Capítulo 6

Módulos em Haskell

Um módulo consiste num conjunto de definições (tipos e funções) com um interface claramente definido. O interface explícita o que é importado e/ou exportado por um dado módulo.

6.1 Definição de Módulos

Cabeçalho `module` <*Nome*> `where`
...

Importação `import` <*Nome*>
Através deste comando todas as definições existentes no módulo importado, e que são definidas neste como exportáveis, passam a ser locais ao presente módulo.

Exportação A definição das definições que um dado módulo exporta podem ser definidas das seguintes formas:

- por omissão todas definições do módulo.
- por declaração explícita no cabeçalho do módulo das definições que se pretende exportar. Por exemplo:

  `module` <*Nome*>`(`<*nome de funções e Tipos*> `where`
- No caso da definição dos tipos se se quiser que os construtores também sejam exportados é necessário explicita-los, por exemplo:

  `module` <*Nome*>`(`<... *Tipos(nome dos construtores)*...>`)` `where`

Quando um módulo é importado por outro podemos ainda explicitar o facto de que esse módulo é por sua vez exportado escrevendo.
`module` <*Nome*>`(`<...`module(`*nome*`)`...>`)` `where`

6.2 Um exemplo, o TAD Pilha

Vejamos um exemplo de definição e utilização dos módulos através da implementação do *TAD Pilha*.

Implementação com Exportação Implícita: vejamos uma primeira implementação em que, por omissão da nossa parte, todos os elementos do módulo são exportados.

```
module Pilha where
data Pilha a = Pvazia | Pl (a,Pilha a)
                  deriving (Show)

top :: Pilha a -> a
top Pvazia = error "Top - Pilha vazia"
top (Pl(a,p)) = a

pop :: Pilha a -> Pilha a
pop Pvazia = error "Pop - Pilha vazia"
pop (Pl(a,p)) = p

push :: (a,Pilha a) -> Pilha a
push (a,p) = Pl(a,p)

évazia :: Pilha a -> Bool
évazia Pvazia = True
évazia _ = False

vazia :: () -> Pilha a
vazia() = Pvazia
```

Como podemos ver é a definição usual do *TAD Pilha* encapsulado num módulo *Haskell*, como nada foi dito no cabeçalho toda a informação é exportada.

Vejamos agora uma segunda implementação com a definição explícita do que se pretende exportar.

Implementação com Exportação Explícita: para exportar só aquilo que deve ser vísivel fora do módulo é necessário explicitar toda a informação que será vísivel no exterior.

```
module Pilha(Pilha,vazia,pop,top,push,évazia) where
data Pilha a = Pvazia | Pl (a,Pilha a)
                  deriving (Show)

top :: Pilha a -> a
top Pvazia = error "pilha vazia"
top (Pl(a,p)) = a

pop :: Pilha a -> Pilha a
pop Pvazia = error "pilha vazia"
pop (Pl(a,p)) = p

push :: (a,Pilha a) -> Pilha a
push (a,p) = Pl(a,p)
```

```
évazia :: Pilha a -> Bool
évazia Pvazia = True
évazia _ = False

vazia :: () -> Pilha a
vazia() = Pvazia
```

É de notar que, aqui tudo o que se omite não é exportado, nomeadamente como não declaramos no cabeçalho os construtores do novo tipo estes não será vísiveis do exterior, obtemos deste modo o comportamento desejado para um Tipo Abstracto de Dados.

Utilização: como exemplo de uma utilização do *TAD Pilha* temos o seguinte módulo em que se procede à linearização de uma pilha.

```
module UsaPilhas where

import Pilha

-- lineariza pilha
lineariza :: Pilha a -> [a]
lineariza p
    | évazia(p) = []
    | otherwise = top(p):lineariza(pop(p))
```

Toda a tentativa de utilização dos constructores provocará um erro dado que no módulo UsaPilhas eles não são conhecidos.