Programação Orientada por Objectos com Java

Ademar Aguiar
www.fe.up.pt/~aaguiar
ademar.aguiar@fe.up.pt

Universidade do Porto
Faculdade de Engenharia
FEUP

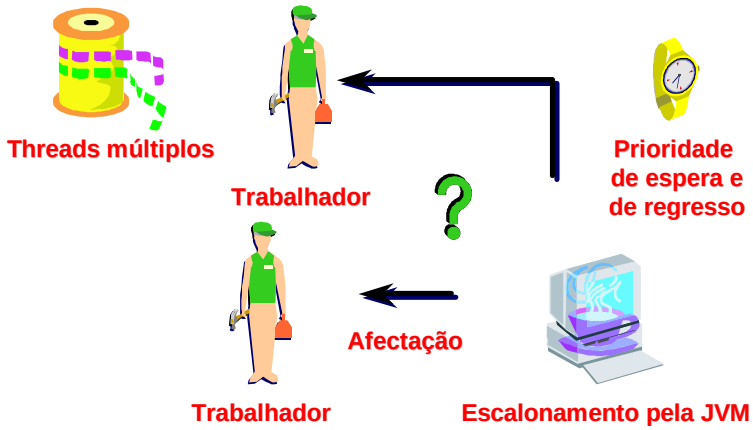
Multithreading

Objectivos

Ser capaz de:

- Explicar os conceitos básicos de 'multithreading'
- Criar threads múltiplos
- Aplicar a palavra reservada 'synchronized'
- Descrever o ciclo de vida de um thread
- Usar wait() e notifyAll()

Conteúdos



O que é um Thread?

- Um *thread* é uma execução sequencial de um programa.
- Cada programa tem pelo menos um *thread*.
- Cada *thread* tem a sua própria pilha, prioridade e conjunto de registos virtuais.
- Os *threads* subdividem o comportamento de um programa em subtarefas independentes.

Onde é que se usam Threads?

- São usados virtualmente em todos os computadores:
  - Em muitas aplicações (imprimir)
  - Em programas como browsers Internet
  - Em bases de dados
  - No sistema operativo
- Os *Threads* são normalmente usados sem serem percebidos pelo utilizador.

Porque se devem usar Threads?

- Para melhor aproveitar as capacidades do computador (utilizar o CPU enquanto se faz entrada/saída de dados)
- Maior produtividade para o utilizador final (uma interface mais interactiva)
- Vantagens para o programador (simplificar a lógica aplicacional)

Os Threads são algo de novo?

- Não!
- Evolução:
  - Utilizador único e sistemas em batch
  - Sistemas multi-processo
  - Sistemas multi-tarefa
  - Sistemas multi-thread
  - Sistemas multi-processador

A Classe Thread

- Mantém o estado de um thread
- Fornece diversos construtores
- Fornece diversos métodos
  - Thread.currentThread()
  - Thread.sleep()
  - Thread.setName()
  - Thread.isAlive()
- Escalonados pela JVM
- Utiliza o sistema operativo ou um package de threads

Exemplo de utilização de Thread

- Cada programa corre num thread.
- Adormecer um thread é uma técnica que permite que outros threads executem.

```
public static void main (args[] s) {
 System.out.println("I am thread "  +
           Thread.currentThread().getName());
 try {Thread.sleep(5000)}
 catch (InterruptedException e){}
…
}
```

Criação de um novo Thread

- 1. Criar a nova classe.
  - a. Definir uma subclasse de Thread.
  - b. Redefinir o seu método run().
- 2. Instanciar e executar o thread.
  - a. Criar uma instância da classe.
  - b. Invocar o método start().
- 3. O escalonador invoca o método run().

Criar a Classe

```
public class SleepingThread extends Thread {
 public void run () {
 Date startTime = new Date();
 try {Thread.currentThread().sleep
        ((int) (1000 * Math.random()));}
 catch (Exception es) {}
 long elapsedTime =
   new Date().getTime() - startTime.getTime();
 System.out.println(
   Thread.currentThread().getName() +
   ": I slept for " + elapsedTime +
   "milliseconds"); }}
```

Instanciar e Executar

```
public static void main(String[] args) {
 new SleepingThread().start();
 new SleepingThread().start();
 System.out.println("Started two threads…");
}
```

```
Started two threads…
Thread-1: I slept for 78 milliseconds
Thread-2: I slept for 428 milliseconds
```

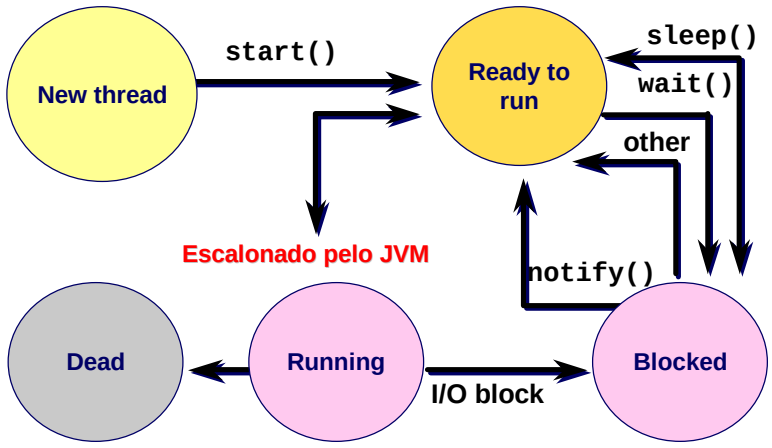

Acesso a Recursos partilhados

- Os dados podem ficar corrompidos se acedidos por vários threads:

```
public class BankAccount {
 private double balance;
 public void withdraw(double amt) {
  balance -= amt;
 }
```

  - **Utilizar a palavra *synchronized* para evitar conflitos de recursos partilhados.**

Ciclo de Vida de um Thread



Bloquear um Thread

- Utilizar wait() para bloquear o thread actual.
- O thread deve conter o monitor do objecto, ie, ser sincronizado (synchronized).
- O monitor será desbloqueado quando o wait() for invocado.
- O thread pode esperar indefinidamente ou por um período fixo de tempo.
- notifyAll() acorda todos os threads bloqueados.

Métodos synchronized

- Se um thread invoca um método synchronized, nenhum outro thread pode executar um método sincronizado no mesmo objecto até o primeiro thread completar a sua tarefa.

```
public class CheckOutFrame extends JFrame {
 public synchronized void updateElapsedTime() {
 …
 }
 public synchronized void
                computeAverageTime(Date old) {
 …
 }
```


Métodos synchronized

```
public class BankAccount {
 private double balance;
 public synchronized void withdraw(double amt) {
  balance -= amt;
 }
 public synchronized void deposit(double amt) {
  balance += amt;
 } …
```

Cuidado com synchronized!

- Cuidado para evitar deadlocks, evitando que todos os métodos sejam synchronized.

```
void FinishButton(ActionEvent e) {
 …
 finished = true;
 while(elapsedTime == 0) {}
 jText.setText("…");
}
```


Outra forma de criar Threads

- Implementar Runnable.
- Implementar o método run().
- Criar uma instância da classe (objecto alvo).
- Criar uma instância do Thread, passando o objecto alvo como um parâmetro.
- Invocar start() no objecto Thread.
- O escalonador invoca run() sobre o objecto alvo.

Exemplo com Runnable

```
public class MyApplet extends Applet
  implements Runnable{
 private Thread t;
 public void startApplet(){    // called by
browser
   t = new Thread(this); // creates a new
                         // runnable Thread
   t.start();            // starts the new Thread
 }
 public void run(){   // The new runnable Thread
  …                   // calls run() and the
                      // method runs in a
 } …                  // separate thread
```


Escalonamento e Prioridades

- Cada thread tem uma prioridade (1 to 10).
- O escalonamento é dependente do sistema operativo.
  - Um thread de alta prioridade pode interromper um de baixa-prioridade.
  - Threads de alta prioridade podem dominar o processador.
  - Threads de prioridade idêntica podem ser escalonados de forma circular.

Exemplo: Servidor de Ficheiros

```
public class FileServer extends Thread {
    public static void main(String[] argv) {
        ServerSocket s;
        try {
            s = new ServerSocket(1234, 10);
        }catch (IOException e) {
            System.err.println("Unable to create socket");
            e.printStackTrace();
            return;
        }
        try {
            while (true) {
                new FileServer(s.accept());
            }
        }catch (IOException e) {
        }
    }
    private Socket socket;
    FileServer(Socket s) {
        socket = s;
        start();
    }
}
```


Exemplo: Servidor de Ficheiros...

```
  public void run() {
        InputStream in;
        String fileName = "";
        PrintStream out = null;
        FileInputStream f;
        try {
            in = socket.getInputStream();
            out = new PrintStream(socket.getOutputStream());
            fileName = new DataInputStream(in).readLine();
            f = new FileInputStream(fileName);
        }catch (IOException e) {
            if (out != null)
              out.print("Bad:"+fileName+"\n");
            out.close();
            try {
                socket.close();
            }catch (IOException ie) {
            }
            return;
        }
        out.print("Good:\n");
        // send contents of file to client.
}
```

Fim