MonadState Class (9B)

Young Won Lim
9/20/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Young Won Lim
9/20/18

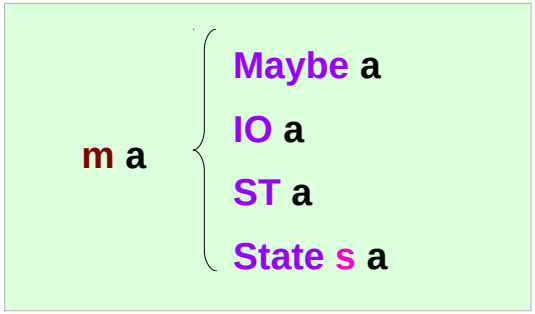
Based on

Haskell in 5 steps

https://wiki.haskell.org/Haskell_in_5_steps

Monad typeclass and Instances

```
class Monad m where
   return :: a -> m a
   (>>=) :: m a -> (a -> m b) -> m b
   (>>) :: m a -> m b -> m b
   fail :: String -> m a
```




```
instance Monad Maybe where
      return x = Just x
      Nothing >>= f = Nothing
      Just x >>= f  = f x
      fail _ = Nothing
```

```
instance  Monad IO  where
   m >> k    = m >>= \ _ -> k
   return    = ...
   (>>=)     = ...
   fail s    = ...
```

Default Implementations in MonadState s m

```
class Monad m => MonadState s m | m -> s where
```

```
-- | Return the state from the internals of the monad.
get :: m s
get = state (\s -> (s, s))

-- | Replace the state inside the monad.
put :: s -> m ()
put s = state (\_ -> ((), s))
```

```
-- | Embed a simple state action into the monad.
state :: (s -> (a, s)) -> m a
state f = do
  s <- get
  let ~(a, s') = f s
  put s'
  return a
```

https://stackoverflow.com/questions/23149318/get-put-and-state-in-monadstate

The mtl package

Control.Monad.State.Class module

No dead loop in the default implementation

the definitions of **get**, **put**,**state** in the **Monad class declaration**

- the default implementations,
- to be overridden in actual **instances** of the class.

the dead loop in the default definition does not happen:

- **put** and **get** in terms of **state**
- **state** in terms of **put** and **get**

* minimal definition is *either* both of **get** and **put** *or* just **state**

```
get :: m s
get = state (\s -> (s, s))

put :: s -> m ()
put s = state (\_ -> ((), s))
```

```
state :: (s -> (a, s)) -> m a
state f = do
  s <- get
  let ~(a, s') = f s
  put s'
  return a
```

https://stackoverflow.com/questions/23149318/get-put-and-state-in-monadstate

Functional Dependency | (vertical bar)

```
class Monad m => MonadState s m | m -> s where …
```

functional dependencies

to constrain the parameters of type classes. **s** and **m**

s can be determined from **m**, **m** → **s**

so that **s** can be the return type **State s** → **s**

but **m** can not be the return type

in a multi-parameter type class,

one of the parameters can be determined from the others,

so that the parameter determined by the others can be the return type

but none of the argument types of some of the methods.

```
class Monad m where
  return :: a -> m a
  (>>=) :: m a -> (a -> m b) -> m b
  (>>) :: m a -> m b -> m b
  fail :: String -> m a
```

m a ⟨ **Maybe a**, **IO a**, **ST a**, **State s a**

https://stackoverflow.com/questions/23149318/get-put-and-state-in-monadstate

Typeclass MonadState s

class Monad m => MonadState s m | m -> s where …

MonadState s

a typeclass

instance MonadState s MM where …

its type instance itself does not specify values

MonadState s m =>

- can be used as class constraint
- all the **Monad m**

  which supports *state operations* with state of type **s**.

:t get

:t put

s ← m **functional dependencies**

m á State s → s

state operations
defined in the
typeclass definition

https://stackoverflow.com/questions/25438575/states-put-and-get-functions

Types of **get** and **put**

:t get ► **get :: MonadState s m => m s**

for all **Monad m** which supports *state operations* over state of type **s**,
we have a value of type **m s** - that is,
the monad operation which yields the current state

get :: m s

:t put ► **put :: MonadState s m => s -> m ()**

a function that takes a value of type **s**
and returns a polymorphic value
representing any **Monad m**
which supports state operations over a state of type **s**

put :: s -> m ()

https://stackoverflow.com/questions/25438575/states-put-and-get-functions

Instances of MonadState s m

```
class Monad m => MonadState s m | m -> s where
```

The mtl package

Control.Monad.State.Class module

```
instance Monad m => MonadState s (Lazy.StateT s m) where …
instance Monad m => MonadState s (Strict.StateT s m) where …
instance MonadState s m => MonadState s (ContT r m) where …
instance MonadState s m => MonadState s (ReaderT r m) where …
instance (Monoid w, MonadState s m) => MonadState s (Lazy.WriterT w m) where …
instance (Monoid w, MonadState s m) => MonadState s (Strict.WriterT w m) where …
```

m

- Lazy.StateT s m
- Strict.StateT s m
- ContT r m
- ReaderT r m
- Lazy.WriterT w m
- Strict.WriterT w m

https://stackoverflow.com/questions/23149318/get-put-and-state-in-monadstate

Instances of the typeclass MonadState s

MonadState s is the class of types that are monads with state.

```
instance MonadState s (State s) where
  get = Control.Monad.Trans.State.get
  put = Control.Monad.Trans.State.put
```

State s is an instance of that typeclass:

```
instance MonadState s (StateT s) where
  get = Control.Monad.Trans.State.get
  put = Control.Monad.Trans.State.put
```

StateT s is an instance of that typeclass:
(the state monad transformer
which adds state to another monad)

https://stackoverflow.com/questions/25438575/states-put-and-get-functions

Overloading get and put

```
instance MonadState s (State s) where
  get = Control.Monad.Trans.State.get
  put = Control.Monad.Trans.State.put
```

This **overloading** was introduced so that

if you're using a stack of monad transformers,

    you do not need to explicitly **lift** operations

    between different transformers.

If you're not doing that,

    you can use the simpler operations from transformers.

The **mtl** package provides

auto-lifting

https://stackoverflow.com/questions/25438575/states-put-and-get-functions

Typeclass Constrain MonadState s m (1)

```
class Monad m => MonadState s m | m -> s where …
```

```
get :: MonadState s m => m s
```

for some monad **m**

storing some state of type **s**,

get is an action in **m**

that returns a value of type **s**.

https://stackoverflow.com/questions/25438575/states-put-and-get-functions

Typeclass Constrain MonadState s m (2)

```
class Monad m => MonadState s m | m -> s where …
```

```
put :: MonadState s m => s -> m ()
```

for some monad **m**

put is an action in **m**

storing the given state of type **s**,

but returns nothing **()**.

https://stackoverflow.com/questions/25438575/states-put-and-get-functions

MonadState Class

class Monad m => MonadState s m | m -> s where

Minimal definition is either both of get and put or just state

Minimal complete definition

state | get, put

Methods

get :: m s

Return the state from the internals of the monad.

put :: s -> m ()

Replace the state inside the monad.

state :: (s -> (a, s)) -> m a

Embed a simple state action into the monad.

https://hackage.haskell.org/package/mtl-2.2.2/docs/Control-Monad-State-Lazy.html

MonadState Class

modify :: MonadState s m => (s -> s) -> m ()

Monadic state transformer.

Maps an old state to a new state inside a state monad. The old state is thrown away.

Main> :t modify ((+1) :: Int -> Int)

modify (...) :: (MonadState Int a) => a ()

This says that modify (+1) acts over any Monad

that is a member of the MonadState class, with an Int state.

modify' :: MonadState s m => (s -> s) -> m ()

A variant of modify in which the computation is strict in the new state.

Since: 2.2

gets :: MonadState s m => (s -> a) -> m a

Gets specific component of the state, using a projection function supplied.

https://hackage.haskell.org/package/mtl-2.2.2/docs/Control-Monad-State-Lazy.html

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf

[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf