Curso de Java

aula 09

Prof. Anderson Henrique

TECH
soluções em ti

Threads

Geralmente escrevemos programas que executam um passo de cada vez em sequência. Essa sequencia de passos executada um de cada vez é chamada de thread, ou seja, até agora programamos usando uma única thread por programa.

Na vida real, várias atividades são feitas ao mesmo tempo, um caixa de banco, p.ex.: ele realiza uma operação por vez, mas, dentro de um banco nós temos vários caixas trabalhando ao mesmo tempo, para simular esses vários caixas precisaríamos usar uma thread para cada caixa.

TECH
soluções em ti

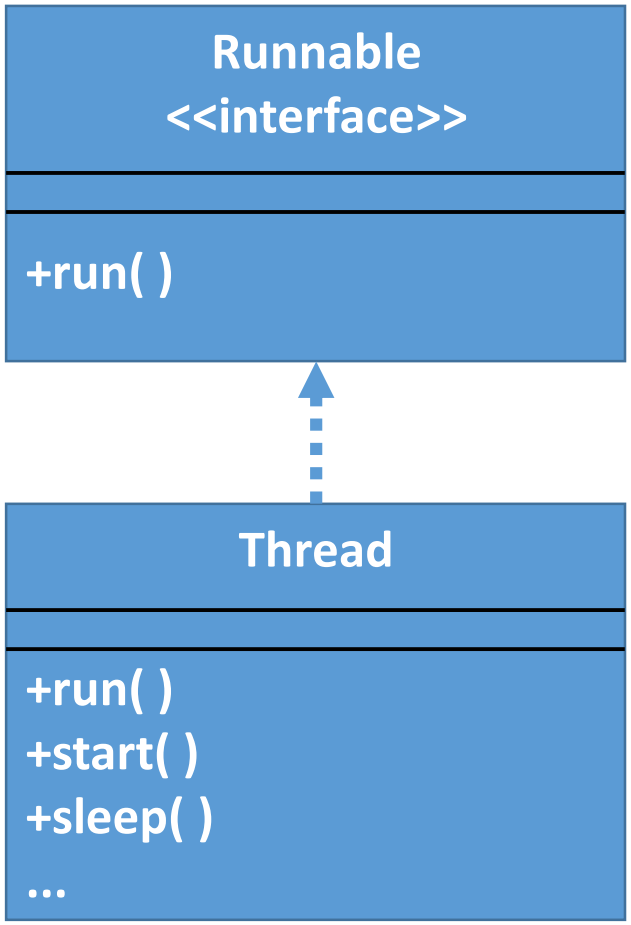
Assim poderíamos ter uma situação mais parecida com a vida real. Esses programas que utilizam várias threads, são chamados de multithread, ou seja, eles são capazes de executar várias atividades ao mesmo tempo.

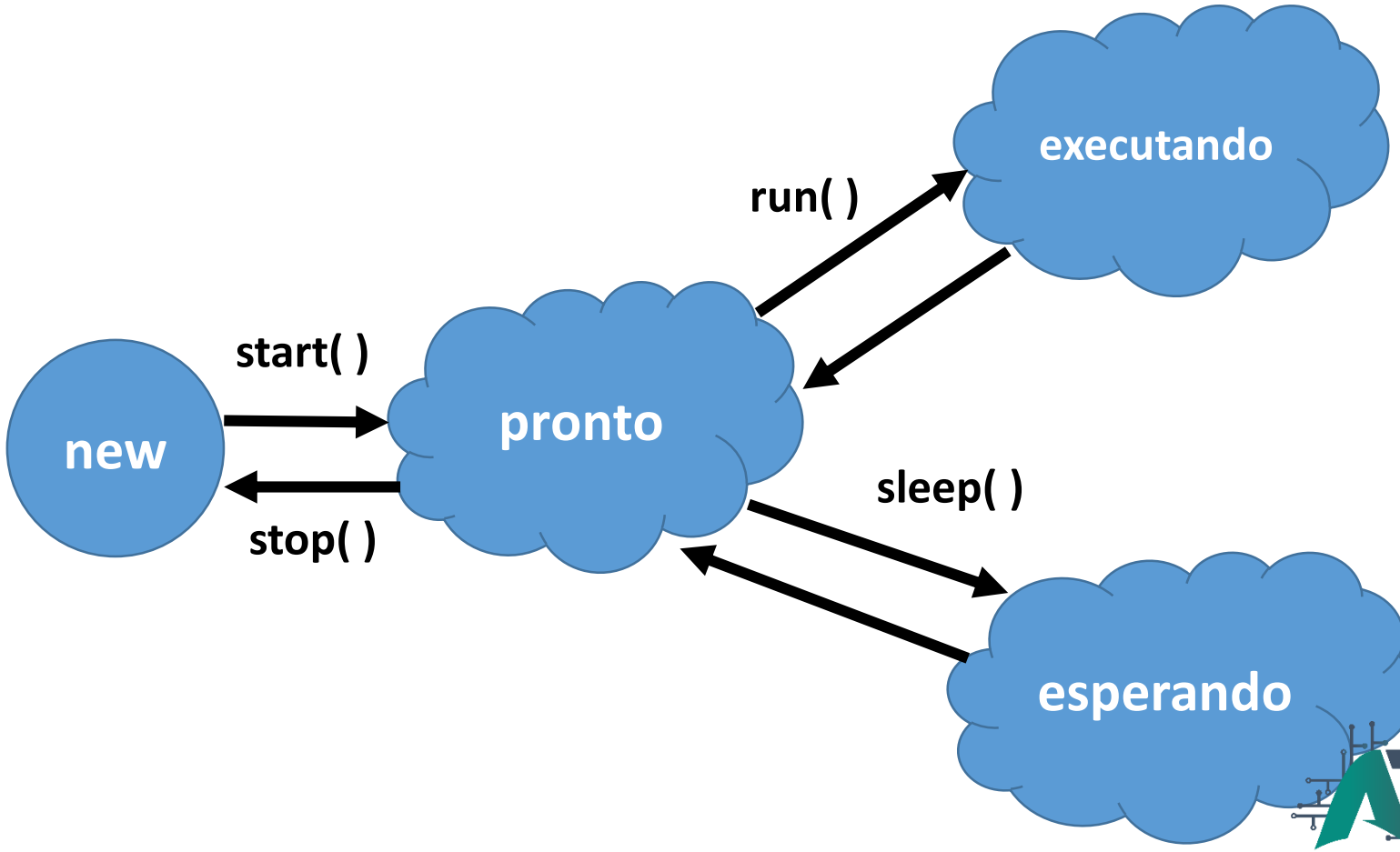
Essa é a capacidade que todo o computador tem, e a linguagem Java permite que utilizemos esse recurso, então o objetivo dessa aula é sair do desenvolvimento tradicional, onde uma atividade é executada de cada vez, para um desenvolvimento multithread onde várias atividades podem acontecer simultaneamente.

E como é que gerenciamos estas várias atividades simultâneas?

TECH
soluções em ti

Através dos objetos Thread. As classes thread implementam uma interface chamada Runnable, que define o objeto executável. Todo o objeto executável tem o método run( )





TECH
soluções em ti

Utilizando a classe Thread

E qual é o ciclo de vida de uma thread? Como inicia, executa e morre? É isso que vamos ver nessa aula.

Vamos criar um novo pacote: br.com.treinacom.java.threads, e dentro do pacote vamos criar nossa classe principal chamada PingPong. Antes de criar uma thread, precisamos entender que “todo programa roda sobre uma thread”, seja ela definida por nós ou não. A JVM cria threads para que um programa seja executável.

Ex.: Thread t = new Thread( ); //criamos uma Thread

A nossa classe PingPong deverá ser uma extensão da classe Thread, no exemplo a seguir vamos criar uma thread que imprimirá uma palavra diversas vezes, no nosso método construtor.

TECH
soluções em ti

Ex.:

```
String palavra;
long tempo;
public PingPong(String palavra, long tempo){
        this.palavra = palavra;
        this.tempo = tempo; //milissegundos
}
```

Usando thread, pode usufruir dos estados da thread, como p.ex.: aguardar um determinado tempo, inicializar ou até mesmo finalizar uma tarefa.

TECH
soluções em ti

```
Public static void main(String args[ ]){
        new PingPong(“ping”, 1500);
        new PingPong(“PONG”, 2000);
}
```

Criar uma thread não a coloca no estado pronto para ser executada, para a JVM recebe-la para começar a executá-la, precisamos utilizar o método start( ).

```
        new PingPong(“ping”, 1500).start( );
        new PingPong(“PONG”, 2000).start( );
```

Para executar, precisamos utilizar o método run( ), vamos criar o nosso método.

TECH
soluções em ti

Ex..:

```
public void run( ){
try{
        for(int i = 0; i < 5; i++){
                System.out.println(palavra);
                Thread.sleep(tempo); //método estático
        }
        }catch(InterruptedException e){

        }
}
```

O método sleep coloca a thread em estado de espera, ele lança uma exceção, podemos trata-la com InterruptedException.

TECH
soluções em ti

E, quando a exceção acontecer, queremos parar a execução da thread, como podemos fazer isso? Assim que o método run( ) terminar de ser executado, vamos parar a execução da thread.

A grande vantagem é que as atividades são executadas simultaneamente. Nunca devemos executar diretamente o método run( ), é de responsabilidade da nossa JVM.

TECH
soluções em ti

Implementando a Interface Runnable

Agora, iremos trabalhar com a interface Runnable. Toda Thread implementa Runnable, que define o objeto executável, e o único método que as classes Runnable tem que implementar é o método run( ).

Então, você utilizar objetos Runnable no lugar de Thread, como? Vamos criar uma classe principal chamada: PingPongRunnable.

Ex.:

```
public class PingPongRunnable implements Runnable{


}
```

Se não tivesse o método run( ) implementado, seria solicitado adicionar o método não implementado.

TECH
soluções em ti

Como podemos executar o método run( ) se a minha classe não é uma Thread?

```
Runnable ping = new PingPongRunnable(“ping”, 1500);
Runnable pong = new PingPongRunnable(“PONG”, 2000);
```

```
new Thread(ping, “ping”).start( );
new Thread(pong, “pong”).start( );
```

Então podemos criar Threads a partir da classe Thread e também implementando a interface Runnable.

TECH
soluções em ti

Synchronized Threads

Imagina a seguinte situação. Você foi contratado por um banco para construir um sistema de saque para contas conjuntas. As contas conjuntas são aquelas em que várias pessoas podem mexer na mesma conta ao mesmo tempo.

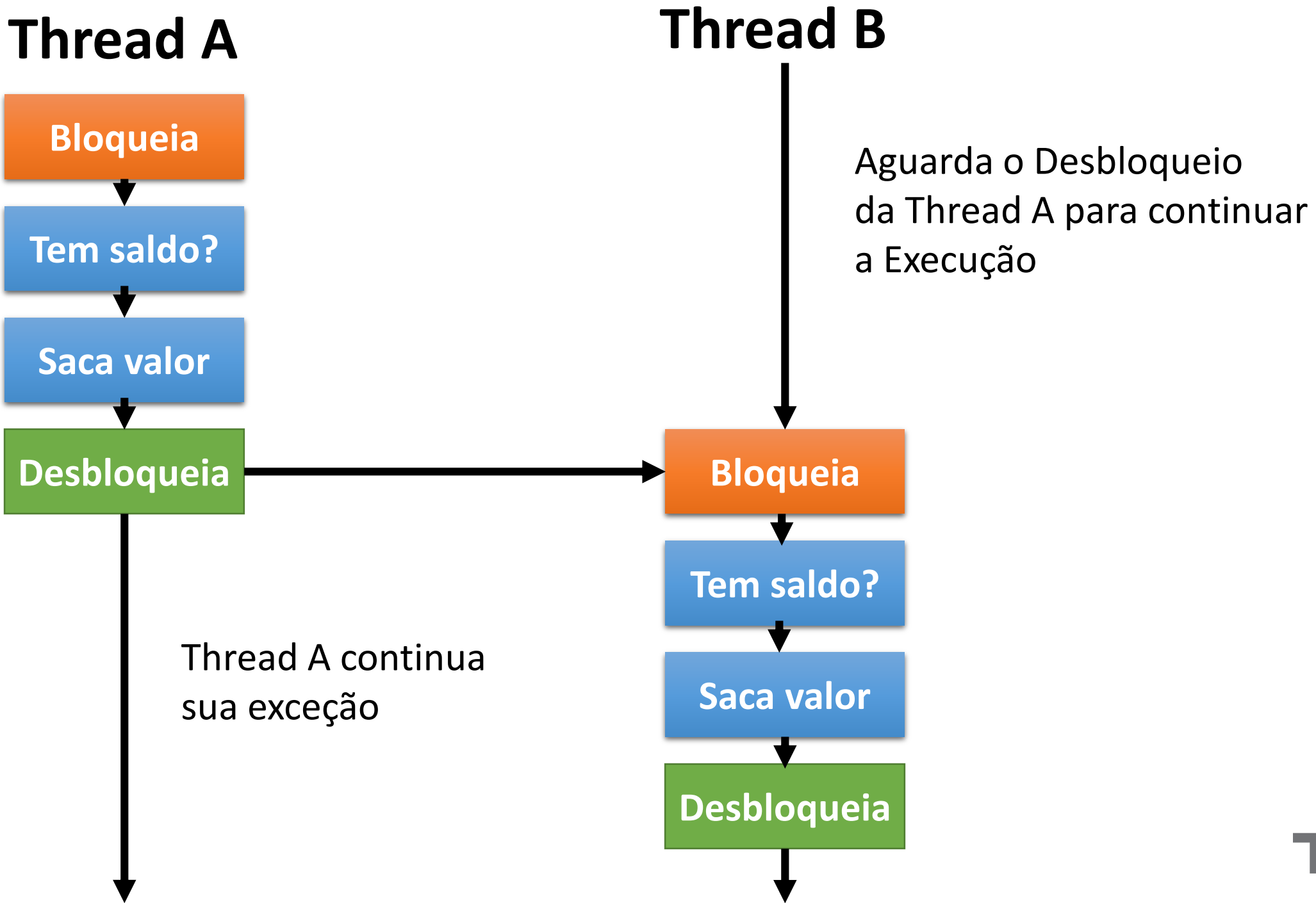
Vamos construir um programa onde diversas pessoas realizem saques simultaneamente, mas podemos ter um grande problema. Imagine, p.ex.: uma conta que tem R$ 100,00 de saldo, e 10 pessoas ao mesmo tempo realizam saques de R$ 100,00, seria efetuado um total de R$ 1.000,00 de uma conta que só tem R$ 100,00 de saldo.

TECH
soluções em ti

Será que existe essa possibilidade de erro utilizando múltiplas threads? É isso que vamos ver..., pode ter certeza que se existir essa possibilidade o dono do banco que te contratou não vai gostar nenhum pouco do sistema desenvolvido.

De alguma forma precisamos bloquear a thread que está executando o processo de saque, pra que só ela realize o saque naquele momento e assim que ela terminar, ela desbloqueia para que outras threads possam executar o processo de saque.

TECH
soluções em ti


Thread A
Thread B
Bloqueia
Tem saldo?
Saca valor
Desbloqueia
Aguarda o Desbloqueio
da Thread A para continuar
a Execução
Bloqueia
Tem saldo?
Saca valor
Desbloqueia
Thread A continua
sua exceção

Vamos criar um pacote chamado br.com.treinacom.java.sincronizar, dentro desse pacote vamos criar uma classe java: ContaConjunta.

Ex.:

```
public class ContaConjunta{


        private int saldo = 100;
        public int getSaldo( ){
                return saldo;
        }
```

TECH
soluções em ti

```
public void sacar(int valor, String cliente){
        if(saldo >= valor){
                int saldoOriginal = saldo;
                System.out.println(cliente + " vai sacar");
        try{
                System.out.println(cliente + " esperando");
                Thread.sleep(1000); //aguardando processamento
        }catch(InterruptedException e){}
                saldo -= valor;
                String msg = cliente + " SACOU "+ valor + "[Saldo Original=" + saldoOriginal + ", Saldo Final=" + saldo + "]";
                System.out.println(msg);
```

```
        }else{
                System.out.println("Saldo insuficiente para "+cliente);
        }
}
```

Vamos disponibilizar essa conta para que várias pessoas possa utilizá-la. Vamos criar uma classe principal chamada ComprasEmFamilia.

Ex.:

```
public class ComprasEmFamilia implements Runnable{
        ContaConjunta conta = new ContaConjunta( );
```

TECH
soluções em ti

```
public static void main(String args[ ]){
        ComprasEmFamilia irAsCompras = new ComprasEmFamilia( );
        new Thread(irAsCompras, “Pai”).start( );
        new Thread(irAsCompras, “Mãe”).start( );
        new Thread(irAsCompras, “Filha”).start( );
        new Thread(irAsCompras, “Filho”).start( );
}

public void run( ){
        String cliente = Thread.currentThread( ).getName( );
for(int i = 0; i < 5; i++){
        conta.sacar(20, cliente);
```

TECH
soluções em ti

```
                if(conta.getSaldo( ) < 0){
                        System.out.println("Estourou!");
                }
        }
}
```

Sem a sincronização, com várias pessoas utilizando a mesma conta bancária, será possível sacar mesmo que a conta não tenha saldo. Com o synchronized, as threads não interferem uma com as outras deixando o sistema livre de possíveis falhas.

Ex.:

Public synchronized void sacar(int valor, String cliente){ }

TECH
soluções em ti

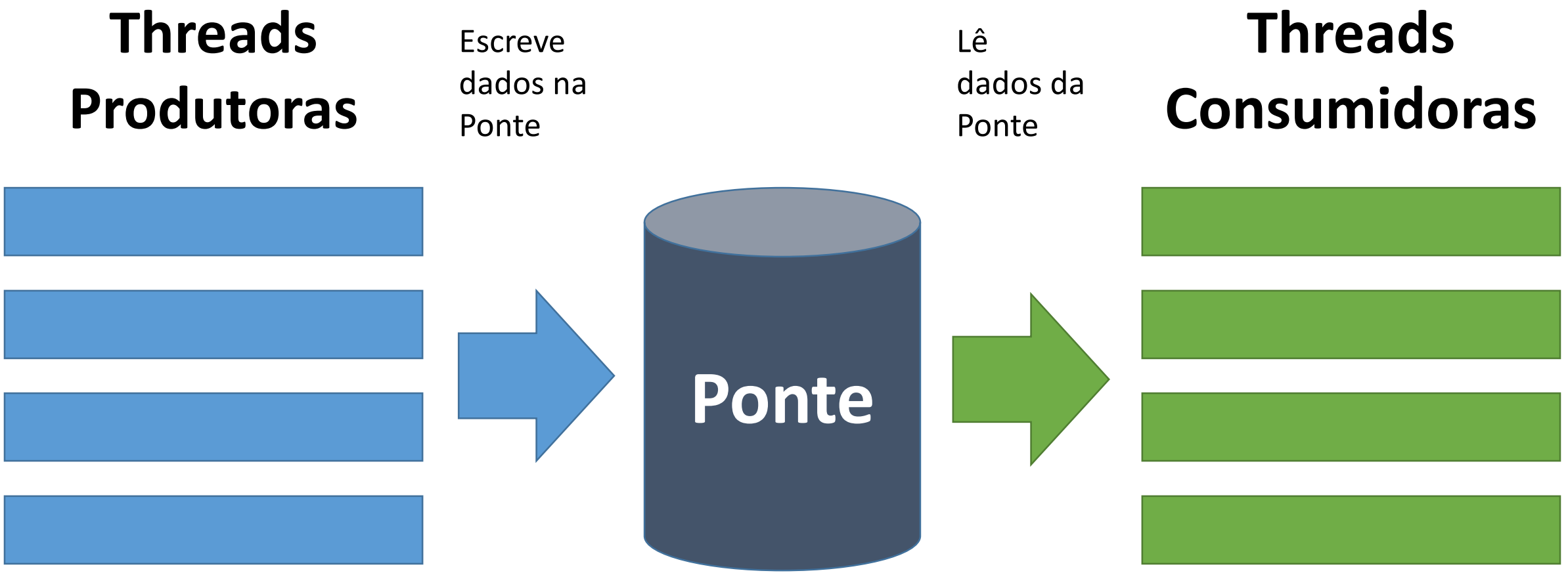
wait e notifyAll Threads

Vimos como sincronizar apenas um método utilizado por várias threads simultâneas, mas era apenas um método.

Imagina que você tenha um conjunto de threads que executam vários métodos, onde que para se executar um método, outro método diferente precisa ser bloqueado.

Aqui temos um nível de sincronização bem mais complexo, que aparece nos relacionamentos entre classes produtoras e classes consumidoras de informação.

TECH
soluções em ti



TECH
soluções em ti

A classe produtora armazena dados em um objeto que é compartilhado com as classes consumidoras. A classe consumidora lê esses dados compartilhados e os processa. Vamos implementar um relacionamento do tipo produtor/consumidor. Sem a sincronização e com sincronização.

Vamos criar a interface que representa a ponte entre os objetos produtores e consumidores:

```
public interface Ponte{ }
```

Dois métodos nessa interface:

```
public void set(int valor) throws InterruptedException;

public int get( ) throws InterruptedException;
```

TECH
soluções em ti

Agora que temos a nossa ponte, vamos construir a nossa classe produtora de informações. A nossa classe produtora precisa receber uma ponte que ela vai utilizar para compartilhar as informações que ela vai produzir. E essas classes produtoras também serão executadas em threads separadas.

Ex.:

```
public class Produtor implements Runnable{
        private Random random = new Random( );
        private Ponte ponte;
        public Produtor(Ponte ponte){
                this.ponte = ponte;
        }
```

TECH
soluções em ti

```
@Override
public void run( ){
        int total = 0;
        for(int i = 0; i < 5; i++){
                try{
                Thread.sleep(Random.nextInt(3000);
                total += i;
                ponte.set(i);
                System.out.println("\t" + total);
                }catch(InterruptedException e){ }
        }
}
```

TECH
soluções em ti

Agora vamos construir a nossa classe consumidora, será semelhante a nossa classe produtora. Ao invés de produzir informações, elas irão ler as informações, utilizando diversas threads.

Ex.:

```
public class Consumidor implements Runnable{
        private Random random = new Random( );
        private Ponte ponte;
        public Consumidor(Ponte ponte){
                this.ponte = ponte;
        }
```

TECH
soluções em ti

```
@Override
public void run( ){
        int total = 0;
        for(int i = 0; i < 5; i++){
                try{
                Thread.sleep(Random.nextInt(3000);
                total += ponte.get( );
                System.err.println(“\t” + total);
                }catch(InterruptedException e){ }
        }
}
```

TECH
soluções em ti

Agora temos a classe produtora, a consumidora e a nossa interface com a ponte de informações que precisa ser implementada.

Vamos criar uma classe: PonteNaoSincronizada

Ex.:

```
Public class PonteNaoSincronizada implements Ponte{
        private int valor = -1;
        public void set(int valor) throws InterruptedException{
                System.out.print(“Produziu “ + valor);
                this.valor = valor;
        }
        public int get( ) throws InterruptedException{
                System.err.print(“Consumiu “ + valor);
                return valor;
        }
```

TECH
soluções em ti

Agora precisamos da classe principal: PonteTeste

Ex.:

```
Public class PonteTeste{
        public static void main(String args[ ]){
                Ponte ponte = new PonteNaoSincronizada( );
                new Thread(new Produtor(ponte)).start( );
                new Thread(new Consumidor(ponte)).start( );
        }
}
```

Observe que as tarefas não estarão sincronizadas. Problema de integridade das informações.

TECH
soluções em ti

Vamos modificar as nossas threads de forma que quando o produtor gerar uma informação, o consumidor consuma em seguida, e o saldo total seja igual.

Vamos criar uma classe java: PonteSincronizada que também implementa a interface Ponte.

Ex.:

```
Public class PonteSincronizada implements Ponte{
        private int valor = -1;
        public synchronized void set(int valor) throws InterruptedException{
                System.out.print(“Produziu “ + valor);
                this.valor = valor;
        }
        public synchronized int get( ) throws InterruptedException{
                System.err.print(“Consumiu “ + valor);
                return valor;
        }
```

Ex.:

```
Public class PonteTeste{
        public static void main(String args[ ]){
                Ponte ponte = new PonteSincronizada( );
                new Thread(new Produtor(ponte)).start( );
                new Thread(new Consumidor(ponte)).start( );
        }
}
```

Precisamos desse novo nível de sincronização, precisamos que nossas threads esperem uma determinada condição para executar o método get e set.

TECH
soluções em ti

No caso de colocar um novo item na ponte, a condição que permite a operação prosseguir é que a ponte não esteja ocupada. E no caso de recuperar um novo item da ponte, a condição que permite que a operação prossiga é que a ponte não esteja desocupada.

Então se a condição for verdadeira, a operação pode seguir, senão, então a thread deve esperar até que retorna valor verdadeiro.

Ex.:

Na classe PonteSincronizada precisamos de um valor que verifica se a ponte está ou não ocupada.

```
private boolean ocupada = false;
//método set fazer essa alteração
while(ocupada){
        System.out.println("Ponte cheia. Produtor aguarda.");
        wait( );
}
```

TECH
soluções em ti

Método wait( ) , deixa a thread aguardando...

System.out.print(“Produziu “ + valor);

this.valor = valor;

Ocupada = true;

notifyAll( );

Método notifyAll( ), notifica as threads se o nosso estado mudou ou não.

//método get fazer essa alteração

```
while(!ocupada){
        System.out.println(“Ponte vazia. Consumidor aguarda”);
        wait( );
}
```

TECH
soluções em ti

```
System.err.print(“Consumiu “ + valor);
Ocupada = false;
notifyAll( );
return valor;
```

Agora temos uma classe multithread que executa de forma correta as operações sem que as informações se percam...

TECH
soluções em ti

Prosseguiremos no próximo slide...

Professor: Anderson Henrique

Programador nas Linguagens Java e PHP