Clase, tipuri, instanțe în Haskell

```
class Eq a where
   (==) :: a -> a -> Bool
 instance Eq Integer where
    x == y   = x `integerEq` y.
 instance (Eq a) => Eq (Tree a) where
       Leaf a == Leaf b  = a == b
       (Branch l1 r1) == (Branch l2 r2) = (l1 == l2) && (r1 == r2)
         _  ==  _  = False
class (Eq a) => Ord a where
     (<), (<=), (>=), (>) :: a -> a -> Bool
      max, min :: a -> a -> a
       x < y   = x <= y && x /= y.
 instance Num Natural where ...
 data Point = Pt {pointx, pointy :: Float}.
      pointx :: Point -> Float  -- data class
      -- la fel și la pointy
      absPoint :: Point -> Float  -- funcție
      absPoint (Pt {pointx = x, pointy = y}) = sqrt (x*x + y*y)
```

Clase standard Haskell

```
data Ordering = EQ | LT | GT
compare :: Ord a => a -> a -> Ordering
show :: (Show a) => a -> String
instance Show a => Show (Tree a) where
        showsPrec _ x = showsTree x.      →
```

```
instance Show a => Show (Tree a) where
        show t = showTree

showsTree :: (Show a) => Tree a -> ShowS
    shows (Leaf x) = shows x
    showsTree (Branch l r) = ('<' : ) . showsTree l . ('|' : ) . showsTree r . ('>' : )

data Tree a = Leaf a | Branch (Tree a) (Tree a) deriving Eq

instance (Ord a) => Ord (Tree a) where
    (Leaf _) <= (Branch _) = True
    (Leaf x) <= (Leaf y) = x <= y
    (Branch _) <= (Leaf _) = False
    (Branch l r) <= (Branch l' r') = l == l' && r <= r' || l <= l'
```