Maybe Instanzen

```haskell
data Maybe a = Nothing | Just a
-- Functor instance for Maybe
instance Functor Maybe where
  fmap :: (a -> b) -> Maybe a -> Maybe b
  fmap _ Nothing = Nothing
  fmap f (Just x) = Just (f x)

-- Applicative instance for Maybe
instance Applicative Maybe where
  pure :: a -> Maybe a
  pure = Just

  (<*>) :: Maybe (a -> b) -> Maybe a -> Maybe b
  Nothing <*> _ = Nothing
  (Just f) <*> something = fmap f something

-- Monad instance for Maybe
instance Monad Maybe where
  return :: a -> Maybe a
  return = Just

  (>>=) :: Maybe a -> (a -> Maybe b) -> Maybe b
  Nothing >>= _ = Nothing
  Just x >>= f = f x

-- Alternative instance for Maybe
instance Alternative Maybe where
  empty :: Maybe a
  empty = Nothing

  (<|>) :: Maybe a -> Maybe a -> Maybe a
  Nothing <|> r = r
  l <|> _ = l

-- Foldable instance for Maybe
instance Foldable Maybe where
  foldr :: (a -> b -> b) -> b -> Maybe a -> b
  foldr _ z Nothing = z
  foldr f z (Just x) = f x z

  foldl :: (b -> a -> b) -> b -> Maybe a -> b
  foldl _ z Nothing = z
  foldl f z (Just x) = f z x

  foldMap :: Monoid m => (a -> m) -> Maybe a -> m
  foldMap _ Nothing = mempty
  foldMap f (Just x) = f x
```

```haskell
-- Traversable instance for Maybe
instance Traversable Maybe where
  traverse :: Applicative f => (a -> f b) -> Maybe a -> f (Maybe b)
  traverse _ Nothing = pure Nothing
  traverse f (Just x) = Just <$> f x

-- Eq instance for Maybe
instance Eq a => Eq (Maybe a) where
  Nothing == Nothing = True
  Just x == Just y = x == y
  _ == _ = False

-- Ord instance for Maybe
instance Ord a => Ord (Maybe a) where
  compare Nothing Nothing = EQ
  compare Nothing (Just _) = LT
  compare (Just _) Nothing = GT
  compare (Just x) (Just y) = compare x y

-- Show instance for Maybe
instance Show a => Show (Maybe a) where
  show Nothing = "Nothing"
  show (Just x) = "Just " ++ show x

-- Read instance for Maybe
instance Read a => Read (Maybe a) where
  readsPrec _ value =
    tryParse [("Nothing", Nothing), ("Just", Just)]
    where
      tryParse [] = []
      tryParse ((attempt, result):xs) =
        case stripPrefix attempt value of
          Just rest -> [(result, rest)]
          Nothing   -> tryParse xs

-- Semigroup instance for Maybe
instance Semigroup a => Semigroup (Maybe a) where
  Nothing <> r = r
  l <> Nothing = l
  Just x <> Just y = Just (x <> y)

-- Monoid instance for Maybe
instance Semigroup a => Monoid (Maybe a) where
  mempty = Nothing
  mappend = (<>)
```