Класс типов MonadFail предназначен для обработки неудачного сопоставления с образцом слева от <- в do-нотации.

```
class Monad m => MonadFail m where
  fail :: String -> m a

instance MonadFail Maybe where
  fail _ = Nothing
```

```
GHCi> do {3 <- Just 5; return 'Z'}
Nothing
GHCi> do {3 <- Identity 5; return 'Z'}
error: Could not deduce (MonadFail Identity) arising
      from a do statement with the failable pattern `3'
```

Сообщение об ошибке выдает тайпчекер, код не пройдет компиляцию.

do-нотация транслируется в Haskell Kernel по-разному, в зависимости от того является ли образец «failable» или нет:

```
GHCi> :t do {x <- return 5; return 'Z'}
do {x <- return 5; return 'Z'} :: Monad m => m Char

GHCi> :t do {3 <- return 5; return 'Z'}
do {3 <- return 5; return 'Z'} :: MonadFail m => m Char

GHCi> :t do {~3 <- return 5; return 'Z'}
do {~3 <- return 5; return 'Z'} :: Monad m => m Char
```

Неопровержимые образцы не являются «failable».

data с одним конструктором и newtype не «failable» сами по себе, но могут оказаться «failable» при вложении образцов.

```
GHCi> :t do {(s,x) <- return ("Answer",42); return 'Z'}
do {(s,x) <- return ("Answer",42); return 'Z'}
    :: Monad m => m Char

GHCi> :t do {(s,42) <- return ("Answer",42); return 'Z'}
do {(s,42) <- return ("Answer",42); return 'Z'}
    :: MonadFail m => m Char

GHCi> :t do {Left x <- return (Left 42); return 'Z'}
do {Left x <- return (Left 42); return 'Z'}
    :: MonadFail m => m Char
```

В GHCi расширенный меанизм дефолтинга при необходимости трактует произвольную монаду как `IO`

```
GHCi> :t fail "qqq"
fail "qqq" :: MonadFail m => m a
GHCi> fail "qqq"
*** Exception: user error (qqq)
```

Когда при неудачном сопоставлении `fail` вызывается системой, в строковой параметр передается информация о типе и месте ошибки

```
GHCi> do {True <- return False; return 42}
*** Exception: user error (Pattern match failure in do
    expression at <interactive>:4:5-8)
```

Закон, связывающий классы типов `Monad` и `MonadFail`

`fail s` — это левый ноль для `(>>=)`

```
fail s >>= k  ≡  fail s
```

Для `Maybe` он, конечно же, выполняется

```
GHCi> :t fail "Oh!" >>= granmas
fail "Oh!" >>= granmas :: Maybe (Name, Name)
GHCi> fail "Oh!" >>= granmas
Nothing
GHCi> fail "Oh!" :: Maybe (Name, Name)
Nothing
```

```
instance MonadFail [] where
  fail :: String -> [a]
  fail _     =  []
```