```
--- Declaremos un Tipo  de dato para
--- representar a personas con datos ApellidoyNombre, DNI,  edad

type Nombre = [Char]
type Edad   = Integer

data Persona = Nadie | Datos {apeyNom::Nombre,dNI::Integer, anios::Edad}
deriving Show

--- Datos de ejemplo
-----------------------------------------------------
p1 = Datos{apeyNom="Perez,Pedro", dNI=38666666, anios=26}
p2 = Nadie
p3 = Datos{apeyNom="Pirez,Pedro", dNI=38666667, anios=27}
p4 = Datos{apeyNom="Villar, Fernando", dNI=17175386, anios=57}
-----------------------------------------------------------

--- Ahora necesitamos hacer que el nuevo tipo "Persona"
--- pueda ser utilizado en una estructura que solo soporta
--- datos que sean instancias de la clase "Ord" y de "Eq"
--- esta clase supone la existencia de las operaciones <, <=, >, >=, ==,
/=
--- Vamos a suponer que la clave para clasificar objetos del tipo Persona
--- va a ser el campo dNI

instance Eq Persona where
   (==) Nadie Nadie = True
   (==) Nadie _     = False
   (==) _     Nadie  = False
   (==) x y         = dNI x == dNI y
   (/=) x y         = not (x == y)

instance Ord Persona where
  (<) Nadie _                                  = True
  (<) _    Nadie                               = False
  (<) x y                                      = dNI x < dNI y

  (>) Nadie _                                  = False
  (>) _ Nadie                                  = True
  (>) x  y                                     = not (x < y)

  (<=) Nadie _                                 = True
  (<=) _    Nadie                              = False
  (<=) x y                                     = dNI x <= dNI y

  (>=) Nadie _                                 = False
  (>=) _ Nadie                                 = True
  (>=) x y                                     = not (x <= y)

--- Observar como podemos declarar operaciones utilizando lo ya definido
--- (>) = not (<)

--- Volvamos a nuestro ejemplo de diccionario implementado con un arbol
binario
--- Primero Definimos el TDA Arbol Binario que sera nuestra herramienta
--Arbol binario de búsqueda

data  Bintree a = EmptyBT | NodoBT a (Bintree a) (Bintree a)
```

```
                  deriving Show

setEmptyTree::(Ord a)=> Bintree a
setEmptyTree = EmptyBT

inTree::Ord a => a -> Bintree a -> Bool
inTree x EmptyBT = False
inTree x (NodoBT y lf rt) | x == y  = True
                          | x < y   = inTree x lf
                          | x > y   = inTree x rt

addTree:: (Ord a)=> a-> Bintree a -> Bintree a
addTree x EmptyBT = NodoBT x EmptyBT EmptyBT
addTree x (NodoBT y lf rt) | x == y    = NodoBT y lf rt
                           | x < y     = NodoBT y (addTree x lf) rt
                           | otherwise = NodoBT y lf (addTree x rt)

delTree:: (Ord a)=> a -> Bintree a -> Bintree a
delTree x EmptyBT = EmptyBT
delTree x (NodoBT y lf EmptyBT)
          | x == y  = lf
delTree x (NodoBT y EmptyBT rt)
          | x == y = rt
delTree x (NodoBT y lf rt)
        |x < y   = NodoBT y (delTree x lf) rt
        |x > y   = NodoBT y lf (delTree x rt)
        |x == y  = let (k,wt) = minTree (rt)
                   in (NodoBT k lf wt)

--- Funcion Auxiliar extrae el minimo de un arbol
--- devuelve la clave mas pequeña y el arbol sin ese elemento
minTree::(Ord a)=> Bintree a -> (a, Bintree a)
minTree (NodoBT  v EmptyBT rt) = (v,rt)
minTree (NodoBT  v    lf   rt )=
                               let (x, new_lf) = minTree lf
                               in
                                   (x, NodoBT v new_lf rt)

--- Diccionario implementado con un arbol binario de busqueda
--- Para ello referimos las operaciones del  diccionario a operaciones
sobre el arbol
--- Esto es Totalmente Invisible para el Usuario del TDA Diccionario

newtype Dict a = Dicc (Bintree a) deriving Show

mkDict:: (Ord a)=> Dict a
mkDict = Dicc (setEmptyTree)

insertDict :: (Ord a)=> a -> Dict a -> Dict a
insertDict x (Dicc t) = Dicc (addTree x t)

inDict::(Ord a) => a -> Dict a -> Bool
inDict x (Dicc t) = inTree x t

delDict::(Ord a)=> a -> Dict a -> Dict a
delDict x (Dicc t) = Dicc ( delTree x t)
```

```
--- _____________________________________________________
--- Para visualizar lo hecho
--- en prompt de Main ejecutar
--- md = mkDict
--- diccionario = insertDict p4 (insertDict p3 (insertDict p2 (insertDict
p1 md)))
--- visualizar el  contenido  de diccionario.
--- Observar como se han almacenado los objetos cuyas claves son ...
---
---                              38666666
---                             /        \
---                        Nadie          38666667
---                       /     \           /     \
---                EmptyBT    17175386   EmptyBT   EmptyBT
---                           /     \
---                      EmptyBT    EmptyBT

--- Luego solicitar "delDict Nadie diccionario"
--- y visualizar
---                              38666666
---                             /        \
---                         17175386      38666667
---                        /     \          /     \
---                  EmptyBT    EmptyBT  EmptyBT   EmptyBT
---
---
```