Modulo 12

Intodução a Multi Thread

Programação Orientada a Objetos

Java

(Rone Ilídio)

Introdução

- Java Concurrency
- Multi Thread
- Programação em paralelo: “o computador executa mais de uma coisa ao mesmo tempo”

Processos e Thread

- Processo
  - Contem um conjunto de execução básico completo: código e área privada de memória
  - A grosso modo: um programa em execução
- Thread
  - Partes de um processo que executam de forma independente
  - Requer menos recurso que um processo
  - Exemplo:
    - navegador de internet: baixa uma página enquanto exibe o resultado do que já foi baixado e executa os gifs (figuras animadas) e flashs contidos na página.
- Todo programa tem no mínimo uma thread (main thread) que tem a habilidade de criar outras threads

Objetos Threads

- Duas formas de executar threads
  - Criar uma instância de Thread (será utilizado)
  - Passar o controle da aplicação para um "executor"
- Como criar objetos Thread
  - Classe filha da interface Runnable
  - Classe filha da classe Thread

Interface Runnable

- Define o método run: contém o código executado pela thread
- Classe filha de Runnable

```
public class HelloRunnable implements
   Runnable{
    public void run() {
        System.out.println("Hello Runnable!");
    }
}
```

Interface Runnable

```
public class ExecutaRunnable {
    public static void main (String arg[]){
        Thread thread = new Thread
                                (new HelloRunnable());
         thread.start();
    }
}
```

Classe Thread

- Ela já implementa Runnable
- Sobrescreve o método run
- Exemplo

```
public class HelloThread extends Thread{
    public void run(){
        System.out.println("Hello Thread!");
      }
}
```

Classe Thread

```
public class ExecutaThread {
    public static void main(String args[]){
        HelloThread thread = new HelloThread();
        thread.start();
    }
}
```

Runnable versus Thread

- Vantagem de extends Thread
  - Thread possui métodos para manipulação de threads
- Vantagens de implements Runnable
  - Pode-se criar suá própria hierarquia de classes

Exemplo de Execução em Paralelo

```
public class ThreadParalelo extends Thread{
    String texto;
    public ThreadParalelo(String texto){
        this.texto = texto;
    }
    public void run(){
        for(int i = 0; i<1000; i++){
            System.out.println(i+":" +texto);
        }
    }
}
```

Exemplo de Execução em Paralelo

```
public class ExecutaParalelo {
        public static void main(String args[]){
            ThreadParalelo a = new ThreadParalelo("#######");
            ThreadParalelo b = new
                                        ThreadParalelo("@@@@@@@");
            a.start();
            b.start();
    }
}
```

Thread Dormindo

- Uma thread pode dormir por um tempo mínimo definido pelo programador
- Não garante-se que a Thread dormirá somente o tempo definido
- Manda dormir:
  - Thread.sleep(milissegundos)
  - Thread.sleep(milissegundos, nanosegundos)
- O método sleep throws InterruptedException

```
public class ThreadParalelo2 extends Thread{
    String texto;
    public ThreadParalelo2(String texto){
        this.texto = texto;
    }
    public void run(){
        for(int i = 1; i<=1000; i++){
            if(i%100 ==0){
                try {
                    System.out.println("Vai dormir!  ("+texto+")");
                    Thread.sleep(0,1);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            System.out.println(i+":" +texto);
        }
    }
}
```

```
public class ExecutaParalelo2 {
    public static void main(String args[]){
      ThreadParalelo2 a = new  ThreadParalelo2("#########");
     ThreadParalelo2 b = new
                                ThreadParalelo2("@@@@@@@@@");
       a.start();
       b.start();
    }
}
```

- Observação:

Para interromper uma thread, basta chamar return, ou chamar o método interrupt() do objeto.

Método Join

- Paralisa a execução de uma thread que chama para até que a thread cujo método foi chamado execute
- O método Join throws InterruptedException
- Exemplo: duas thread (main e t)

```
public class JoinThread extends Thread {
    public void run(){
        for(int i=0;i<1000;i++)
            System.out.println(i+" executar
                                        JoingThread!");
    }
}
```

```
public class ExecutaJoingThread {
public static void imprime(){
    for(int i=0;i<1000;i++)
        System.out.println(i+" Executar thread principal!");
}
public static void main(String[] args) {
    JoinThread t = new JoinThread();
    t.start();
    try{
        t.join();
    }
    catch(InterruptedException iex){
        iex.printStackTrace();
    }
    imprime();
    }
}
```

Sincronização

- Mecanismo no qual duas ou mais Threads compartilham variáveis
- Como realizar sincronização:
  - Métodos sincronizados
  - Comandos sincronizados
- Exemplo de erro:
    - Interferência entre threads: duas threads atuam sobre o mesmo dado de maneira diferente

Interferência entre threads

```
public class Armazena {
    private int t=100;
    private int v[] = new int[t];
    public void incremente(){
        for(int i=0;i<t;i++)
            v[i]++;
    }
    public void decremente(){
        for(int i=0;i<t;i++)
            v[i]--;
    }
    public void imprime(){
    for(int i=0;i<t;i++)
        System.out.println(v[i] + " ");
    }
}
```

```
public class ThreadInterferencia extends Thread{
    Armazena a;
    boolean incrementar;
    public ThreadInterferencia(Armazena a, boolean incrementar){
        this.a=a;
        this.incrementar = incrementar;
    }
    public void run(){
        for(int i=0; i<1000; i++)
            if(incrementar)
                a.incremente();
            else
                a.decremente();
        }
}
```

```
public class ExecutaInterferencia {
   public static void main(String[] args) {
      Armazena arm = new Armazena();
      ThreadInterferencia a = new ThreadInterferencia(arm, true);
      ThreadInterferencia b = new ThreadInterferencia(arm, false);
      a.start();
      b.start();
      arm.imprime();
   }
}
```

Métodos Sincronizados

- Possuem a palavra reservada **synchronized** na sua declaração
- Resultado:
  - Não pode ocorrer duas invocações do método no mesmo objeto
- Exemplo:
  - Mesma situação, mas os métodos incrementa e decrementa são declarados como synchronized

```
public class ArmazenaSincronizado {
    private int t=100;
    private int v[] = new int[t];
    public synchronized void incremente(){
        for(int i=0;i<t;i++)
        v[i]++;
    }
    public synchronized void decremente(){
        for(int i=0;i<t;i++)
        v[i]--;
    }
    public synchronized void imprime(){
       for(int i=0;i<t;i++)
        System.out.print(v[i] + " ");
    }
}
```

```
public class ThreadSincronizada extends Thread{
    ArmazenaSincronizado a;
    boolean incrementar;
    public ThreadSincronizada(ArmazenaSincronizado a,
                                        boolean incrementar){
        this.a=a;
        this.incrementar = incrementar;
    }
    public void run(){
        for(int i=0; i<1000; i++)
          if(incrementar)
            a.incremente();
          else
            a.decremente();
    }
}
```

```
public class ExecutaSincronizado{
    public static void main(String[] args) {
        ArmazenaSincronizado armazena = new
                                        ArmazenaSincronizado();
        ThreadSincronizada a = new ThreadSincronizada(armazena,
                                                      true);
        ThreadSincronizada b = new ThreadSincronizada(armazena,
                                                      false);

        a.start();
        b.start();

        armazena.imprime();
    }
}
```

Comandos Sincronizados

- Bloco de comandos que executa de forma atômica
- Não pode ser executado ao mesmo tempo em threads diferentes
- Sintaxe:

```
synchronized(this){
    ... Comandos sincronizados ...
}
```

```
public class ArmazenaComandoSincronizado{
    private int t=100;   private int v[] = new int[t];
    public void incremente(){
        synchronized(this){
            for(int i=0;i<t;i++)  v[i]++;
        }
    }
    public synchronized void decremente(){
        synchronized(this){
            for(int i=0;i<t;i++)  v[i]--;
        }
    }
    public synchronized void imprime(){
        synchronized(this){
           for(int i=0;i<t;i++) System.out.print(v[i] + " ");
        }
    }
}
```