JFP **13** (1): 191–192, January 2003. © 2003 Cambridge University Press
DOI: 10.1017/S0956796803002016 Printed in the United Kingdom

Chapter 18

Maybe Utilities

```
module Maybe(
    isJust, isNothing,
    fromJust, fromMaybe, listToMaybe, maybeToList,
    catMaybes, mapMaybe,

    -- ...and what the Prelude exports
    Maybe(Nothing, Just),
    maybe
  ) where

isJust, isNothing    :: Maybe a -> Bool
fromJust             :: Maybe a -> a
fromMaybe            :: a -> Maybe a -> a
listToMaybe          :: [a] -> Maybe a
maybeToList          :: Maybe a -> [a]
catMaybes            :: [Maybe a] -> [a]
mapMaybe             :: (a -> Maybe b) -> [a] -> [b]
```

The type constructor `Maybe` is defined in `Prelude` as

```
data Maybe a = Nothing | Just a
```

The purpose of the `Maybe` type is to provide a method of dealing with illegal or optional values without terminating the program, as would happen if `error` were used, and without using

`IOError` from the `IO` monad, which would cause the expression to become monadic. A correct result is encapsulated by wrapping it in `Just`; an incorrect result is returned as `Nothing`.

Other operations on `Maybe` are provided as part of the monadic classes in the Prelude.

18.1 Library `Maybe`

```
module Maybe(
    isJust, isNothing,
    fromJust, fromMaybe, listToMaybe, maybeToList,
    catMaybes, mapMaybe,

    -- ...and what the Prelude exports
    Maybe(Nothing, Just),
    maybe
  ) where

isJust                 :: Maybe a -> Bool
isJust (Just a)        =  True
isJust Nothing         =  False

isNothing              :: Maybe a -> Bool
isNothing              =  not . isJust

fromJust               :: Maybe a -> a
fromJust (Just a)      =  a
fromJust Nothing       =  error "Maybe.fromJust: Nothing"

fromMaybe              :: a -> Maybe a -> a
fromMaybe d Nothing    =  d
fromMaybe d (Just a)   =  a

maybeToList            :: Maybe a -> [a]
maybeToList Nothing    =  []
maybeToList (Just a)   =  [a]

listToMaybe            :: [a] -> Maybe a
listToMaybe []         =  Nothing
listToMaybe (a:_)      =  Just a

catMaybes              :: [Maybe a] -> [a]
catMaybes ms           =  [ m | Just m <- ms ]

mapMaybe               :: (a -> Maybe b) -> [a] -> [b]
mapMaybe f             =  catMaybes . map f
```