

Gestão de Tarefas em Java

Fundamentos de Sistemas Operativos

Licenciatura em Engenharia Informática e Multimédia

Carlos Gonçalves

Versão 1.1 — Fevereiro 2019

Índice

Lista de Figuras	V
Lista de Tabelas	VII
Lista de Listagens	IX
Lista de Acrónimos	1
1 Notas Introdutórias	1
2 Classe Thread	3
2.1 Métodos da classe Thread	3
3 Interface Runnable	7
4 Exemplos de Criação de Tarefas	9
4.1 Por derivação da classe Thread	9
4.2 Por implementação da interface Runnable	10
4.3 Definição do nome — Derivação da classe Thread	12
4.4 Definição do nome — Implementação da interface Runnable	13
4.5 Sincronização utilizando o método join	15
4.6 Sincronização entre tarefas	16
5 Tarefas e Modo Gráfico <i>Swing</i>	23
Bibliografia	28

Lista de Figuras

3.1	Diagrama UML da classe Thread e da interface Runnable	8
4.1	Resultado da execução da versão 1	10
4.2	Resultado da execução da versão 2	12
4.3	Resultado da execução da versão 3	13
4.4	Resultado da execução da versão 4	14
4.5	Resultado da execução da versão 5	16
4.6	Resultado da execução da versão 6	18
4.7	Diagrama UML da interface ISync e das implementações associadas	19
5.1	Diagrama UML da aplicação desenvolvida	23
5.2	Aplicação <i>swing</i> desenvolvida — Comportamento indefinido	24
5.3	Aplicação <i>swing</i> desenvolvida — Comportamento correto	24
5.4	Resultado da invocação do método <code>invokeLater</code>	27

Lista de Tabelas

2.1	Principais métodos da classe Thread	3
-----	---	---

Versão 1.1

Lista de Listagens

2.1	Utilização do método <code>run</code>	4
2.2	Utilização do método <code>join</code>	4
2.3	Especificação do nome da tarefa	5
2.4	Utilização do método <code>Thread.sleep</code>	5
2.5	Utilização do método <code>Thread.currentThread</code>	5
3.1	Criação de tarefa utilizando <code>Runnable</code>	7
4.1	Versão 1 — Por derivação da classe <code>Thread</code>	9
4.2	Versão 2 — Por implementação da interface <code>Runnable</code>	10
4.3	Versão 3 — Definição do nome da tarefa — Classe <code>Thread</code>	12
4.4	Versão 4 — Definição do nome da tarefa — Interface <code>Runnable</code>	13
4.5	Versão 5 — Sincronização utilizando o método <code>join</code>	15
4.6	Versão 6 — Sincronização entre tarefas	16
4.7	Definição da interface <code>ISync</code>	19
4.8	Implementação da interface <code>ISync</code> — Função <code>TestAndSet</code>	19
4.9	Implementação da interface <code>ISync</code> — <code>Semaphore</code>	20
4.10	Implementação da interface <code>ISync</code> — Monitores Java	20
5.1	Manipulação de componentes <i>Swing</i> por tarefas do utilizador — Versão 1 . . .	25
5.2	Manipulação de componentes <i>Swing</i> por tarefas do utilizador — Versão 2 . . .	25
5.3	Manipulação de componentes <i>Swing</i> por tarefas do utilizador — Versão 3 . . .	26
5.4	Manipulação de componentes <i>Swing</i> por tarefas do utilizador — Versão 4 . . .	27

Capítulo 1

Notas Introdutórias

Este texto pretende ser um complemento à documentação fornecida no contexto da Unidade Curricular (UC) de Fundamentos de Sistemas Operativos (FSO) do curso de Licenciatura em Informática e Multimédia, e não pode ser visto como uma sebenta.

O objetivo deste texto é completar a informação disponibilizada nas folhas da UC de FSO [1] com pormenores específicos da gestão e criação de tarefas (*threads*) no contexto da linguagem de programação Java.

No contexto deste texto são utilizadas as seguintes definições:

- **Ficheiro** — conjunto de *bytes*;
- **Programa** — ficheiro que contém um conjunto de instruções prontas a ser executadas por um CPU (*Central Processing Unit* ou Unidade Central de Processamento);
- **Processo** (ou **processo nativo**) — programa em execução.

Além deste capítulo introdutório este texto contém mais 3 capítulos: Capítulo 2 onde se apresenta a classe `Thread`; Capítulo 3 onde se apresenta a interface `Runnable`; Capítulo 4 onde se apresentam alguns exemplos que exemplificam a criação de tarefas e sincronização entre tarefas. O Capítulo 5 apresenta os procedimentos que devem ser seguidos em aplicações que envolvam a manipulação de componentes gráficos *Swing* por parte de tarefas Java que não a `EventDispatchThread`.

Capítulo 2

Classe Thread

A linguagem Java disponibiliza de raiz mecanismos para instanciar tarefas. Sempre que o sistema nativo suportar o conceito de tarefa (também designadas de processos leves ou *threads* na designação em inglês) as tarefas criadas no contexto de uma máquina virtual Java são mapeadas em tarefas do sistema operativo. Nas situações em que o sistema nativo não suporta o conceito de tarefa a máquina virtual Java implementa internamente este conceito. Do ponto de vista do programador não existe nenhuma diferença. Do ponto de vista de um programa Java uma tarefa é representada por uma instância da classe Thread [2] ou por uma classe derivada Thread. A tabela 2.1 apresenta os principais métodos disponibilizados na classe Thread.

Tabela 2.1: Principais métodos da classe Thread

Método
<code>start()</code>
<code>run()</code>
<code>join()</code>
<code>getName()</code>
<code>setName()</code>
<code>Thread.sleep()</code>
<code>Thread.currentThread()</code>

2.1 Métodos da classe Thread

start

O método `start` inicia a execução de uma nova tarefa. Ao invocar o método `start` é criada uma nova tarefa, que se executa em concorrência com as restantes tarefas do processo. As instruções executadas pela nova tarefa correspondem ao conteúdo do método `run`. A listagem 2.1 exemplifica a utilização do método `run`.

Listing 2.1: Utilização do método run

```
1    ...
2    class MyThread extends Thread {
3        ...
4        public void run() {
5            ...
6        }
7    }
8    ...
9    Thread th = new MyThread(...);
10   th.start();
11   ...
```

Neste exemplo a execução da linha 10 inicia uma nova tarefa que começa a cumprir as instruções associadas ao método run da classe MyThread. Quando o método run terminar a tarefa termina a sua execução.

run

O método run representa as instruções que serão executadas pela nova tarefa. Caso a tarefa necessite de argumentos os mesmos devem ser passados como argumentos no construtor da classe associada à tarefa.

join

O método join permite esperar em modo passivo, sem gastar CPU, pela terminação da tarefa sobre a qual se invoca o método. A listagem 2.2 exemplifica a utilização do método join.

Listing 2.2: Utilização do método join

```
1    ...
2    Thread th = new MyThread(...);
3    th.start();
4    ...
5    th.join();
6    ...
```

Este método pode ser utilizado em situações em que uma tarefa necessita de se sincronizar com a terminação de outra tarefa.

getName e setName

Os métodos `getName` e `setName` permitem, respectivamente, obter ou modificar o nome da tarefa. Por omissão todas as tarefas têm um nome da forma `Thread-x` onde `x` é um valor inteiro que representa o número da tarefa (0, 1, 2, etc.). O nome das tarefas também pode ser especificado aquando da construção do objecto que está associado à classe que suporta a tarefa tal como se exemplifica na listagem 2.3.

Listing 2.3: Especificação do nome da tarefa

```
1    ...
2    public class MyThread extends Thread {
3        public MyThread(String name) {
4            super( name );
5            ...
6        }
7        ...
8    }
9    ...
```

Thread.sleep

O método estático `sleep` permite fazer uma pausa durante um dado tempo (argumento do método `sleep`). O valor da pausa é expresso em milisegundos. A listagem 2.4 exemplifica a utilização do método `Thread.sleep`.

Listing 2.4: Utilização do método `Thread.sleep`

```
1    ...
2    Thread.sleep( 250 );
3    ...
```

Thread.currentThread

O método estático `currentThread` permite obter, em qualquer ponto de um programa Java, uma referência para um objecto do tipo `Thread` que representa a tarefa Java que está a executar o código nesse ponto do programa. A listagem 2.5 exemplifica a utilização do método `Thread.currentThread`.

Listing 2.5: Utilização do método `Thread.currentThread`

```
1  ...
2  System.out.printf( "[%s]...", Thread.currentThread().
   ↪getName() );
3  ...
```

Versão 1.1

Capítulo 3

Interface Runnable

A linguagem Java não permite que uma classe derive em simultâneo de mais do que um objecto. Assim, não é possível ter uma classe C2 a derivar de uma classe C1 ao mesmo tempo que deriva da classe Thread. Nestas situações como alternativa é possível definir a classe C1 como uma classe que implementa a interface Runnable e posteriormente criar uma tarefa utilizando uma instância de Runnable [3] tal como se apresenta na listagem 3.1.

Listing 3.1: Criação de tarefa utilizando Runnable

```
1    ...
2    public class MyRunnable implements Runnable {
3        public void run() {
4            ...
5        }
6        ...
7    }
8    ...
9    Thread th = Thread( new MyRunnable(...) );
10   r.start();
11   ...
```

Na figura 3.1 apresenta-se um diagrama simplificado UML (*Unified Modeling Language*) que relaciona a classe Thread e a interface Runnable.

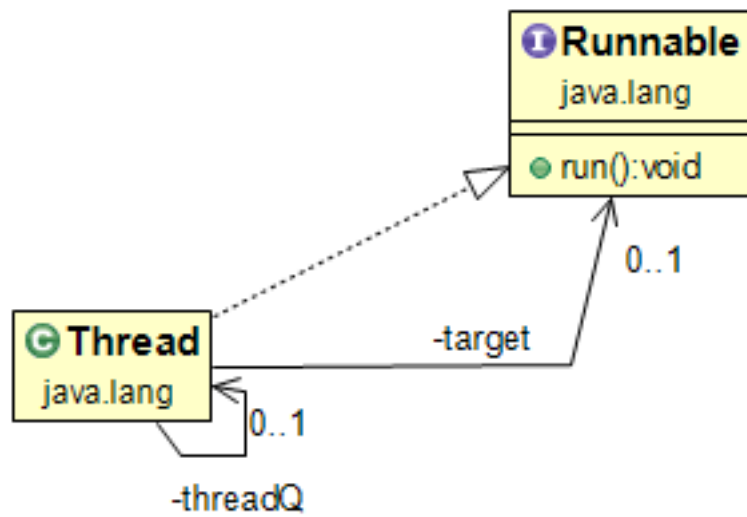


Figura 3.1: Diagrama UML da classe Thread e da interface Runnable

Capítulo 4

Exemplos de Criação de Tarefas

Neste capítulo apresentam-se alguns exemplos de criação de tarefas. Na maioria dos exemplos apresentados de seguida pretende-se escrever na consola a mensagem «*Hello world*» sendo que as palavras «*Hello*» e «*world*» são escritas por tarefas independentes. As várias versões serão progressivamente modificadas/melhoradas de modo a se conseguir garantir que a palavra «*world*» só é apresentada depois da palavra «*Hello*».

4.1 Por derivação da classe Thread

A listagem 4.1 exemplifica a criação de tarefas onde a classe que suporta a tarefa deriva da classe base Thread.

Listing 4.1: Versão 1 — Por derivação da classe Thread

```
1 public class HelloWorldVer01 extends Thread {
2
3     private String name;
4
5     public HelloWorldVer01(String name) {
6         this.name = name;
7     }
8
9     public void run() {
10         try {
11             Thread.sleep( (new Random()).nextInt( 250 ) );
12             System.out.print( this.name );
13         }
14         catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18 }
```

```

16     }
17 }
18
19 public static void main(String[] args) {
20     Thread thHello, thWorld;
21
22     thHello = new HelloWorldVer01( "Hello" );
23     thWorld = new HelloWorldVer01( " world.\n" );
24
25     thHello.start();
26     thWorld.start();
27 }
28 }

```

A figura 4.1 apresenta o resultado da execução da versão 1. Tal como se pode observar a ordem com que as palavras «Hello» e «world» aparecem não é determinística.

```

C:\WINDOWS\system32\cmd.exe
> java -cp bin fso.guioes.threads.HelloWorldVer01
world.
Hello
> java -cp bin fso.guioes.threads.HelloWorldVer01
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer01
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer01
world.
Hello
>

```

Figura 4.1: Resultado da execução da versão 1

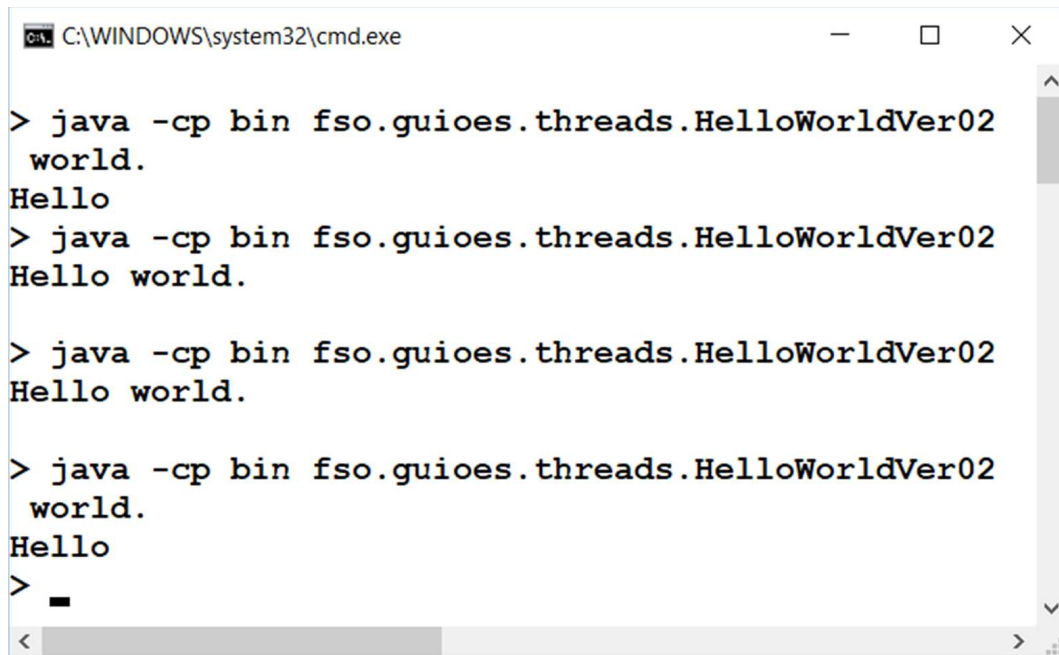
4.2 Por implementação da interface Runnable

A listagem 4.2 exemplifica a criação de tarefas onde a classe que suporta a tarefa implementa a interface Runnable.

Listing 4.2: Versão 2 — Por implementação da interface Runnable

```
1 public class HelloWorldVer02 implements Runnable {
2
3     private String name;
4
5     public HelloWorldVer02(String name) {
6         this.name = name;
7     }
8
9     public void run() {
10         try {
11             Thread.sleep( (new Random()).nextInt( 250 ) );
12             System.out.print( this.name );
13         }
14         catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18
19     public static void main(String[] args) {
20         Runnable rHello, rWorld;
21         Thread thHello, thWorld;
22
23         rHello = new HelloWorldVer02( "Hello" );
24         rWorld = new HelloWorldVer02( " world.\n" );
25
26         thHello = new Thread( rHello );
27         thWorld = new Thread( rWorld );
28
29         thHello.start();
30         thWorld.start();
31     }
32 }
```

A figura 4.2 apresenta o resultado da execução da versão 2. Tal como para a versão 1 a ordem com que as palavras «Hello» e «world» aparecem não é determinística.



```
C:\WINDOWS\system32\cmd.exe

> java -cp bin fso.guioes.threads.HelloWorldVer02
world.
Hello
> java -cp bin fso.guioes.threads.HelloWorldVer02
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer02
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer02
world.
Hello
> _
```

Figura 4.2: Resultado da execução da versão 2

4.3 Definição do nome — Derivação da classe Thread

A listagem 4.3 exemplifica a criação de tarefas e a redefinição do seu nome quando a classe que suporta a tarefa deriva da classe Thread.

Listing 4.3: Versão 3 — Definição do nome da tarefa — Classe Thread

```
1 public class HelloWorldVer03 extends Thread {
2
3     public HelloWorldVer03(String name) {
4         super( name );
5     }
6
7     public HelloWorldVer03() {
8     }
9
10    public void run() {
11        System.out.printf(
12            "[%s] Running...\n",
13            this.getName() );
14    }
15 }
```

```

16 public static void main(String[] args) {
17     Thread th1, th2, th3;
18
19     th1 = new HelloWorldVer03();
20     th1.start();
21
22     th2 = new HelloWorldVer03();
23     th2.setName( "Hello2");
24     th2.start();
25
26     th3 = new HelloWorldVer03( "Hello3" );
27     th3.start();
28 }
29 }

```

O resultado da execução da versão 3 é apresentado na figura 4.3.



```

C:\WINDOWS\system32\cmd.exe

> java -cp bin fso.guioes.threads.HelloWorldVer03
[Thread-0] Running...
[Hello3] Running...
[Hello2] Running...

```

Figura 4.3: Resultado da execução da versão 3

4.4 Definição do nome — Implementação da interface Runnable

A listagem 4.4 exemplifica a criação de tarefas e a redefinição do seu nome quando a classe que suporta a tarefa implementa a interface Runnable.

Listing 4.4: Versão 4 — Definição do nome da tarefa — Interface Runnable

```

1 public class HelloWorldVer04 implements Runnable {
2
3     public HelloWorldVer04() {
4     }
5
6     public void run() {

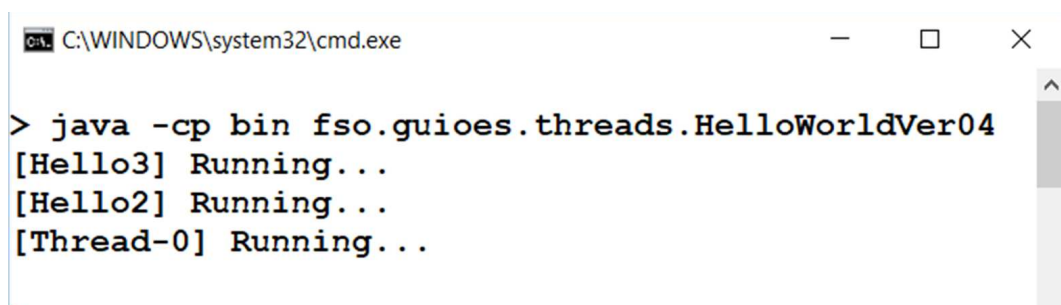
```

```

7      System.out.printf(
8          "[%s] Running...\n",
9          Thread.currentThread().getName() );
10 }
11
12 public static void main(String[] args) {
13     Runnable r1, r2, r3;
14     Thread th1, th2, th3;
15
16     r1 = new HelloWorldVer04();
17     r2 = new HelloWorldVer04();
18     r3 = new HelloWorldVer04();
19
20     th1 = new Thread( r1 );
21     th1.start();
22
23     th2 = new Thread( r2 );
24     th2.setName( "Hello2" );
25     th2.start();
26
27     th3 = new Thread( r3, "Hello3" );
28     th3.start();
29 }
30 }

```

O resultado da execução da versão 4 é apresentado na figura 4.4.



```

C:\WINDOWS\system32\cmd.exe
> java -cp bin fso.guioes.threads.HelloWorldVer04
[Hello3] Running...
[Hello2] Running...
[Thread-0] Running...

```

Figura 4.4: Resultado da execução da versão 4

4.5 Sincronização utilizando o método `join`

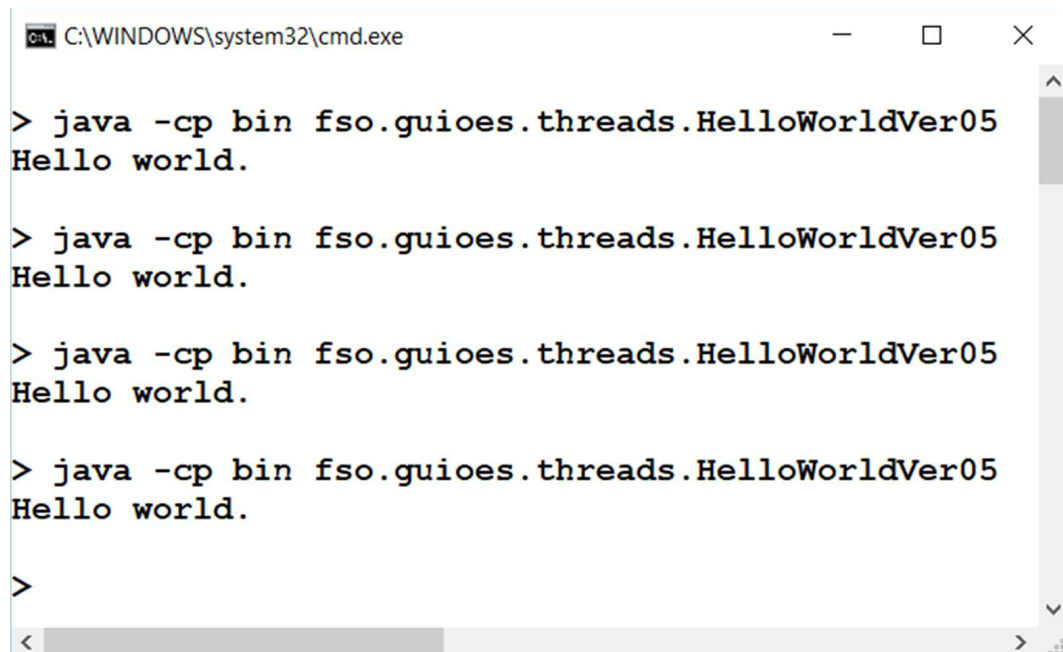
A listagem 4.5 exemplifica a criação e sincronização de tarefas utilizando o método `join`. Esta versão garante que a palavra «*world*» só é escrita depois da palavra «*Hello*». No entanto, apenas é possível escrever a sequência «*Hello world*» uma única vez.

Listing 4.5: Versão 5 — Sincronização utilizando o método `join`

```
1 public class HelloWorldVer05 extends Thread {
2
3     private String name;
4
5     public HelloWorldVer05(String name) {
6         this.name = name;
7     }
8
9     public void run() {
10         try {
11             Thread.sleep( (new Random()).nextInt( 250 ) );
12             System.out.print( this.name );
13         }
14         catch (Exception e) {
15             e.printStackTrace();
16         }
17     }
18
19     public static void main(String[] args) {
20         try {
21             Thread thHello, thWorld;
22
23             thHello = new HelloWorldVer05( "Hello" );
24             thWorld = new HelloWorldVer05( " world.\n" );
25
26             thHello.start();
27             thHello.join();
28             thWorld.start();
29         }
30         catch (InterruptedException e) {
31             e.printStackTrace();
```

```
32     }
33 }
34 }
```

O resultado da execução da versão 5 é apresentado na figura 4.5. Tal como se pode observar a palavra «world» aparece sempre depois da palavra «Hello».



```
C:\WINDOWS\system32\cmd.exe

> java -cp bin fso.guioes.threads.HelloWorldVer05
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer05
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer05
Hello world.

> java -cp bin fso.guioes.threads.HelloWorldVer05
Hello world.

>
```

Figura 4.5: Resultado da execução da versão 5

4.6 Sincronização entre tarefas

A listagem 4.6 exemplifica a criação e sincronização de tarefas utilizando diferentes mecanismos de sincronização, nomeadamente: i) Função `TestAndSet` (disponibilizada por exemplo na classe `AtomicBoolean` [4]); ii) Semáforos; iii) Monitores. Esta versão garante que a palavra «world» só é escrita depois da palavra «Hello» e que o resultado não depende da terminação das tarefas.

Listing 4.6: Versão 6 — Sincronização entre tarefas

```
1 public class HelloWorldVer06 extends Thread {
2     private final int Iterations = 5;
3
4     private String name;
5
6     private ISync previous, next;
```

```

7
8 public HelloWorldVer06(String name, ISync previous, ISync
   ↪next) {
9     this.name = name;
10
11     this.previous = previous;
12     this.next = next;
13 }
14
15 public void run() {
16     for(int idxIter=0; idxIter< this.Iterations; ++idxIter) {
17         try {
18             this.previous.syncWait();
19
20             Thread.sleep( (new Random()).nextInt( 250 ) );
21             System.out.print( this.name );
22
23             this.next.syncSignal();
24         }
25         catch (Exception e) {
26             e.printStackTrace();
27         }
28     }
29 }
30
31 public static void main(String[] args) {
32     try {
33         ISync syncHelloToWorld, syncWorldToHello;
34
35         //syncHelloToWorld = new ...
36         //syncHelloToWorld = new ...
37
38         Thread thHello, thWorld;
39
40         thHello = new HelloWorldVer06(
41             "Hello",
42             syncHelloToWorld,

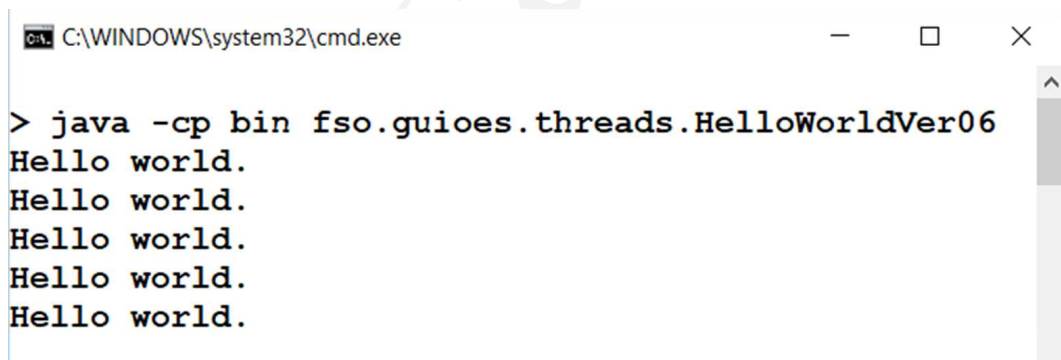
```

```

43         syncWorldToHello);
44     thWorld = new HelloWorldVer06(
45         " world.\n",
46         syncWorldToHello,
47         syncHelloToWorld);
48
49     thHello.start();
50     thWorld.start();
51
52     syncHelloToWorld.syncSignal();
53 }
54 catch (InterruptedException e) {
55     e.printStackTrace();
56 }
57 }
58 }

```

O resultado da execução da versão 6 é apresentado na figura 4.6. Tal como se pode observar a palavra «world» aparece sempre depois da palavra «Hello».



```

C:\WINDOWS\system32\cmd.exe
> java -cp bin fso.guioes.threads.HelloWorldVer06
Hello world.
Hello world.
Hello world.
Hello world.
Hello world.

```

Figura 4.6: Resultado da execução da versão 6

Na figura 4.7 apresenta-se o diagrama UML da interface do mecanismo de sincronização ISync (apresentada na listagem 4.7) e das implementações do mecanismo de sincronização ISync utilizando: i) As funções TestAndSet (listagem 4.8); ii) Semáforos Java, com base na classe Semaphore disponível no *package* java.util.concurrent [5] (listagem 4.9); iii) Utilizando o mecanismo de sincronização Monitor disponível na linguagem Java (listagem 4.10).

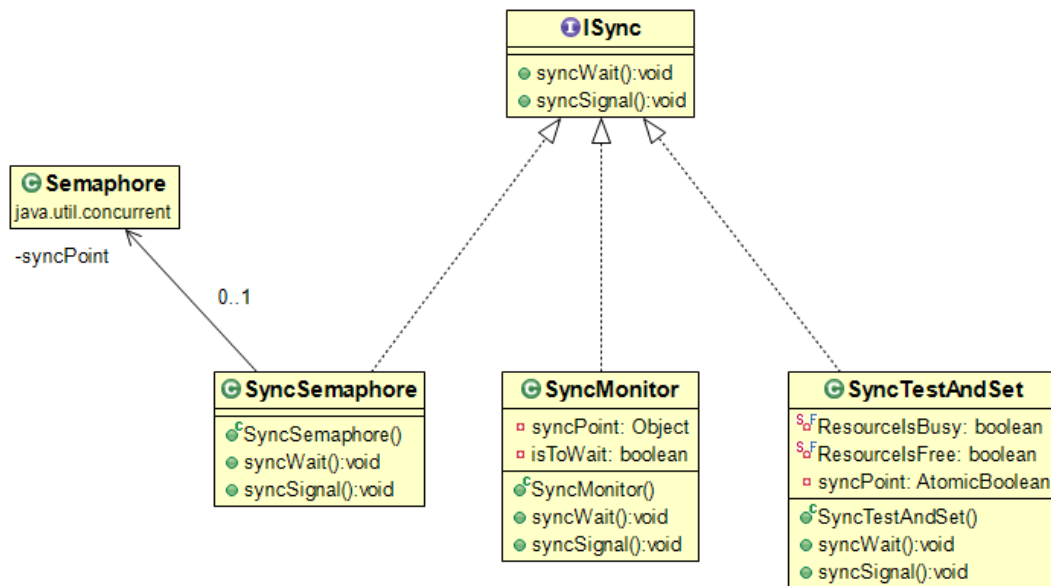


Figura 4.7: Diagrama UML da interface ISync e das implementações associadas

Listing 4.7: Definição da interface ISync

```

1 public interface ISync {
2     public void syncWait() throws InterruptedException;
3
4     public void syncSignal() throws InterruptedException;
5 }
  
```

Listing 4.8: Implementação da interface ISync — Função TestAndSet

```

1 import java.util.concurrent.atomic.AtomicBoolean;
2
3 public class SyncTestAndSet implements ISync {
4
5     private static final boolean ResourceIsBusy = true;
6     private static final boolean ResourceIsFree = false;
7
8     private AtomicBoolean syncPoint;
9
10    public SyncTestAndSet() {
11        this.syncPoint = new AtomicBoolean( ResourceIsBusy );
12    }
13
14    @Override
  
```

```

15 public void syncWait() throws InterruptedException {
16     while ( this.syncPoint.compareAndSet(
17         ResourceIsFree , ResourceIsBusy)==false )
18         ;
19 }
20
21 @Override
22 public void syncSignal() throws InterruptedException {
23     this.syncPoint.set( false );
24 }
25 }

```

Listing 4.9: Implementação da interface ISync — Semaphore

```

1 import java.util.concurrent.Semaphore;
2
3 public class SyncSemaphore implements ISync {
4
5     private Semaphore syncPoint;
6
7     public SyncSemaphore() {
8         this.syncPoint = new Semaphore(0);
9     }
10
11     @Override
12     public void syncWait() throws InterruptedException {
13         this.syncPoint.acquire();
14     }
15
16     @Override
17     public void syncSignal() throws InterruptedException {
18         this.syncPoint.release();
19     }
20 }

```

Listing 4.10: Implementação da interface ISync — Monitores Java

```

1 public class SyncMonitor implements ISync {
2

```

```

3  private Object syncPoint;
4  private boolean isToWait;
5
6  public SyncMonitor() {
7      this.syncPoint = new Object();
8      this.isToWait = true;
9  }
10
11  @Override
12  public void syncWait() throws InterruptedException {
13      synchronized ( this.syncPoint ) {
14          while ( this.isToWait==true ) {
15              this.syncPoint.wait();
16          }
17          this.isToWait=true;
18      }
19  }
20
21  @Override
22  public void syncSignal() throws InterruptedException {
23      synchronized (this.syncPoint ) {
24          this.isToWait = false;
25          this.syncPoint.notifyAll();
26      }
27  }
28 }

```


Capítulo 5

Tarefas e Modo Gráfico *Swing*

Dado que o a maioria dos componentes gráficos disponibilizados no *package* `javax.swing` não são *thread safe*, isto é o seu comportamento é indefinido se forem acedidos por uma tarefa que não a tarefa `EventDispatchThread`, o desenvolvimento de aplicações gráficas em Java que envolvam *Swing* e tarefas requer alguns cuidados [6] «*most Swing object methods are not "thread safe": invoking them from multiple threads risks thread interference or memory consistency errors. Some Swing component methods are labelled "thread safe" in the API specification; these can be safely invoked from any thread. All other Swing component methods must be invoked from the event dispatch thread. Programs that ignore this rule may function correctly most of the time, but are subject to unpredictable errors that are difficult to reproduce.*»

Como exemplo de demonstração vai ser utilizado uma aplicação que contem um componente de texto *Swing* (zona de *logging*) que é acedido por diferentes tarefas, incluindo a tarefa `EventDispatchThread`. A figura 5.1 apresenta o diagrama UML da aplicação desenvolvida.

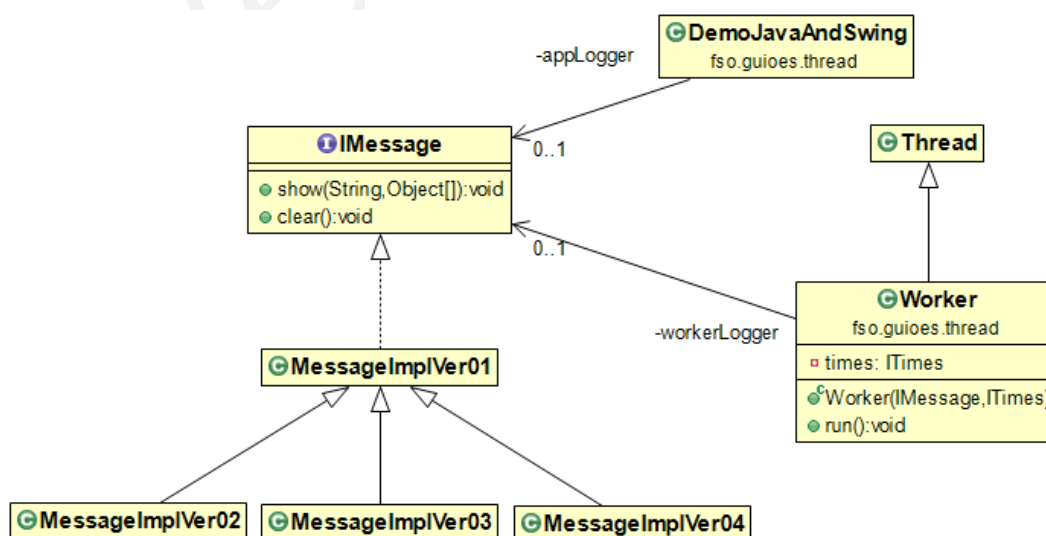


Figura 5.1: Diagrama UML da aplicação desenvolvida

As figuras 5.2 e 5.3 apresentam o resultado da execução da aplicação, desenvolvida com recurso ao *toolkit* Java *Swing*, onde um conjunto de tarefas (instâncias da classe *Worker*) acedem ao componente de *logging* (representado pela interface *IMessage*) cujas implementações estão desenvolvidas utilizando componentes *Swing*.

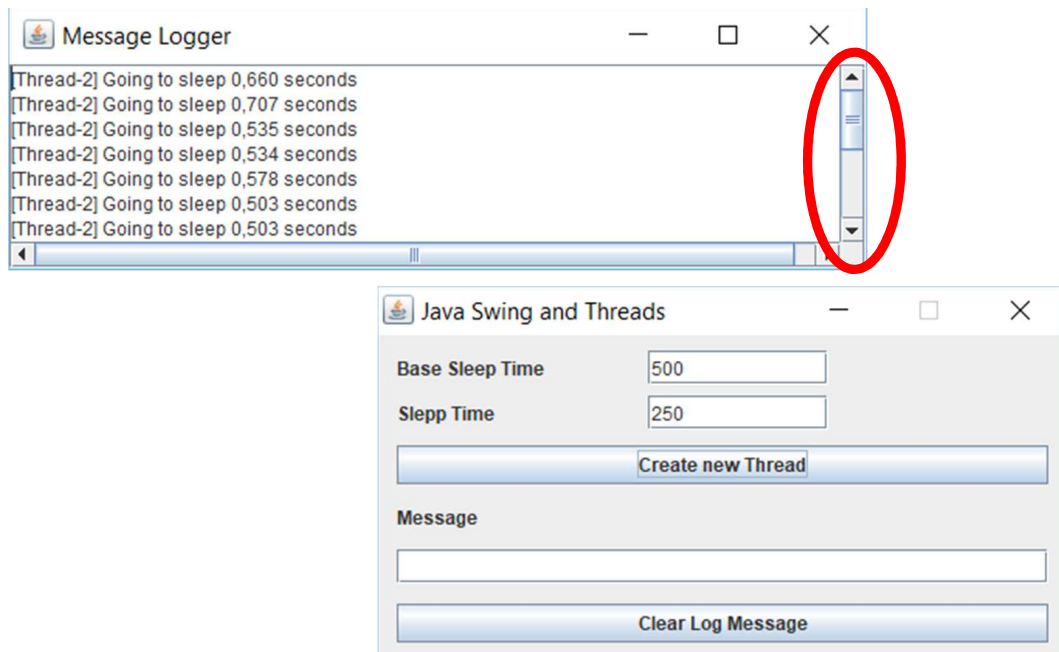


Figura 5.2: Aplicação *swing* desenvolvida — Comportamento indefinido

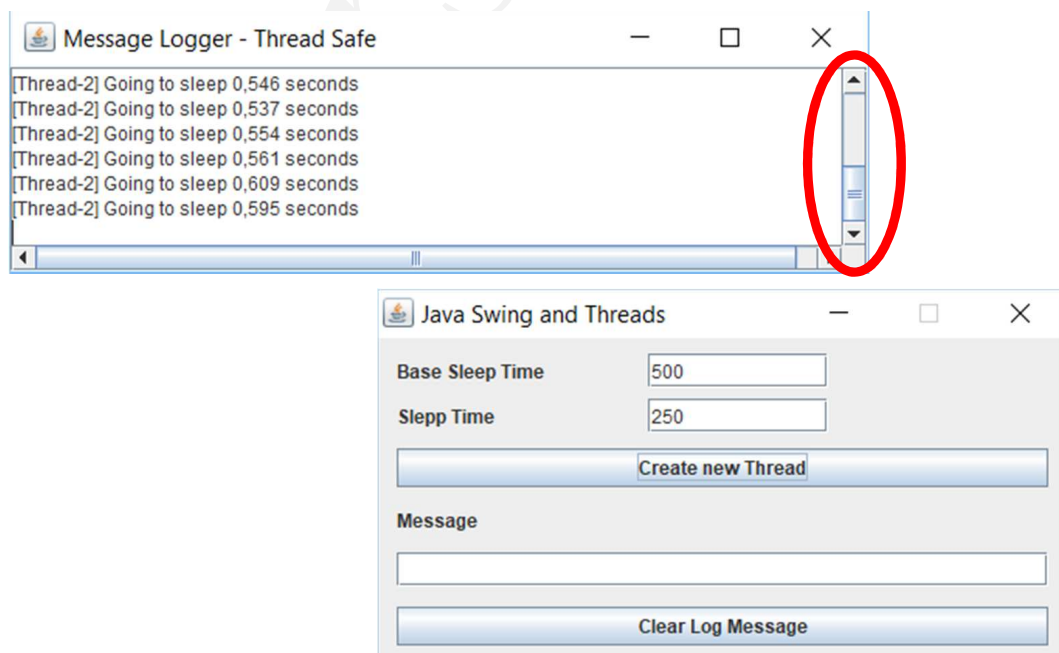


Figura 5.3: Aplicação *swing* desenvolvida — Comportamento correto

Tal como se pode observar nas figuras o posicionamento da barra de *scroll* apresenta comportamentos diferentes. A figura 5.3 representa o comportamento correto.

Assim, sempre que uma tarefa necessita de manipular um componente *Swing* deve delegar essa manipulação na tarefa *EventDispatchThread*. Essa delegação pode ser feita invocando os métodos estáticos *invokeLater* ou *invokeAndWait* disponíveis na classe *SwingUtilities* [7] do *package* *javax.swing*. Ambos os métodos recebem como argumento um objecto que implementa a interface *Runnable* e o método *run* desse objecto contém as instruções a aplicar ao componente *Swing*.

A listagem 5.1 exemplifica o acesso direto a um componente *Swing* por parte de uma tarefa que não a *EventDispatchThread*.

Listing 5.1: Manipulação de componentes *Swing* por tarefas do utilizador — Versão 1

```
1 public void show(String message, Object... args) {
2     String messageToShow;
3     messageToShow = String.format( "[%s] %s\n", Thread.
        ↳currentThread().getName(), String.format(message,
        ↳args) );
4
5     this.textAreaMessages.append( messageToShow );
6 }
```

A listagem 5.2 exemplifica o acesso indireto a um componente *Swing* (utilizando o método *invokeLater*) por parte de uma tarefa que não a *EventDispatchThread*.

Listing 5.2: Manipulação de componentes *Swing* por tarefas do utilizador — Versão 2

```
1 public void show(String message, Object... args) {
2     String messageToShow;
3     messageToShow = String.format( "[%s] %s\n", Thread.
        ↳currentThread().getName(), String.format(message,
        ↳args) );
4
5     Runnable update = new Runnable() {
6         @Override
7         public void run() {
8             textAreaMessages.append( messageToShow );
9         }
10    };
11
12    SwingUtilities.invokeLater( update );
```

A solução apresentada na listagem 5.2 funciona correctamente se for utilizada por uma tarefa que não a `EventDispatchThread`. No entanto, existem soluções em que o mesmo método também tem de ser invocado no contexto da tarefa `EventDispatchThread`. Por exemplo, a listagem 5.3 exemplifica o código implementado para limpar a área de *logging* que pode ser invocado por qualquer tarefa, incluindo a tarefa `EventDispatchThread` quando se prime o botão «Clear Log Message».

Listing 5.3: Manipulação de componentes *Swing* por tarefas do utilizador — Versão 3

```
1  public void clear() {
2      Runnable update = new Runnable() {
3          @Override
4          public void run() {
5              textAreaMessages.setText( "" );
6          }
7      };
8
9      try {
10         SwingUtilities.invokeLaterAndWait( update );
11     }
12     catch (Exception e) {
13         e.printStackTrace( System.err );
14     }
15 }
```

O resultado desta invocação resulta na excepção apresentada na figura 5.4.

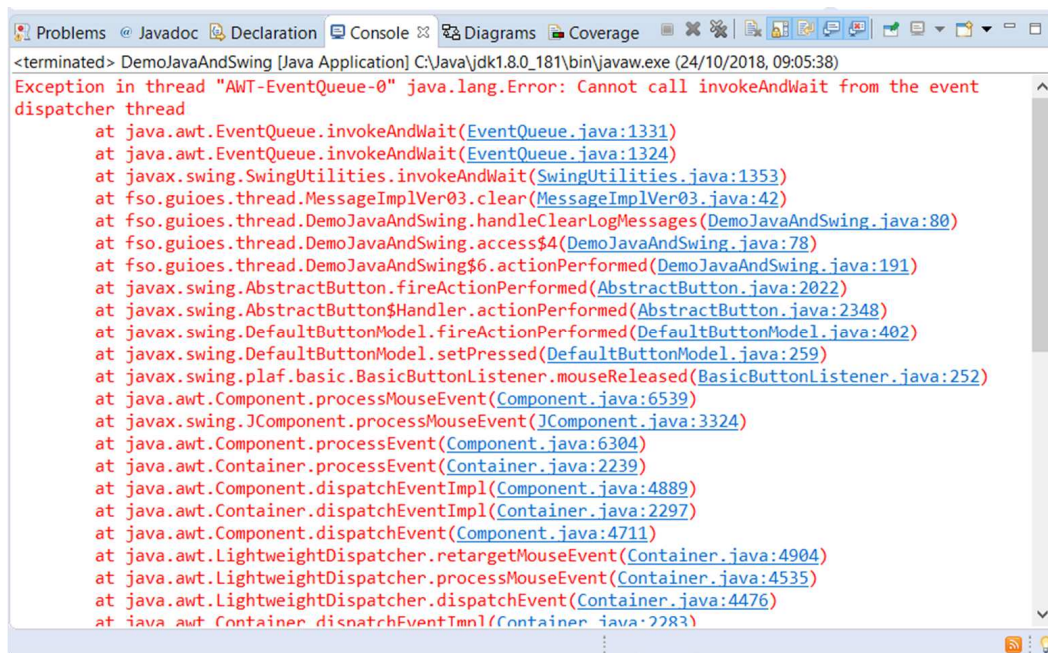


Figura 5.4: Resultado da invocação do método `invokeLater` por parte da tarefa `EventDispatchThread`

Nas situações em que o código que manipula os componentes *Swing* pode ser invocado por diferentes tarefas (incluído a `EventDispatchThread`), pode-se recorrer ao método estático `isEventDispatchThread` da classe `SwingUtilities` que devolve o valor booleano `true` se a tarefa que invoca o método for a tarefa `EventDispatchThread`. A listagem 5.4 exemplifica a utilização do método `invokeLater` em conjunto com o método `isEventDispatchThread`.

Listing 5.4: Manipulação de componentes *Swing* por tarefas do utilizador — Versão 4

```

1  public void clear() {
2      Runnable update = new Runnable() {
3          @Override
4          public void run() {
5              textAreaMessages.setText( "" );
6          }
7      };
8
9      try {
10         if ( !SwingUtilities.isEventDispatchThread() ) {
11             SwingUtilities.invokeLater( update );
12         }
13         else {
14             update.run();

```

```
15     }  
16 }  
17 catch (Exception e) {  
18     e.printStackTrace( System.err );  
19 }  
20 }
```

Versão 1.1

Bibliografia

- [1] J. Pais, “Folhas de apoio à Unidade Curricular de Fundamentos de Sistemas Operativos,” Sep. 2018. [Online]. Available: <https://1819.moodle.isel.pt/mod/resource/view.php?id=66270>
- [2] Oracle - Java Documentation. (2018, Oct.) Java TMPlatform, Standard Edition 8 API Specification — Classe Thread. Oracle. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>
- [3] ——. (2018, Oct.) Java TMPlatform, Standard Edition 8 API Specification — Interface Runnable. Oracle. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html>
- [4] ——. (2018, Oct.) Java TMPlatform, Standard Edition 8 API Specification — Classe AtomicBoolean. Oracle. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicBoolean.html>
- [5] ——. (2018, Oct.) Java TMPlatform, Standard Edition 8 API Specification — Classe AtomicBoolean. Oracle. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html>
- [6] ——. (2018, Oct.) The Java TMTutorials — The Event Dispatch Thread. Oracle. [Online]. Available: <https://docs.oracle.com/javase/tutorial/uiswing/concurrency/dispatch.html>
- [7] ——. (2018, Oct.) Java TMPlatform, Standard Edition 8 API Specification — Classe AtomicBoolean. Oracle. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/javax/swing/SwingUtilities.html>