Haskell Clases de Tipos – sobrecarga - Herramientas

Clases de Tipos - sobrecarga

Con firma y sin sobrecarga	Con firma y con sobrecarga

esPar.hs

```
{-
Para saber si un numero es par
-}
esPar :: Int -> Bool
esPar x = if (mod x 2 == 0) then True else False
```

esPar_1.hs

```
{-
Para saber si un numero es par
-}
esPar :: (Integral a)=> a -> Bool
esPar x = if (mod x 2 == 0) then True else False
```

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
GHCi, version 8.8.4: https://www.haskell.org/ghc/  :? for help
Prelude> :cd haskell
Prelude> :l esPar.hs
[1 of 1] Compiling Main             ( esPar.hs, interpreted )
Ok, one module loaded.
*Main> esPar 1011
False
*Main> esPar 1022
True
*Main> esPar 1011000000002003000000

<interactive>:5:7: warning: [-Woverflowed-literals]
    Literal 1011000000002003000000 is out of the Int range -9223372036854775808..9223372036854775807
True
*Main> :l esPar_1.hs
[1 of 1] Compiling Main             ( esPar_1.hs, interpreted )
Ok, one module loaded.
*Main> esPar 1011000000002003000000
True
*Main>
```

Evidentemente el número es muy grande … para el caso de la función esPar.hs y no para la función esPar_1.hs ya que esta incluye el conjunto de tipos Integral

Se puede utilizar el comando :info para conocer las Clases de Tipos

Por ejemplo

:info Int

Lo primero que muestra es que es un tipo de datos Int: **data Int**

:info Num

Lo primero que nos muestra es que se trata de una clase de tipo: **class Num** y luego nos muestra los métodos: (+), (-) , (*), negate, etc.

También las instancias: Int, Integer, Float, Duble, Word

:info Integral

También vemos que se trata de una clase y que sus instancias son Word, Int e Integer.

Entre sus métodos está mod

```
*Main> :info Int
data Int = GHC.Types.I# GHC.Prim.Int#   -- Defined in ‘GHC.Types’
instance Eq Int -- Defined in ‘GHC.Classes’
instance Ord Int -- Defined in ‘GHC.Classes’
instance Enum Int -- Defined in ‘GHC.Enum’
instance Num Int -- Defined in ‘GHC.Num’
instance Real Int -- Defined in ‘GHC.Real’
instance Show Int -- Defined in ‘GHC.Show’
instance Integral Int -- Defined in ‘GHC.Real’
instance Bounded Int -- Defined in ‘GHC.Enum’
instance Read Int -- Defined in ‘GHC.Read’
*Main> :info Num
class Num a where
  (+) :: a -> a -> a
  (-) :: a -> a -> a
  (*) :: a -> a -> a
  negate :: a -> a
  abs :: a -> a
  signum :: a -> a
  fromInteger :: Integer -> a
  {-# MINIMAL (+), (*), abs, signum, fromInteger, (negate | (-)) #-}
        -- Defined in ‘GHC.Num’
instance Num Word -- Defined in ‘GHC.Num’
instance Num Integer -- Defined in ‘GHC.Num’
instance Num Int -- Defined in ‘GHC.Num’
instance Num Float -- Defined in ‘GHC.Float’
instance Num Double -- Defined in ‘GHC.Float’
*Main> :info Integral
class (Real a, Enum a) => Integral a where
  quot :: a -> a -> a
  rem :: a -> a -> a
  div :: a -> a -> a
  mod :: a -> a -> a
  quotRem :: a -> a -> (a, a)
  divMod :: a -> a -> (a, a)
  toInteger :: a -> Integer
  {-# MINIMAL quotRem, toInteger #-}
        -- Defined in ‘GHC.Real’
instance Integral Word -- Defined in ‘GHC.Real’
instance Integral Integer -- Defined in ‘GHC.Real’
instance Integral Int -- Defined in ‘GHC.Real’
*Main>
```

Podemos ampliar la información de tipos y firma de las funciones o métodos con :type o :t

```
Prelude> :info Integral
class (Real a, Enum a) => Integral a where
  quot :: a -> a -> a
  rem :: a -> a -> a
  div :: a -> a -> a
  mod :: a -> a -> a
  quotRem :: a -> a -> (a, a)
  divMod :: a -> a -> (a, a)
  toInteger :: a -> Integer
  {-# MINIMAL quotRem, toInteger #-}
        -- Defined in ‘GHC.Real’
instance Integral Word -- Defined in ‘GHC.Real’
instance Integral Integer -- Defined in ‘GHC.Real’
instance Integral Int -- Defined in ‘GHC.Real’
Prelude> :t mod
mod :: Integral a => a -> a -> a
Prelude> :t div
div :: Integral a => a -> a -> a
Prelude> :t rem
rem :: Integral a => a -> a -> a
Prelude> :t quot
quot :: Integral a => a -> a -> a
Prelude> :t rem 10 4
rem 10 4 :: Integral a => a
Prelude> rem 10 4
2
Prelude> rem 10 3
1
Prelude> mod 10 4
2
Prelude> mod 10 3
1
```

```
Prelude> mod 40 4
0
Prelude> divMod 40 4
(10,0)
Prelude> quot 40 4
10
Prelude> rem 40 4
0
Prelude> quotRem 40 4
(10,0)
Prelude> toInteger 10
10
Prelude> :t 10
10 :: Num p => p
Prelude> :t (toInteger 10)
(toInteger 10) :: Integer
Prelude> :t quotRem 40 4
quotRem 40 4 :: Integral a => (a, a)
Prelude>
```

http://learnyouahaskell.com/types-and-typeclasses

Continua ...