronized void atualiza ( int novapedra ){
        try{ pedra = novapedra;
            wait();
        } catch( Exception e ){}
    }
    public synchronized void wakeUpAg( int fid ){
        System.out.println( " FUMANTE: " + fid + " ACORDANDO AGENTE " );
        notify();
    }
}
```

Exemplos

- Exemplo 7 – Fumantes

```
public class Fumante extends Thread {
    int pedra, fid; Mesa mesa;
    public Fumante ( int fid, Mesa m ){
        this.fid = fid; mesa = m;
    }
    public void run(){
        while ( true ){
            try{
                pedra = mesa.consulta ( fid );
                while ( pedra != fid ){
                    yield(); // libera a CPU
                }
                pedra = mesa.consulta ( fid );
            }
            System.out.println( " FUMANTE: " + fid + " FUMANDO " );
            sleep ( ( int ) (Math.random()* 1000 ));
            mesa.wakeUpAg(fid);
        } catch ( Exception e ){System.out.println(e);}
    }
}
```

Exemplos

- Exemplo 7 – Fumantes

```
public class Agente extends Thread{
    int pedra;
    Mesa mesa;
    public Agente ( Mesa m ){
        mesa = m;
    }
    public void run(){
        while ( true ){
            try{
                pedra = ( int ) ( 1 + Math.random()*3 );
                System.out.println( " PEDRA SORTEADA: " + pedra );
                sleep ( ( int ) (Math.random()* 1000 ));
            } catch (Exception e) {System.out.println(e);}
            mesa.atualiza(pedra );
        }
    }
}
```

Exemplos

• Exemplo 7 – Fumantes

```
public class Main {  
  
    public static void main(String[] args) {  
        Fumante f1, f2, f3;  
        Agente ag;  
        Mesa mesa = new Mesa ();  
        f1 = new Fumante ( 1, mesa );  
        f2 = new Fumante ( 2, mesa );  
        f3 = new Fumante ( 3, mesa );  
  
        ag = new Agente ( mesa );  
  
        f1.start();  
        f2.start();  
        f3.start();  
        ag.start();  
    }  
}
```

Exemplos

• Exemplo 8 – Estacionamento

```
public class Parking {  
    private int vaga, nprof, nfunc, naluno;  
    public Parking ( int vaga ){  
        this.vaga = vaga;  
        nprof = 0;  
        nfunc = 0;  
        naluno = 0;  
    }  
  
    public synchronized void in( String classe, int pid ){  
        try{  
            if ( vaga == 0 ) {  
                System.out.println( classe+" NUM: "+pid+ " # FILA " );  
                if ( classe == "PROFESSOR"){  
                    nprof++;  
                    while ( vaga == 0 ) wait();  
                    nprof--;  
                }  
            }  
        }  
    }  
}
```

Exemplos

• Exemplo 8 – Estacionamento

```
        else{  
            if ( classe == "FUNCIONARIO"){  
                nfunc++;  
                while ( ( vaga == 0 ) || ( nprof > 0 ) ) wait();  
                nfunc--;  
            }  
            else{  
                naluno++;  
                while ( ( vaga == 0 ) || ( nprof > 0 ) || ( nfunc > 0 ) )  
                    wait();  
                naluno--;  
            }  
        }  
        vaga--;  
        System.out.println( classe+" NUM: "+pid+ " >> ENTRANDO " );  
    } catch (Exception e ){}  
}
```

Exemplos

• Exemplo 8 – Estacionamento

```
    public synchronized void out ( String classe, int pid ){  
        System.out.println( classe+" NUM: "+pid+ " << SAINDO " );  
        vaga++;  
        notifyAll();  
    }  
}
```

Exemplos

• Exemplo 8 – Estacionamento

```
public class Pessoa extends Thread{
    String classe;
    int pid;
    Parking parking;
    public Pessoa ( String classe , int pid, Parking parking ){
        this.classe = classe;
        this.pid = pid;
        this.parking= parking;
    }

    public void run(){
        try{
            sleep( ( int )(Math.random()*9000 ));
            parking.in( classe , pid);
            sleep( ( int )(Math.random()*7000 ));
            parking.out( classe , pid);
        }catch(Exception e ){}
    }
}
```

Exemplos

• Exemplo 8 – Estacionamento

```
public class Main {
    public static void main ( String [] s ){
        Parking parking = new Parking ( 10 );
        Pessoa [] prof  = new Pessoa [ 10 ];
        Pessoa [] func  = new Pessoa [ 15 ];
        Pessoa [] aluno = new Pessoa [ 30 ];
        for ( int i = 0 ; i < 10 ; i ++ ){
            prof[i] = new Pessoa("PROFESSOR", i, parking );
            prof[i].start();
        }
        for ( int i = 0 ; i < 15 ; i ++ ){
            func[i] = new Pessoa("FUNCIONARIO", i, parking );
            func[i].start();
        }
        for ( int i = 0 ; i < 30 ; i ++ ){
            aluno[i] = new Pessoa("ALUNO", i, parking );
            aluno[i].start();
        } }
}
```

Exemplos

• Exemplo 9 – Jantar dos Filósofos – Versão 1

```
public class Garfos {
    private int[] D, E, G;
    private char[] ST;
    public Garfos(){
        D = new int [ 5 ];
        E = new int [ 5 ];
        G = new int [ 5 ];
        ST = new char [ 5 ];
        for (int i= 0; i < 5 ; i++ ){
            ST [ i ] = 'P';
            D [ i ] = i + 1;
            E [ i ] = i - 1;
            G [ i ] = 2;
        }
        D [ 4 ] = 0;
        E [ 0 ] = 4;
        printST();
    }
}
```

Exemplos

• Exemplo 9 – Jantar dos Filósofos – Versão 1

```
void printST(){
    System.out.println("=====");
    for ( int f = 0; f < 5 ; f++ ) System.out.print(" "+ST[ f ]);
    System.out.println();
    System.out.println("=====");
}

public synchronized void pega( int fid ){
    try{
        while ( G [ fid ] != 2 ){
            ST[ fid ] = 'F';
            printST();
            wait();
        }
        G [ D [ fid ] ]--; G [ E [ fid ] ]--;
        ST[ fid ] = 'C';
        printST();
    } catch ( Exception e ){
    }
}
```

Exemplos

- Exemplo 9 – Jantar dos Filósofos – Versão 1

```
public synchronized void larga( int fid ){
    G [ D [ fid ] ]++; G [ E [ fid ] ]++;
    ST[ fid ] = 'P';
    printST();
    notifyAll();
}
```

Exemplos

- Exemplo 9 – Jantar dos Filósofos – Versão 1

```
public class Filosofo extends Thread {
    private int fid;
    Garfos garfos;
    public Filosofo ( int fid, Garfos garfos ){
        this.fid = fid;
        this.garfos = garfos;
    }
    public void run(){
        try{
            while ( true ){
                sleep ( ( int ) ( Math.random() * 1500 ) );
                garfos.pegar ( fid );
                sleep ( ( int ) ( Math.random() * 800 ) );
                garfos.larga( fid );
            }
        }catch ( Exception e ){}
    }
}
```

Exemplos

- Exemplo 9 – Jantar dos Filósofos – Versão 1

```
public class Main {
    public static void main ( String [] s ){
        Garfos garfos = new Garfos ();
        Filosofo[] filo = new Filosofo [ 5 ];
        for ( int f = 0 ; f < 5 ; f++ ){
            filo [ f ] = new Filosofo ( f, garfos );
            filo [ f ].start();
        }
    }
}
```