



UNIVERSIDADE
ESTADUAL DE LONDRINA

Threads

Laboratório de Programação (5COP011)

Prof. Bruno Bogaz Zarpelão

Threads

- Nos computadores, sempre precisamos executar várias coisas ao mesmo tempo.
- Como fazer isso se não temos processadores suficientes para todos os programas em execução?
- Normalmente, o próprio sistema operacional cuida disso.

Threads

- Quando programamos, também precisamos, às vezes, executar tarefas em paralelo.
- Exemplo:
 - enquanto programa faz download de vídeo, queremos já ir assistindo o início do vídeo.
 - enquanto o programa gera um relatório, queremos mostrar uma barra de progresso.
- Vocês conseguiriam programar esses cenários com o que sabem até agora sobre Java?

Threads

- Para que o programa execute múltiplas tarefas simultaneamente, usamos o recurso das threads.
- O Java dá suporte a threads!

Threads

- Para trabalhar com threads em Java, podemos recorrer a duas soluções diferentes:
 - implementar a interface `Runnable`.
 - estender a classe `Thread`.
- Quando estendemos a classe `Thread`, herdamos todos os métodos dessa classe, até aqueles que não necessitamos.

Threads

- Quando implementamos a interface `Runnable`, SÓ devemos implementar o método `run()`.
- Qual é a melhor solução?

Exemplo

```
public class GeradorPDF implements Runnable {  
    public void run() {  
        //lógica para gerar o PDF.  
    }  
}
```

```
public class BarraDeProgresso implements Runnable {  
    public void run() {  
        //lógica para barra de progresso.  
    }  
}
```

Exemplo

```
public class MeuPrograma {  
  
    public static void main (String args[]) {  
  
        GeradorPDF g = new GeradorPDF();  
        Thread tPDF = new Thread(g);  
        tPDF.start();  
  
        BarraDeProgresso b = new BarraDeProgresso();  
        Thread tB = new Thread(b);  
        tB.start();  
  
    }  
}
```


Outros métodos interessantes

- Fazer a thread dormir:
 - `TimeUnit.SECONDS.sleep(10)` OU
 - `Thread.sleep(10000)`
- Em ambos, a thread vai dormir por 10 segundos.
- Esperar a execução da thread:
 - `t1.join()` ;
 - `t1` é um objeto qualquer da classe `Thread`.

Escalonamento e troca de contexto

- No computador, temos apenas um ou poucos processadores.
- Dessa forma, essas threads tem de ser escalonadas.
- O escalonador faz com que o processador fique alternando entre as threads.
- Em cada alternância, há uma troca de contexto.

Escalonamento e troca de contexto

- Nós não temos controle sobre o trabalho do escalonador.
- Vamos a um exemplo!

Problemas com a concorrência

- Imagine um sistema de banco:
 - milhares de clientes fazem transações todos os dias.
 - em um dado momento do dia, é necessário aplicar um percentual de rendimento a um conjunto de contas.
 - Isso pode gerar um problema de inconsistência!

Problemas com a concorrência

```
public class Conta {  
  
    private double saldo;  
  
    public void atualizar (double taxa) {  
        double saldoAtualizado = this.saldo*(1+taxa);  
        this.saldo = saldoAtualizado;  
    }  
  
    public void depositar(double valor) {  
        double novoSaldo = this.saldo + valor;  
        this.saldo = novoSaldo;  
    }  
  
}
```

Problemas com a concorrência

- Para solucionar esse problema, precisamos definir uma região crítica.
- A região crítica é uma região que só pode ser acessada por uma thread de cada vez.
- Em Java, usamos a palavra chave `synchronized` para definir essa região.

Problemas com a concorrência

```
public class Conta {  
  
    private double saldo;  
  
    public void atualizar (double taxa) {  
        synchronized(this){  
            double saldoAtualizado = this.saldo*(1+taxa);  
            this.saldo = saldoAtualizado;  
        }  
    }  
  
    public void depositar(double valor) {  
        synchronized(this){  
            double novoSaldo = this.saldo + valor;  
            this.saldo = novoSaldo;  
        }  
    }  
  
}
```

Problemas com a concorrência

- Alguns tipos de *collections* no Java não são *thread-safe*:
 - LinkedList, por exemplo;
 - Nesse caso, o programador deve se preocupar em criar as regiões críticas.
- Outros tipos são *thread-safe*:
 - Vector, por exemplo.
 - Programador não precisa se preocupar com a concorrência.

Problemas com a concorrência

- Vamos ver um exemplo maior!

Referências bibliográficas

- Apostila da Caelum FJ 11 – Java e Orientação a Objetos