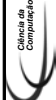


Programação em Java:
Prof. Riccioni
Teoria e Prática
UNIVALI São José
http://www.si.univali.br




Múltiplas Linhas de
Execução

riccioni@univali.br

Curso de Ciência da Computação - Matutino

Programação em Java:
Prof. Riccioni
Teoria e Prática
UNIVALI São José
http://www.si.univali.br



Multithreading


- Multitarefa:** a capacidade de ter mais de um programa funcionando no que parece ser simultâneo.

Por exemplo:

Você pode imprimir enquanto edita ou envia um fax. É claro que, a menos que você tenha uma máquina com vários processadores, o que ocorre realmente é que o sistema operacional está repartindo recursos para cada programa, dando a impressão de atividade paralela. Essa distribuição de recursos é possível pois, enquanto o usuário pode pensar que está mantendo o computador ocupado para, por exemplo, introduzir dados, a maior parte do tempo da CPU estará ociosa. (Afinal, um digitador rápido leva cerca de 1/20 segundos por caractere digitado.)

Curso de Ciência da Computação - Matutino

Programação em Java:
Prof. Riccioni
Teoria e Prática
UNIVALI São José
http://www.si.univali.br



Multithreading

A multitarefa pode ser realizada de duas maneiras:

- multitarefa preemptiva** - se o sistema operacional interromper o programa sem consulta-lo primeiro;

EX: QNX (uma versão do Unix para sistemas de tempo real) é preemptivo. (Embora seja mais difícil de implementar, a multitarefa preemptiva é muito mais eficiente. Com a multitarefa cooperativa, um programa de comportamento defeituoso pode travar tudo.)

- multitarefa cooperativa** (ou simplesmente não-preemptiva) - se os programas são interrompidos apenas quando estão querendo produzir controle.

EX: O Windows 3.1, 95, 98 e NT são sistemas multitarefa cooperativos

Curso de Ciência da Computação - Matutino

Multithreading

Os programas de múltiplas linhas de execução ampliam a idéia da multitarefa levando-a um nível mais abaixo: os programas individuais parecerão realizar várias tarefas ao mesmo tempo.

Normalmente, cada tarefa é chamada de linha de execução (*thread*) que é a abreviação de linha de execução de controle.

Diz-se que os programas que podem executar mais de uma linha de execução simultaneamente são multilinhas, ou têm múltiplas linhas de execução.

Considere cada linha de Execução como sendo executada em um contexto separado: os contextos fazem parecer que cada linha de execução possui sua própria CPU com registradores, memória e seu próprio código.

Multithreading

Qual a diferença entre múltiplos *processos* e múltiplas *linhas de execução*?

A diferença básica é que, enquanto cada processo tem um conjunto completo de variáveis próprias, as linhas de execução compartilham os mesmos dados.

As múltiplas linhas de execução são extremamente úteis na prática: por exemplo, um navegador deve tratar com vários hosts, abrir uma janela de correio eletrônico ou ver outra página, enquanto descarrega dados.

A própria linguagem de programação Java usa uma linha de execução para fazer coleta de lixo em segundo plano evitando assim o problema de gerenciar memória.

Os programas GUI têm uma linha de execução separada para reunir eventos da interface com o usuário do ambiente operacional hospedeiro.

Multithreading

- Há várias formas de implementação para atender a vários clientes simultaneamente.
- Mas o uso de threads torna a implementação bastante simples.
- Na API do Java, no próprio pacote `java.lang`, está a classe `Thread`, que implementa a seguinte interface:

```
interface Runnable {  
    void run(); ..... Ponto de entrada  
                        de uma thread.  
}
```

Programação em Java:
Prof. Riccioni
Teoria e Prática
UNIVALI São José
<http://www.ig.univali.br>

A Classe Thread (java.lang)

```
public class Thread implements Runnable {  
    //...  
  
    void run() {  
        // deve ser redefinido para  
        // implementar o comportamento desejado.  
    }  
  
    //...  
}
```

Curso de Ciência da Computação - Matutino

Programação em Java:
Prof. Riccioni
Teoria e Prática
UNIVALI São José
<http://www.ig.univali.br>

Servidor com Multithread

```
public class Servidor {  
    ServerSocket ss;  
  
    public Servidor()  
    {  
        ss = new ServerSocket(porta);  
  
        while(true) {  
            Socket conexao = ss.accept(); // aguarda conexão...  
            // Conectado!  
            ServidorThread st = new ServidorThread(conexao);  
            st.start();  
        }  
    }  
}
```

Curso de Ciência da Computação - Matutino

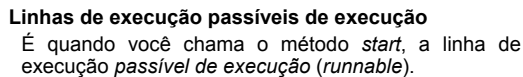
Programação em Java:
Prof. Riccioni
Teoria e Prática
UNIVALI São José
<http://www.ig.univali.br>

A Classe ServidorThread

```
class ServidorThread extends Thread {  
    private ObjectInputStream in;  
    private ObjectOutputStream out;  
  
    public ServidorThread(Socket cs) {  
        out = new ObjectOutputStream(cs.getOutputStream());  
        out.flush();  
        in = new ObjectInputStream(cs.getInputStream());  
    }  
  
    public void run() {  
        // ...implementação do protocolo da aplicação...  
    }  
}
```

Curso de Ciência da Computação - Matutino

Multithreading – Estado de uma linha de execução



Quando o código dentro da linha de execução começa a ser executado, a linha de execução está *em execução*. (Contudo, a documentação da plataforma Java não chama isso de um estado separado. Uma linha de execução em execução ainda está no estado passível de execução.)

Programação em Java:
Teoria e Prática

Prof. Ricconi

UNIVALI São José

http://www3.univali.br

Multithreading – Estado de uma linha de execução

Linhas de execução bloqueadas

Uma linha de execução entra no estado *bloqueada* quando uma das seguintes ações ocorre:

1. Alguém chama o método *sleep()* da linha de execução.
2. A linha de execução chama uma operação que está *bloqueando entrada/saída*; isto é, uma operação que não retornará ao seu chamador até que uma ou mais operações de entrada e saída estejam concluídas.
3. A linha de execução chama o método *wait()*.
4. A linha de execução tenta bloquear um objeto que está bloqueado por outra execução.
5. Alguém chama o método *suspend()* da linha de execução. Entretanto, esse método foi desaproado e você não deve chama-lo em seu código.

Curso de Ciência da Computação - Matutino

Programação em Java:
Teoria e Prática

Prof. Ricconi

UNIVALI São José

http://www3.univali.br

Multithreading – Estado de uma linha de execução

Linhas de execução mortas

Uma linha de execução torna-se morta por uma das duas razões:

Ela morre de morte natural, pois o método *run()* encerra normalmente;

Ela morre abruptamente, pois uma exceção não-capturada encerra o método *run()*.

Em particular, é possível matar uma linha de execução chamando seu método *stop()*. Esse método lança um objeto de erro *ThreadDeath*, que mata a linha de execução. Entretanto, o método *stop()* foi depreciado e você não deve chama-lo em seu código.

Uma linha de execução termina quando seu método *run()* retorna. Como o método *stop()* foi agora depreciado, não existe outro modo interno de terminar uma linha de execução.

Curso de Ciência da Computação - Matutino

Programação em Java:
Teoria e Prática

Prof. Ricconi

UNIVALI São José

http://www3.univali.br

Multithreading - Prioridades da Linha de Execução

Na linguagem de programação Java, toda linha de execução tem uma *prioridade*. Por definição, uma linha de execução herda a prioridade de sua linha de execução progenitora.

Você pode aumentar ou diminuir a prioridade de qualquer linha de execução com o método *setpriority*. Você pode configurar a prioridade para qualquer valor entre *MIN_PRIORITY* (configurada como 1 na classe *Thread*) e *MAX_PRIORITY* (configurada como 10). *NORM_PRIORITY* é configurada como 5.

Quando a agendador (scheduler) de linhas de execução tem uma chance de escolher uma nova linha de execução, geralmente ele escolhe a linha de execução de prioridade mais alta que é passível de execução.

Curso de Ciência da Computação - Matutino

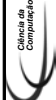
Se você der um clique no botão “Start”, cinco linhas de execução será ativadas com prioridade normal, animando cinco bolas pretas. Se você der um clique no botão “Express”, então ativará cinco bolas vermelhas, cuja linha de execução é executada em uma prioridade mais alta do que as bolas normais.

```
public BounceExpress ()
{....
    public void actionPerformed(ActionEvent evt)
    { for (int i = 0; i < 5; i++)
      { Ball b = new Ball(canvas, Color.black);
        b.setPriority(Thread.NORM_PRIORITY);
        b.start();
      }
    }
    public void actionPerformed(ActionEvent evt)
    { for (int i = 0; i < 5; i++)
      { Ball b = new Ball(canvas, Color.red);
        b.setPriority(Thread.NORM_PRIORITY + 2);
        b.start();
      }
    }
}
}....
}
```

Os arquivos:
Bounce.java;
BounceThread.java;
BounceExpress.java;
Lejame.doc.

São exemplos disponíveis em ftp, juntamente com a descrição dos exemplos.

Programação em Java:
Teoria e Prática
UNIVALI - São José
http://www.sig.univali.br




Curso de Ciência da Computação - Matutino

Multithreading - Sincronismo

Na maioria dos aplicativos com múltiplas linhas de execução (duas ou mais linhas) precisam compartilhar o acesso aos mesmos objetos.

O que acontece se duas linhas de execução têm acesso ao mesmo objeto e cada uma chama um método que modifica a estado do objeto? Conforme você poderia imaginar, as linhas de execução pisam nos pés umas das outras. Dependendo da ordem em que as dados foram acessados a resultado pode ser, de um objetos danificados. Tal situação é freqüentemente chamado de *condição de corrida* (race condition).

Programação em Java:
Teoria e Prática
UNIVALI - São José
http://www.sig.univali.br




Curso de Ciência da Computação - Matutino

Multithreading - Sincronismo

Para entendimento simulamos um banco com 10 contas. Geramos, aleatoriamente, transações que movimentam dinheiro entre essas contas. Existem 10 linhas de execução, uma para cada conta. Cada transação movimenta uma quantidade aleatória de dinheiro da conta atendida pela linha de execução, para outra conta aleatória.

O código da simulação é simples. Temos a classe Bank, com o método transfer. Esse método transfere alguma quantidade de dinheiro de uma conta para outra. Se a conta de origem não tiver dinheiro suficiente, então a chamada simplesmente retorna.

Programação em Java:
Teoria e Prática
UNIVALI - São José
http://www.sig.univali.br



Curso de Ciência da Computação - Matutino

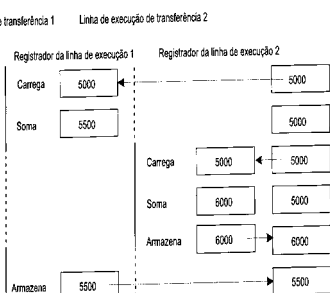
Multithreading - Sincronismo

Aqui está o código do método transfer da classe Bank.

```
public void transfer (mt from, mt to, int amount)
{
    if (accounts[from] < amount) return; accounts[from] -= amount;
    Accounts[to] += amount;
    ntransacts++;
    if (ntransacts % NTEST == 0) test();
}
```

Multithreading - Sincronismo

Linha de execução 1 Linha de execução 2 Linha de execução 1 Linha de execução 2



Curso de Ciência da Computação - Matutino

Multithreading - Sincronismo

```
public synchronized void transfer(int from, int to,int amount)
{
    {           if (accounts[from] < amount) return; accounts[from] -=
amount;
Accounts[to] += amount;
ntransacts++;
if (ntransacts % NTEST == 0) test();
}
```

Curso de Ciência da Computação - Matutino
