Los **Tipos Abstractos de Datos (TAD)** se caracterizan por "esconder" la implementación de un tipo de dato que cumple con ciertas características que el que pensó en ese tipo de datos, decidió.

Por ejemplo, una Cola, donde los elementos entran y salen de la misma forma que en una cola de personas, en orden de llegada. Si inserto 2 elementos, y despues "pido" un elemento a la cola el que se me da, es el primero que entró.

Otro ejemplo es una Pila, donde la característica principal es que el último elemento en insertarse es el primero que puedo obtener.

Pero que significa eso de que tal o cual elemento es el que obtengo primero ? Significa que las funciones que se me proveen para acceder a los datos de el TAD en cuestión se ocupan de que esa sea la única realidad posible. Volviendo al ejemplo de la Cola, la cola provee un conjunto de funciones específico para acceder a sus datos (al ese conjunto de funciones le llamamos **Interfaz**) y esas funciones que nos dan no nos permiten hacer otra cosa que lo que el tipo de datos Cola establece:

- **queue**: inserta un elemento en la cola (cabe aclarar que aca no dije al final de la cola, porque realmente yo, como usuario de la cola, no se (Y NO TENGO PORQUE SABER) realmente cual es el orden o de que manera se insertan internamente). El hecho de decir "al final" solo implica que el elemento insertado mas reciéntemente será el último en sacarse.
- **firstQ** : de todos los elementos que haya en la cola, me devuelve el primero que se insertó en ella
- **dequeQ**: devuelve una cola con todos los elementos que la cola que vino por parámetro, menos el primero (o sea, menos el que devolvería firstQ)
- **emptyQ** : devuelve una cola sin ningún elemento
- **isEmptyQ** : determina si una cola está vacía o no.

Cabe destacar, que en las descripciones que di arriba de las funciones de la interfaz de cola, siempre hablo del orden en el que se insertaron los elementos en términos de "linea de tiempo" , o sea, el orden que me importa es cual elemento se insertó primero y cual despues; y NUNCA hablo de COMO los guardo internamente. Podría guardarlos en una lista, en un árbol, etc, podría guardarlos ordenados por cualquier criterio que se me cante, y daría igual, porque la única manera de acceder a los elementos de un TAD, es por sus funciones de interfaz, que son las que me aseguran que el elemento que voy a obtener va a responder a las características del tipo de datos.

Ahora toda esta intro sirve para que quede claro que hay 2 aspectos claramente diferentes que hay que tener en mente al pensar en TADs:

- por un lado qué puede hacer un usuario usando la interfaz,
- y por el otro aspecto: COMO está implementado.

El cómo está implementado es una cuestión interna del TAD y nunca le debe importar al usuario. Dentro de la implementación hay un par de conceptos que hay que conocer y tener muy en claro :

- la **REPRESENTACIÓN** es qué tipos de elementos voy a usar para representar internamente este tipo de datos.
  Por ejemplo, una Pila también podríamos representarla naturalmente con una lista en Haskell. Una cola también podríamos representarla con una lista.
  La representación, en Haskell particularmente, en general la denotamos usando un data , de la siguiente manera:

```
data Cola a =          C                  [a]
               |--------------|   |----------------|
                 funcion  de           tipo de
                  abstracción       representación
```

esto dice que va a existir un nuevo tipo de datos que vamos a identificar con la palabra Cola.

La **función de abstracción** (que sería el constructor del data) es lo que identifica intermanete a nuestro tipo de datos para poder desarmarlo usando pattern matching. De esta manera, COMO IMPLEMENTADOR (y NO COMO USUARIO) yo puedo "desarmar" el tipo de datos "pattern-matcheando" (C [ ]) o (C (x:xs)), por ejemplo.

El **tipo de representación** es el punto clave , que es básciamente el (o los) elemento que contienen los datos, justamente. En este caso es una lista, pero podría ser una combinación de varias cosas. Por ejemplo si queremos hacer una cola especial que sepa cuantos elementos de los insertados son números pares y cuantos impares, por ejemplo podríamos tener una representación que guarde esa información, para obtenerla inmediatamente. por ej:

```
data ColaParesImpares =    C            [ Int ]                  Int                  Int
                                |- lista de elementos -|  |- cant de pares -| |- cant de impares -|
                                |-    de la cola      -|

                               |-------------------------------------------------------------------|
                                                Tipo de representación
```

Obviamente esta Cola particular debería proveer al usuario de dos funciones que le devuelvan esta información , ya que como dije antes, el usuario de un TAD no puede ver la representación interna del tipo de datos. Por ej:

```
-- paresC :: ColaParesImpares -> Int
-- imparesC :: ColaParesImpares -> Int
```

- los **INVARIANTES de REPRESENTACIÓN**: Si el tipo de representación son los elementos que "representan" internamente al tipo de datos, los invariantes son las condiciones que ponemos los que elegimos la representación , que queremos que se cumplan a medida que los datos en la representación se van modificando. Es decir, si para que las características del TAD sigan siendo válidas , los datos que la representan deben ser coherentes.

  Por ejemplo, dada la ColaParesImpares que puse de ejemplo anteriormente, sabemos que cuando creemos una nueva, los datos van a ser los siguientes:

  `C [] 0 0`
  -- La cola está vacía, por lo tanto la lista que contiene a sus elementos está vacía tmb (`[]`) , y los contadores de elementos pares e impares están en `0`.

  `--` Si hago un queue 5 a la cola, entonces el contador de numeros impares debería incrementarse en uno.
  `C [`**`5`**`] 0` **`1`**

  -- Si hago un queue 4 el contador de pares debería también incrementarse en uno
  `C [`**`4,`**`5] `**`1`**` 1`

  -- Si hago un dequeue de la cola que tiene esos datos debería decrementarse el contador de números impares en uno (porque el elemento que se quita es el 5)
  `C `**`[`**`4 `**`]`**` 1 `**`0`**

  `--` Si hago otro dequeue de la cola que tiene esos datos debería decrementarse el contador de números pares en uno
  `C `**`[] 0`**` 0`

  O sea, que de aca puedo deducir que las condiciones que se tienen que cumplir para que los datos de mi tipo de representación tengan sentido es que la cantidad de números pares que hay en la lista de elementos, tiene que ser el mismo número que el designamos como contador de los pares en el tipo de representación. De la misma manera pasaría con los impares y el otro contador. En otras palabras, y apenas mas formal:

  ```
  Sea (C l cp ci) la representación de una ColaParesImpares :
     * cp es el número de elementos pares que hay en la lista 'l'
     * ci es el número de elementos impares que hay en la lista 'l'
  ```

  Esos serían nuestros invariantes de representación para la cola `ColaParesImpares` que nos aseguran que cuando implementamos las funciones

  Los invariantes los usamos como reglas que debemos respetar al implementar las funciones de la interfaz del TAD. Si podemos garantizar que todos los invariantes se cumplen en la implementación de estas funciones, (y que además los invariantes fueron

correctamente elegidos), entonces el TAD no debería fallar.

Para distinguir si algo es un invariante o no, hay que fijarse si se trata de una propiedad que se puede verificar de manera independiente por sí o no cuando solo tengo una instancia y aunque solo tenga el tipo de representación y no mencione al tipo abstracto.

Entonces, si dijeramos:

- *el elemento se agrega por adelante*

NO es algo que se pueda verificar dada una cola cualquiera, sino una observación de cómo esperamos que se modifique la cola. (¿Cómo verificamos esto si yo te doy la instancia "C [7,5,4,3,9,2] 2 4"?)

En cambio

- *cp es el número de elementos impares en l*

es verificable (y en el ejemplo anterior es verdadero).

OJO con esta confusión porque se califica en el parcial!!

El tipo de representacióny los invariantes además se eligen con el fin de variar el "orden de complejidad" de las diferentes operaciones del TAD. Por ejemplo, si yo hubiera querido que agregar un elemento en la cola sea una operación de tiempo constante (**O(1)**) , me hubiera convenido que los elementos se agreguen a la izquierda en la lista, ya que agregar un elemento a la lista a la izquierda lo hago en O(1) mientras que agregarlo a la derecha me obliga a recorrer toda la lista (con append) , haciendo que la operación sea lineal en vez de constante.

A su vez , ColaParesImpares, podría ofrecer las funciones `paresC` e `imparesC` sin necesidad de poner los contadores '`cp`' y '`ci`' que puse en la representación, pero eso haría que tenga que recorrer toda la lista en busca de los pares o impares cada vez que se llame a esas funciones, haciéndolas lineales (**O(n)**). En cambio, mantener el número de los contadores me hace esas dos operaciones constantes, ya que solo tengo que devolver el número del contador apropiado, y por supuesto, para que los números sean correctos, se tienen que respetar los invariantes de representación.