Anexo C - **Reutilização de código**

O DrScheme tem vindo a oferecer esquemas simples, mas suficientemente flexíveis e poderosos, utilizados na reutilização de código. Utilizar código previamente desenvolvido e testado é uma atitude normal, que se aconselha, especialmente quando se enfrentam problemas que exijam programas de envergadura média ou acima da média.

Na versão 103 do DrScheme, era necessário começar por colocar os ficheiros, com o código a reutilizar, num directório criado em *PLT\collects*. Se esse directório se designasse, por exemplo, *user-feup* e se fosse *tartaruga.scm* o nome de um desses ficheiros, bastaria escrever, no início do programa

```
(require-library "tartaruga.scm" "user-feup")
```

para usufruir de todos os procedimentos e outras entidades incluídos naquele ficheiro.

Em versões mais recentes do DrScheme, o esquema oferecido é um pouco mais sofisticado, pois permite descriminar, daqueles ficheiros, as partes que interessam, mantendo tudo o resto escondido e inacessível. É assim possível controlar com rigor as entidades que podem ser reutilizadas, escondendo, por exemplo, procedimentos auxiliares que não interessa divulgar, o que contribui para minimizar a ocorrência de conflitos de nomes.

Nestas versões mais recentes, os ficheiros a reutilizar são também colocados num directório de *PLT\collects*. A requisição das entidades faz-se também de uma forma muito parecida com a anterior e quase que se confundem. Assim, se o directório a criar continuar a ser designado por *user-feup* e se o nome do ficheiro for *tartaruga.scm*, bastará agora escrever, no início do programa

```
(require (lib "tartaruga.scm" "user-feup"))
```

para usufruir não de todas, como acontecia na versão 103 do DrScheme, mas apenas de determinadas entidades daquele ficheiro. A principal diferença reside na forma como o ficheiro é constituído, pois deverá ser organizado como um módulo no qual são claramente especificadas as entidades a disponibilizar.
Sugere-se agora a análise cuidada do exemplo que se segue, em especial as características que se colocam em destaque:

- *eixos1* é o nome do módulo, que é seguido por *mzscheme*.
- dos vários procedimentos do módulo, apenas o procedimento *eixos* é disponibilizado, ficando os restantes escondidos.
- supondo *eixos1.scm* o nome do ficheiro com este módulo, colocado no directório *user-feup*, a sua reutilização ou, mais especificamente, a reutilização do procedimento *eixos* é possível escrevendo no início de um programa

```
(require (lib "eixos1.scm" "user-feup"))
```

```
(module eixos1
   mzscheme

   (define eixos
      (lambda (origem escala eixo-xx eixo-yy)
       ...      ; definição do procedimento eixos

   (define origem-eixos
      (lambda (sistema-de-eixos)
       ...      ; definição do procedimento origem-eixos

   (define escala-eixos
      (lambda (sistema-de-eixos)
       ...      ; definição do procedimento escala-eixos

   (define num-divisorias
      (lambda (sistema-de-eixos)
       ...      ; definição do procedimento num-divisorias

   (provide eixos))
```

pormenor de sintaxe na criação de um módulo

disponibilizado apenas o procedimento *eixos*

No exemplo que se segue, o módulo é designado por *eixos2* e supõe que o próprio módulo em definição requer entidades de outro módulo, colocado no directório *user-feup*, no ficheiro com o nome *swgr.scm*.

```
(module eixos2
   mzscheme

   (require (lib "swgr.scm" "user-feup"))

   (define eixos
      (lambda (origem escala eixo-xx eixo-yy)
       ...      ; definição do procedimento eixos

   (define origem-eixos
      (lambda (sistema-de-eixos)
       ...      ; definição do procedimento origem-eixos

   (define escala-eixos
      (lambda (sistema-de-eixos)
       ...      ; definição do procedimento escala-eixos

   (define num-divisorias
      (lambda (sistema-de-eixos)
       ...      ; definição do procedimento num-divisorias


   (provide eixos)

   ; a entidade janela é suposta
   ; disponível em "swgr.scm"

   (provide janela))
```

As entidades disponibilizadas por *swgr.scm*, apesar de estarem acessíveis em *eixos2*, não são automaticamente disponibilizadas para outros módulos que venham a requer *eixos2*. No entanto, entre as entidades disponibilizadas por *eixos2* podem também ser incluídas as que requereu a *swgr.scm*, situação ilustrada com o procedimento *janela*, suposto disponibilizado por *swgr.scm*.

Apresentam-se, seguidamente, a *abstracção tartaruga* e a *abstracção simulação de deslocamento*. A primeira é suposta guardada no ficheiro *tartaruga.scm* do directório *PLT\collects\user-feup* e a segunda no ficheiro *simula-desloca.scm* do mesmo directório.

```
(module tartaruga
  mzscheme

  (define faz-tartaruga          ; Três parâmetros, inteiros positivos.
    (lambda (col lin cor)             ; Devolve uma tartaruga situada em col
      (cons cor (cons col lin))))    ; e lin, com a cor dada. É um construtor.

  (define linha-tar              ; parâmetro do tipo tartaruga. Devolve um inteiro
    (lambda (tart)                   ; positivo que identifica a linha onde se
      (cdr (cdr tart))))             ; encontra a tartaruga. É um selector.

  (define coluna-tar             ; parâmetro do tipo tartaruga. Devolve um inteiro
    (lambda (tart)                        ; positivo que identifica a coluna onde se
      (car (cdr tart))))                  ; encontra a tartaruga. É um selector.

  (define cor-tar                ; Um parâmetro do tipo tartaruga. Devolve um
    (lambda (tart)                        ; inteiro positivo que identifica a cor da
      (car tart)))                        ; tartaruga. É um selector.

  (define visu-tar
    (lambda (tart)
      (display "tartaruga em col: ")
      (display (coluna-tar tart))
      (display ", lin: ")
      (display (linha-tar tart))
      (display ", cor: ")
      (display (cor-tar tart))
      (newline)))

  (provide faz-tartaruga)
  (provide linha-tar)
```

```
  (provide coluna-tar)
  (provide cor-tar)
  (provide visu-tar))
```

A abstracção *simulação de deslocamento* requer a *abstracção tartaruga.*

```
(module simula-desloca
  mzscheme

  (require (lib "tartaruga.scm" "user-feup"))

  (define vai-para-norte                    ; deslocamento para Norte de n posições.
    (lambda (tart n)
      (let ((destino (- (linha-tar tart)
                        n)))                  ; de facto, o que acontece é definir
        (faz-tartaruga (coluna-tar tart)      ; uma nova tartaruga com a mesma cor
                       (if (< destino 1)      ; e na mesma coluna e numa linha mais
                           1                  ; acima, não ultrapassando a linha 1.
                           destino)
                       (cor-tar tart)))))

  (define vai-para-sul                      ; deslocamento para Sul de n posições.
    (lambda (tart n)
      (faz-tartaruga (coluna-tar tart)        ; neste caso, não há limitações ao
                     (+ (linha-tar tart) n)   ; deslocamento.
                     (cor-tar tart))))

  (define vai-para-oeste                    ; ver comentários do procedimento
    (lambda (tart n)                          ; vai-para-norte.
      (let ((destino (- (coluna-tar tart)
                        n)))
        (faz-tartaruga (if (< destino 1)
                           1
                           destino)
                       (linha-tar tart)
                       (cor-tar tart)))))

  (define vai-para-este                      ; ver comentários do procedimento
    (lambda (tart n )                          ; vai-para-sul.
      (faz-tartaruga (+ (coluna-tar tart) n)
                     (linha-tar tart )
                     (cor-tar tart))))

  (define distancia-horizontal
    (lambda (tart1 tart2)               ; a distância horizontal entre tart1 e tart2
      (abs (- (coluna-tar tart1)        ; é dada pelo valor absoluto da diferença
              (coluna-tar tart2)))))    ; entre as colunas de tart1 e tar2

  (define distancia-vertical
    (lambda (tart1 tart2)
      (abs (- (linha-tar tart1)
              (linha-tar tart2)))))

  (define distancia
    (lambda (tart1 tart2)
      (+ (distancia-horizontal tart1 tart2)    ; este procedimento recorre aos dois
         (distancia-vertical tart1 tart2))))   ; procedimentos anteriores

  (provide vai-para-norte)
  (provide vai-para-sul)
  (provide vai-para-oeste)
  (provide vai-para-este)
  (provide distancia-horizontal)
  (provide distancia-vertical)
  (provide distancia))
```

Requerendo as duas abstracções, isoladamente, ficam acessíveis, obviamente, os recursos disponibilizados por ambas.

```
(require (lib "simula-desloca.scm" "user-feup"))
(require (lib "tartaruga.scm" "user-feup"))


> (define tar1 (faz-tartaruga 10 20 3))
> (define tar2 (vai-para-norte tar1 2))
> tar1
(3 10 . 20)
```

```
> tar2
(3 10 . 18)
```

Que alterações introduziria na abstracção simulação de deslocamento para tornar possível o diálogo que se segue, imaginando que utiliza qualquer procedimento da abstracção tartaruga.

```
(require (lib "simula-desloca.scm" "user-feup"))

> (define tar1 (faz-tartaruga 10 20 3))
> (define tar2 (vai-para-norte tar1 2))
> tar1
(3 10 . 20)
> tar2
(3 10 . 18)
>
```