Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática

Introdução a Threads

NOTAS DE AULA

Linguagem de Programação II
2º semestre de 2015
Prof. Tomaz Mikio Sasaki
Versão 1.0

Introdução

- Atualmente a maioria dos sistemas operacionais permite que um programa execute múltiplas tarefas de forma aparentemente simultânea.
- Exemplo:
    - Um navegador web permite que o usuário inicie o download de um arquivo e continue navegando em outras páginas enquanto o download está sendo executado.

Referência

A referências para esta aula é o capítulo 14:

> HORSTMANN, C.S.; CORNELL, G. **Core Java, Volume I - Fundamentals.** Prentice Hall, 2012.

Observação: As notas de aula são material de apoio para estudo e não têm o objetivo de apresentar o assunto de maneira exaustiva. Não deixe de ler o material de referência da disciplina.

Para reduzir o número de linhas de código, os exemplos apresentados omitem propositalmente a importação das classes. As principais IDEs Java possuem recursos para auxiliar a inclusão das importações das classes.

Execução síncrona

Quando o programa possui uma única tarefa, dizemos que a execução é síncrona pois as instruções são executadas de forma sequencial de acordo com a ordem das instruções, e uma instrução é executada somente após o término da instrução anterior.

Exemplo 1 (execução síncrona)

A execução do programa abaixo apresenta primeiro os números e depois as letras, de acordo com a ordem das instruções dentro do método **main**:

```
public class Exemplo1 {

    public static void main(String[] args) {
        for (int i=0; i<10; i++) {
            System.out.print(i + " ");
        }
        System.out.println("FimNumeros");
        for (char c=65; c<75; c++) {
            System.out.print(c + " ");
        }
        System.out.println("FimLetras");
    }
}
```

Isto irá imprimir na tela do computador primeiro os números e depois as letras:

```
0 1 2 3 4 5 6 7 8 9 FimNumeros
A B C D E F G H I J FimLetras
```

Pausa entre a execução das instruções

É possível inserir uma pausa entre a execução das instruções utilizando o método **sleep** da classe **Thread**. Este é um método estático que recebe como parâmetro o número de milissegundos da pausa desejada

Exemplo 2 (com pausa entre as impressões)

```
public class Exemplo2 {

    public static void main(String[] args) throws InterruptedException {
        for (int i=0; i<10; i++) {
            System.out.print(i + " ");
            Thread.sleep(500);
        }
        System.out.println("FimNumeros");
        for (char c=65; c<75; c++) {
            System.out.print(c + " ");
            Thread.sleep(500);
        }
        System.out.println("FimLetras");
    }
}
```

O resultado na tela do computador será o mesmo que no exemplo anterior, só que com uma pausa de 500ms entre as impressões dos números/letras.

Exemplo 3 (chamando métodos)

```
class ImpressorNumeros {
  public void imprimir()
        throws InterruptedException {
    for (int i = 0; i < 10; i++) {
      System.out.print(i + " ");
      Thread.sleep(500);
    }
    System.out.println("FimNumeros");
  }
}

class ImpressorLetras {
  public void imprimir()
        throws InterruptedException {
    for (char c = 65; c < 75; c++) {
      System.out.print(c + " ");
      Thread.sleep(500);
    }
    System.out.println("FimLetras");
  }
}
```

```
public class Exemplo3 {
  public static void main(String[] a)
        throws InterruptedException {
    ImpressorNumeros impNum;
    impNum = new ImpressorNumeros();
    ImpressorLetras impLet;
    impLet = new ImpressorLetras();
    impNum.imprimir();
    impLet.imprimir();
  }
}
```

Multithreading

Uma aplicação *multithread* executa duas ou mais tarefas de forma aparentemente simultânea. Estas tarefas podem estar sendo executadas realmente de forma simultânea (por exemplo, se cada tarefa for atribuída a um *core* diferente do processador) ou estar compartilhando o tempo de execução de um único *core*, dando a impressão para o usuário de que a execução é simultânea.

Declaração de uma nova Thread em Java

Em Java podemos declarar uma nova *thread* em um programa:

1) Criando uma classe-filha de **Thread**.

2) Criando uma implementação da interface **Runnable** (que depois será instanciada e utilizada para instanciar uma **Thread**).

Exemplo 4 (declaração de nova thread derivando a classe Thread)

```
public class ThreadNumeros extends Thread {

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
            try { Thread.sleep(500); }
            catch(InterruptedException e) {}
        }
        System.out.println("FimNumeros");
    }
}
```

Exemplo 5 (implementação da interface Runnable)

```
public class RunnableLetras implements Runnable {

    @Override
    public void run() {
        for (char c = 65; c < 75; c++) {
            System.out.print(c + " ");
            try { Thread.sleep(500); }
            catch (InterruptedException e) {}
        }
        System.out.println("FimLetras");
    }

}
```

Execução da thread em Java

Para indicar que a *thread* pode ser executada, deve-se chamar o seu método **start**.

Exemplo 6 (execução das threads)

```
public class Exemplo6 {

    public static void main(String[] args) {
        Thread tNumeros = new ThreadNumeros();
        Runnable rLetras = new RunnableLetras();
        Thread tLetras = new Thread(rLetras);
        tNumeros.start();
        tLetras.start();
        System.out.println("FimMain");
    }
}
```

Resultado da execução (o resultado pode variar de computador para computador):

```
FimMain
0 A B 1 C 2 D 3 E 4 F 5 G 6 H 7 I 8 9 J FimLetras
FimNumeros
```

Observação 1: chamar o método run não cria uma nova Thread

start vs. run

Embora as instruções de cada tarefa estejam definidas no método **run**, chamar o método **run** só irá executar estas instruções na *thread* atual.

A tarefa só será executada em outra *thread* se chamarmos o método **start**.

Exemplo 7 (chamada do método run)

```
public class Exemplo7 {

    public static void main(String[] args) {
        Thread tNumeros = new ThreadNumeros();
        Runnable rLetras = new RunnableLetras();
        Thread tLetras = new Thread(rLetras);
        tNumeros.run();
        tLetras.run();
        System.out.println("FimMain");
    }
}
```

Como a execução é feita em uma única *thread*, o resultado da execução é:

```
0 1 2 3 4 5 6 7 8 9 FimNumeros
A B C D E F G H I J FimLetras
FimMain
```

Observação 2: um objeto pode ser compartilhado por duas ou mais threads

Exemplo 8

```
public class Impressor {
    public void imprimirNumeros() {
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
            try { Thread.sleep(500); }
            catch (InterruptedException e) {
            }
        }
        System.out.println("FimNumeros");
    }
    public void imprimirLetras() {
        for (char c = 65; c < 75; c++) {
            System.out.print(c + " ");
            try { Thread.sleep(500); }
            catch (InterruptedException e) {
            }
        }
        System.out.println("FimLetras");
    }
}
```

Exemplo 8 (cont.)

```
class RunImpNum implements Runnable {
  Impressor imp;
  public RunImpNum(Impressor imp) {
    this.imp = imp;
  }
  @Override
  public void run() {
    imp.imprimirNumeros();
  }
}

class RunImpLet implements Runnable {
  Impressor imp;
  public RunImpLet(Impressor imp) {
    this.imp = imp;
  }
  @Override
  public void run() {
    imp.imprimirLetras();
  }
}
```

```
public class Exemplo8 {
  public static void main(String[] a){
    Impressor imp = new Impressor();
    Runnable rNum;
    rNum = new RunImpNum(imp);
    Runnable rLet;
    rLet = new RunImpLet(imp);
    Thread tNum = new Thread(rNum);
    Thread tLet = new Thread(rLet);
    tNum.start();
    tLet.start();
    System.out.println("FimMain");
  }
}
```

Note que a mesma instância de **Impressor** é compartilhada pelas instâncias de **RunImpNum** e **RunImpLet**.

Observação 3:

Se o objeto compartilhado por duas ou mais threads possui um atributo que pode ser acessado simultaneamente por duas ou mais threads, é necessário controlar este acesso para evitar **erros de violação de acesso à memória** (assunto que será explorado melhor em outras disciplinas).

Bom estudo!