Tema C

Ejercicio 1:

Una casa de venta de piedras preciosas desea representar su stock usando tipos en Haskell. Las piedras que tenemos en cuenta son: `Diamante, Esmeralda, Rubi y Zafiro.` La idea es poder detallar para cada tipo de piedra, las características más importantes. En tal sentido identificamos las siguientes características de cada una de las piedras a tener en cuenta:

`Diamante`

- `Color`, que es un tipo enumerado con las siguientes opciones: `Azul, Rosa, Amarillo, Marron, Gris y Blanco`
- `Dureza que es un enumerado con las siguientes opciones: MuyDura, Dura, Fragil.`
- `Peso, que es un sinónimo de Int indicando el peso de la piedra`
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Esmeralda`

- `Dureza (las mismas opciones que en Diamante)`
- `Peso que es un sinónimo de Int indicando el peso de la piedra`
- `Precio`, que es un sinónimo de `Int` indicando el precio

`Rubi`

- `ColorRubi con opciones Sangre o Carmesi`
- `Precio`, que es un sinónimo de `Int` indicando el precio

Zafiro

- `Color con opciones iguales que para Diamante`
- `Precio`, que es un sinónimo de `Int` indicando el precio

Para ello:

a) Definir el tipo `PiedrasPreciosas` que consta de los constructores `Diamante, Esmeralda, Rubi y Zafiro` constructores con parámetros descriptos arriba (Se deben definir también los tipos enumerados y los sinónimos de tipos`)`. **Los tipos `definidos` no deben tener deriving `Eq`**, ni **`Ord`**. Agregue deriving Show a todos los tipos.

b) Definir la función `cuantosDiamantes` de la siguiente manera:

```
cuantosDiamantes :: [PiedrasPreciosas] -> Color -> Int
```

que dada una lista de `PiedrasPreciosas lm` y un valor `x` de `Color`, me devuelve un entero indicando la cantidad de Diamantes que hay en `lm` de color `x`.
NOTA: Dejar como comentario un ejemplo donde hayas probado la función `cuantosDiamantes` con una lista con al menos 3 `PiedrasPreciosas`.

c) Definir igualdad para el tipo de `PiedrasPreciosas`: de tal manera que, dos valores de tipo `Diamante` son iguales sólo si tienen el mismo **Precio** y el mismo **Peso**, dos `Esmeraldas` son iguales solo si tienen la misma **dureza** y el mismo **Precio**, mientras que dos Rubies son iguales si tienen el mismo **Precio** y 2 zafiros son iguales si tienen el mismo **Color** y el mismo **Precio**. Como es de suponer los Diamante Esmeraldas Rubi y Zafiros son distintos entre sí.
NOTA: Dejar como comentario en el código dos ejemplos en los que probaste la igualdad.

d) Definir la función, hay dos piedras distintas de manera consecutiva en una lista de `PiedrasPreciosas`. La función `hayDosDistintas,` tiene la siguiente definición de tipos:

```
hayDosDistintas :: [PiedrasPreciosas] -> Bool
```

Dada una lista de `PiedrasPreciosas lm,` debe devolver `True` en caso que en la lista `lm` existan dos piedras que sean distintas de manera consecutiva, y `False` en caso contrario.
NOTA: Recordar que definiste la igualdad anteriormente. Dejar como comentario en el código dos ejemplos en los que probaste la función.

Ejercicio 2

Queremos hacer un programa, para que el dueño de una veterinaria pueda saber si puede darles el alta a sus pacientes animales.

a) Definir un tipo recursivo `EstadoMascotas`, que permite guardar el estado que tuvieron los animales en la veterinaria. El tipo `EstadoMascotas`, tendrá dos constructores:

   1) `EvolucionDeMascota`, que tiene 6 parámetros:

   - `String`, para el nombre y apellido de la mascota
   - `Raza` (tipo enumerado con las siguientes: Pastor, Labrador, Golden, Siveriano, Otro)
   - `Int` (estado, entre 1 y 10)
   - `Int` (estado, entre 1 y 10)
   - `Int` (estado entre 1 y 10)
   - `EstadoMascotas,` recursión con el resto de las mascotas.

   2) `NoHayMascotas`, que es un constructor sin parámetros, similar al de la lista vacía.

Las condiciones para poder obtener el alta de la veterinaria se describen a continuación, según los estados de las mascotas:

- Si la `mascota es Golden o Siveriano`, debe tener más de 7 en alguno de los 3 estados, y no haber tenido ningún estado menos de 6.
- Si la mascota es Pastor debe tener al menos un 7 en cada estado.
- Si es Labrador u Otra necesita al menos 8 en cada estado.

b) Programar la función seVaALaCasa, que toma como primer parámetro **`EstadoMascotas,`** y como segundo parámetro el `nombre` de la mascota de tipo **`String`** y retorna un valor de tipo **`Bool,`** indicando si la mascota con `nombre` es **se va a la casa o no (es decir obtiene el alta).**

```
seVaALaCasa :: EstadoMascotas -> String -> Bool
```

NOTA: Dejar como comentario un ejemplo donde hayas probado **seVaALaCasa** con un parámetro de tipo **`EstadoMascotas`** que tenga al menos 3 mascotas.

c) Programar la función **`devolverMejorEstado`** con la siguiente declaración:

```
devolverMejorEstado :: EstadoMascotas -> String -> Maybe Int
```

que toma una variable **`EstadoMascotas`**, y como segundo argumento un `nombre`, que identifica a la mascota, y en caso que la mascota esté en `la lista` (con estados a b c), retorna Just (el mayor de los estados) y `Nothing` en caso contrario.

NOTA: Dejar como comentario un ejemplo donde hayas probado la función.