# Diseño I: Elección de estructuras

## Clase práctica

Algoritmos y Estructuras de Datos II

# Diferencias entre especificación y diseño

- En la etapa de especificación:
  - Nos ocupamos del '¿Qué?'
  - Lo explicamos usando TADs
  - Ese '¿Qué?' se explica bajo el paradigma funcional

- En la etapa de diseño:
  - Nos ocupamos del '¿Cómo?'
  - Lo explicamos con *módulos de abstracción*
  - Ese '¿Cómo?' lo explicaremos usando el paradigma imperativo
  - Tenemos un *contexto de uso* que nos fuerza a tomar decisiones respecto de la estructura.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.
   - Postcondición: lo que debe cumplirse después de la ejecución.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.
   - Postcondición: lo que debe cumplirse después de la ejecución.
   - Descripción (Importante!).

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.
   - Postcondición: lo que debe cumplirse después de la ejecución.
   - Descripción (Importante!).
   - Complejidad.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.
   - Postcondición: lo que debe cumplirse después de la ejecución.
   - Descripción (Importante!).
   - Complejidad.
   - Aliasing.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.
   - Postcondición: lo que debe cumplirse después de la ejecución.
   - Descripción (Importante!).
   - Complejidad.
   - Aliasing.

2. Representación: sección no accesible a los usuarios del módulo que detalla la estructura de representación y los algoritmos justificando elección de estructuras así y complejidad de los algoritmos. Define *Rep* y *Abs*.

# Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

1. Interfaz: sección accesible a los usuarios del módulo (e.g., otros módulos) que detalla cada operación exportada:
   - Signatura: que recibe y que devuelve.
   - Precondición: condiciones sobre los input.
   - Postcondición: lo que debe cumplirse después de la ejecución.
   - Descripción (Importante!).
   - Complejidad.
   - Aliasing.

2. Representación: sección no accesible a los usuarios del módulo que detalla la estructura de representación y los algoritmos justificando elección de estructuras así y complejidad de los algoritmos. Define *Rep* y *Abs*.

3. Servicios usados: detalla los supuestos sobre los servicios usados de otros módulos. Estos requisitos justifican las complejidades de la sección anterior (que a su vez justifican las de la interfaz).
   - Ejemplo: conjunto implementado sobre una lista enlazada.

# Apuntes

- No se olviden de los apuntes:
  - Apunte de diseño.
  - Apunte de módulos básicos (módulos completos).

# Apuntes

- No se olviden de los apuntes:
  - Apunte de diseño.
  - Apunte de módulos básicos (módulos completos).

- Veamos un poco estos apuntes...

# Apuntes

- No se olviden de los apuntes:
  - Apunte de diseño.
  - Apunte de módulos básicos (módulos completos).
- Veamos un poco estos apuntes...
- En general no les vamos a pedir módulos completos.

# Apuntes

- No se olviden de los apuntes:
  - Apunte de diseño.
  - Apunte de módulos básicos (módulos completos).
- Veamos un poco estos apuntes...
- En general no les vamos a pedir módulos completos.
- Sólo por hoy: diseñar un módulo completo.

# Ejercicio: ¡Subite!

Sistema de tarjetas de transporte para el subte:

- En las boleterías, se venden tarjetas de 1, 2, 5, 10 y 30 viajes
- En la entrada a los andenes, hay molinetes con lectores de tarjetas
- Cuando el usuario pasa su tarjeta por el lector:
  - Si la tarjeta está agotada o es inválida, se informa de esto en el visor y no se abre el molinete
  - Si hay viajes en la tarjeta
    - Se abre el molinete
    - Se muestra el saldo en el visor
    - Se actualiza en el sistema el nuevo saldo para esa tarjeta
    - Se registra en el sistema el día y la hora del acceso

# Ejercicio: ¡Subite!

Sistema de tarjetas de transporte para el subte:

- En las boleterías, se venden tarjetas de 1, 2, 5, 10 y 30 viajes
- En la entrada a los andenes, hay molinetes con lectores de tarjetas
- Cuando el usuario pasa su tarjeta por el lector:
  - Si la tarjeta está agotada o es inválida, se informa de esto en el visor y no se abre el molinete
  - Si hay viajes en la tarjeta
    - Se abre el molinete
    - Se muestra el saldo en el visor
    - Se actualiza en el sistema el nuevo saldo para esa tarjeta
    - Se registra en el sistema el día y la hora del acceso

- Nos piden tiempo de acceso sublineal (en peor caso y en la cantidad total de tarjetas) para la operación de usar la tarjeta.

# TAD

**TAD** SUBITE

**observadores básicos**

tarjetas : subite $\longrightarrow$ conj(tarjeta)

crédito : subite $s$ $\times$ tarjeta $t$ $\longrightarrow$ nat $\quad \{t \in tarjetas(s)\}$

viajes : subite $s$ $\times$ tarjeta $t$ $\longrightarrow$ secu(FechaHora) $\quad \{t \in tarjetas(s)\}$

**generadores**

Crear : $\longrightarrow$ subite

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\quad \{c \in \{1,2,5,10,30\}\}$

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in tarjetas(s) \land crédito(s,t) > 0\}$

**otras operaciones**

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\quad \{\#tarjetas(s) > 0\}$

**Fin TAD**

# TAD

**TAD** SUBITE

**observadores básicos**

tarjetas : subite $\longrightarrow$ conj(tarjeta)

crédito : subite $s$ $\times$ tarjeta $t$ $\longrightarrow$ nat $\quad \{t \in tarjetas(s)\}$

viajes : subite $s$ $\times$ tarjeta $t$ $\longrightarrow$ secu(FechaHora) $\quad \{t \in tarjetas(s)\}$

**generadores**

Crear : $\longrightarrow$ subite

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\quad \{c \in \{1,2,5,10,30\}\}$

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in tarjetas(s) \land crédito(s,t) > 0\}$

**otras operaciones**

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\quad \{\#tarjetas(s) > 0\}$

**Fin TAD**

**Observación:** ultimaTarjeta devuelve la última tarjeta creada por NuevaTarjeta.

# Diseñanado...

¿Qué hacemos ahora?

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
  $\hookrightarrow$ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
  $\hookrightarrow$ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
  ↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
  ↪ ¡Sí! No siempre salen directamente del TAD...

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
  ↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
  ↪ ¡Sí! No siempre salen directamente del TAD...

- Definamos entonces las operaciones y para cada una definir:
  - **Descripción**
  - Precondición
  - Postcondición
  - Complejidad (!)
  - Aliasing (!)

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
  $\hookrightarrow$ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
  $\hookrightarrow$ ¡Sí! No siempre salen directamente del TAD...

- Definamos entonces las operaciones y para cada una definir:
  - **Descripción**
  - Precondición
  - Postcondición
  - Complejidad (!)
  - Aliasing (!)

- ¿Cómo elegimos entonces las operaciones?

# Diseñanado...

¿Qué hacemos ahora?

- ¿Elegir estructura o definir la interfaz?
  ↪ La interfaz será la misma sin importar la estructura, y además nos puede dar ideas para esta última...

- ¿Hay que pensar cuáles serán las operaciones provistas por la interfaz?
  ↪ ¡Sí! No siempre salen directamente del TAD...

- Definamos entonces las operaciones y para cada una definir:
  - **Descripción**
  - Precondición
  - Postcondición
  - Complejidad (!)
  - Aliasing (!)

- ¿Cómo elegimos entonces las operaciones?
  ↪ Depende de muchas cosas (principalmente el contexto de uso).

# Diseñanado...

Antes que nada arrancamos con la cabecera:

- **Módulo**: Subite
- **Se explica con**: Subite
- **Géneros**: Subite
- **Operaciones**: ...

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura:

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre:

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post:

# Subite: Creación y nuevas tarjetas

**Crear : $\longrightarrow$ subite**

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción:

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\{\#\text{tarjetas}(s) > 0\}$

- Signatura:

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\{\#\text{tarjetas}(s) > 0\}$

- Signatura: NuevaTarjeta(inout s: Subite, in c: Nat) $\rightarrow$ res: Tarjeta
- Pre:

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\{\#\text{tarjetas}(s) > 0\}$

- Signatura: NuevaTarjeta(inout s: Subite, in c: Nat) $\rightarrow$ res: Tarjeta
- Pre: { $s = s_0 \wedge c \in \{1, 2, 5, 10, 30\}$ }

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\{\#\text{tarjetas}(s) > 0\}$

- Signatura: NuevaTarjeta(inout s: Subite, in c: Nat) $\rightarrow$ res: Tarjeta
- Pre: $\{ s = s_0 \wedge c \in \{1, 2, 5, 10, 30\} \}$
- Post:

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\quad \{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\quad \{\#\text{tarjetas}(s) > 0\}$

- Signatura: NuevaTarjeta(inout s: Subite, in c: Nat) $\rightarrow$ res: Tarjeta
- Pre: { $s = s_0 \land c \in \{1, 2, 5, 10, 30\}$ }
- Post: { $res \notin tarjetas(s_0) \land res \in tarjetas(s) \land s = NuevaTarjeta(s_0, c)$ }

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\quad \{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\quad \{\#\text{tarjetas}(s) > 0\}$

- Signatura: NuevaTarjeta(inout s: Subite, in c: Nat) $\rightarrow$ res: Tarjeta
- Pre: $\{ s = s_0 \land c \in \{1, 2, 5, 10, 30\} \}$
- Post: $\{ res \notin tarjetas(s_0) \land res \in tarjetas(s) \land s = NuevaTarjeta(s_0, c) \}$
- Descripción: Agrega una nueva tarjeta al sistema y la retorna.

# Subite: Creación y nuevas tarjetas

Crear : $\longrightarrow$ subite

- Signatura: Crear() $\rightarrow$ res: Subite
- Pre: { true }
- Post: { res = Crear } (¿sombrerito?)
- Descripción: Crea una nueva instancia
- Complejidad: La definiremos más adelante.
- Aliasing: (no aplica)

NuevaTarjeta : subite $s$ $\times$ nat $c$ $\longrightarrow$ subite $\{c \in \{1,2,5,10,30\}\}$

ultimaTarjeta : subite $s$ $\longrightarrow$ tarjeta $\{\#\text{tarjetas}(s) > 0\}$

- Signatura: NuevaTarjeta(inout s: Subite, in c: Nat) $\rightarrow$ res: Tarjeta
- Pre: $\{ s = s_0 \land c \in \{1, 2, 5, 10, 30\} \}$
- Post: $\{ res \notin tarjetas(s_0) \land res \in tarjetas(s) \land s = NuevaTarjeta(s_0, c) \}$
- Descripción: Agrega una nueva tarjeta al sistema y la retorna.
- Complejidad: La definiremos más adelante.
- Aliasing: (E.g) Se devulve una referencia no modificable a la nueva tarjeta.

# Subite: Usar tarjeta

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

Supongamos que el contexto de uso nos indica lo siguiente:

- Queremos que UsarTarjeta indique si la misma era válida o no.
- Y además, de usarla, que nos devuelva el nuevo saldo de la tarjeta.

¿Podemos alterar la signatura para mejorar la usabilidad del módulo?

# Subite: Usar tarjeta

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:

# Subite: Usar tarjeta

$$\text{UsarTarjeta} : \text{subite } s \times \text{tarjeta } t \times \text{FechaHora} \longrightarrow \text{subite}$$

$$\{\text{t} \in \text{tarjetas(s)} \wedge \text{crédito(s,t)} > 0\}$$

- Signatura:
  UsarTarjeta(inout s: Subite, in t: Tarjeta, in h: FechaHora, **out** c: Nat)
  $\rightarrow$ res: Bool
- Pre:

# Subite: Usar tarjeta

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:
  UsarTarjeta(inout s: Subite, in t: Tarjeta, in h: FechaHora, **out** c: Nat)
  $\rightarrow$ res: Bool
- Pre: $\{ s = s_0 \}$

# Subite: Usar tarjeta

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:
  UsarTarjeta(inout s: Subite, in t: Tarjeta, in h: FechaHora, **out** c: Nat)
  $\rightarrow$ res: Bool
- Pre: $\{ s = s_0 \}$
- Post:

# Subite: Usar tarjeta

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{$t $\in$ tarjetas(s) $\wedge$ crédito(s,t) $>$ 0$\}$

- Signatura:
  UsarTarjeta(inout s: Subite, in t: Tarjeta, in h: FechaHora, **out** c: Nat)
  $\rightarrow$ res: Bool
- Pre: $\{\ s = s_0\ \}$
- Post: $\{\ res = (t \in tarjetas(s_0) \wedge \text{crédito}(s_0, t) > 0) \wedge_{\text{L}}$
  $(res \Rightarrow_{\text{L}} s = UsarTarjeta(s_0, t, h) \wedge \text{c} = \text{crédito}(s, t)) \wedge (\neg res \Rightarrow s = s_0)\ \}$

# Subite: Usar tarjeta

UsarTarjeta : subite $s$ $\times$ tarjeta $t$ $\times$ FechaHora $\longrightarrow$ subite

$\{t \in \text{tarjetas}(s) \wedge \text{crédito}(s,t) > 0\}$

- Signatura:
  UsarTarjeta(inout s: Subite, in t: Tarjeta, in h: FechaHora, **out** c: Nat)
  $\rightarrow$ res: Bool
- Pre: $\{\ s = s_0\ \}$
- Post: $\{\ res = (t \in tarjetas(s_0) \wedge \text{crédito}(s_0, t) > 0) \wedge_{\text{L}}$
  $(res \Rightarrow_{\text{L}} s = UsarTarjeta(s_0, t, h) \wedge\ c = \text{crédito}(s, t)) \wedge (\neg res \Rightarrow s = s_0)\ \}$
- Descripción: Si la tarjeta pasada por parámetro está registrada y puede utilizarla, entonces actualiza su crédito y retorna true
- Complejidad: tiene que ser $< O(n)$, pero lo precisamos luego.
- Aliasing: idem...

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{t \in \text{tarjetas}(s)\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{t \in \text{tarjetas}(s)\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
  - Viajes(in s: Subite, in t: Tarjeta, out vs: Secu(FechaHora)) $\rightarrow$ res: Bool

# Subite: Ver viajes y crédito

$$\text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} \quad \{t \in \text{tarjetas}(s)\}$$
$$\text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} \quad \{t \in \text{tarjetas}(s)\}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
  - Viajes(in s: Subite, in t: Tarjeta, out vs: Secu(FechaHora)) $\rightarrow$ res: Bool
  - Pre:

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
  - Viajes(in s: Subite, in t: Tarjeta, out vs: Secu(FechaHora)) $\rightarrow$ res: Bool
  - Pre: { true }

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
  - Viajes(in s: Subite, in t: Tarjeta, out vs: Secu(FechaHora)) $\rightarrow$ res: Bool
  - Pre: { true }
  - Post:

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
  - Viajes(in s: Subite, in t: Tarjeta, out vs: Secu(FechaHora)) $\rightarrow$ res: Bool
  - Pre: { true }
  - Post: { $(res = t \in tarjetas(s)) \wedge_{\mathrm{L}}$ (res $\Rightarrow_{\mathrm{L}}$ vs = viajes$(s, t)$) }

# Subite: Ver viajes y crédito

| | | |
|---|---|---|
| viajes : subite $s$ × tarjeta $t$ | $\longrightarrow$ secu(FechaHora) | {t ∈ tarjetas(s)} |
| crédito : subite $s$ × tarjeta $t$ | $\longrightarrow$ nat | {t ∈ tarjetas(s)} |

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver viajes:
  - Viajes(in s: Subite, in t: Tarjeta, out vs: Secu(FechaHora)) → res: Bool
  - Pre: { true }
  - Post: { $(res = t \in tarjetas(s)) \wedge_{\mathrm{L}}$ (res $\Rightarrow_{\mathrm{L}}$ vs = viajes$(s, t)$) }
  - Descripción: Si la tarjeta pasada por parámetro está registrada, retorna true y asigna en *viajes* los viajes realizados
  - Complejidad: La definimos más adelante
  - Aliasing: (E.g) Se devuelve una referencia no modificable a la lista de viajes realizados.

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{t \in \text{tarjetas}(s)\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{t \in \text{tarjetas}(s)\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
  - Crédito(in s: Subite, in t: Tarjeta, out cred: nat) $\rightarrow$ res: Bool

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
  - Crédito(in s: Subite, in t: Tarjeta, out cred: nat) $\rightarrow$ res: Bool
  - Pre:

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{t \in \text{tarjetas}(s)\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{t \in \text{tarjetas}(s)\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
  - Crédito(in s: Subite, in t: Tarjeta, out cred: nat) $\rightarrow$ res: Bool
  - Pre: { true }

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
  - Crédito(in s: Subite, in t: Tarjeta, out cred: nat) $\rightarrow$ res: Bool
  - Pre: { true }
  - Post:

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
  - Crédito(in s: Subite, in t: Tarjeta, out cred: nat) $\rightarrow$ res: Bool
  - Pre: { true }
  - Post: { $(res = t \in tarjetas(s)) \wedge_{\mathrm{L}}$ (res $\Rightarrow_{\mathrm{L}}$ cred = crédito$(s, t)$) }

# Subite: Ver viajes y crédito

$$\begin{array}{lll} \text{viajes} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{secu(FechaHora)} & & \{\text{t} \in \text{tarjetas(s)}\} \\ \text{crédito} : \text{subite } s \times \text{tarjeta } t \longrightarrow \text{nat} & & \{\text{t} \in \text{tarjetas(s)}\} \end{array}$$

Podemos hacer algo parecido para ver los viajes y el crédito....

- Ver crédito:
  - Crédito(in s: Subite, in t: Tarjeta, out cred: nat) $\rightarrow$ res: Bool
  - Pre: { true }
  - Post: { $(res = t \in tarjetas(s)) \wedge_{\mathrm{L}}$ (res $\Rightarrow_{\mathrm{L}}$ cred = crédito$(s, t)$) }
  - Descripción: Si la tarjeta pasada por parámetro está registrada, retorna true y asigna el crédito en *cred* si no retorna false
  - Complejidad: Libre, al no tener restricciones podemos elegirla más adelante
  - Aliasing: -

# Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)

# Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
  - `estr` es `Dicc(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: Secuencia(FechaHora)>`

# Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
  - `estr` es `Dicc(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: Secuencia(FechaHora)>`
- ¿Así cumple con las complejidades requeridas?

# Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
  - `estr` es `Dicc(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: Secuencia(FechaHora)>`
- ¿Así cumple con las complejidades requeridas?
- Recordemos que utilizar la tarjeta implica:
  - **Buscar la tarjeta**
  - Descontarle el saldo
  - Agregarle un viaje

# Subite: Elección de estructuras

- Elijamos la estructura, sabiendo que la única restricción de complejidad que tenemos es en utilizar la tarjeta ($< O(n)$)
- Subite se podría representar con `estr`, donde:
  - `estr` es `Dicc(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: Secuencia(FechaHora)>`
- ¿Así cumple con las complejidades requeridas?
- Recordemos que utilizar la tarjeta implica:
  - **Buscar la tarjeta**
  - Descontarle el saldo
  - Agregarle un viaje
- Escribamos el algoritmo y veamos...

# Subite: Algoritmo de UsarTarjeta

```
function iUsarTarjeta(inout s: estr, in i: Tarjeta, out crédito: Nat)
    if ¬definido(e, i) then                                   ▷ O(definido)
        res ← false
    else
        datos ← obtener(e, i)                                 ▷ O(obtener)
        if datos.credito = 0 then
            res ← false
        else
            datos.credito ← datos.credito–1
            agregar(datos.viajes, FechaHoraActual)            ▷ O(agregar)
            credito ← datos.credito
            res ← true
        end if
    end if
    devolver res
end function
```

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)
- ¿Y el diccionario?
  - Vectores de tuplas

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)
- ¿Y el diccionario?
  - Vectores de tuplas
  - Tabla de Hash (lo van a ver en la teórica y en el labo):
    Permite buscar «casi» en $O(1)$

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)
- ¿Y el diccionario?
  - Vectores de tuplas
  - Tabla de Hash (lo van a ver en la teórica y en el labo):
    Permite buscar «casi» en $O(1)$
  - Árbol ABB:
    Si agregan con una distribución uniforme permite buscar en $O(log\ n)$

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)
- ¿Y el diccionario?
  - Vectores de tuplas
  - Tabla de Hash (lo van a ver en la teórica y en el labo):
    Permite buscar «casi» en $O(1)$
  - Árbol ABB:
    Si agregan con una distribución uniforme permite buscar en $O(log\ n)$
  - Árbol AVL:
    Permite buscar en $O(log\ n)$

# Ejercicio: Requisitos de complejidad

- Ok, entonces `definido`, `obtener` y `agregar` tienen que ser sublineales...
- Para el agregar, si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)
- ¿Y el diccionario?
  - Vectores de tuplas
  - Tabla de Hash (lo van a ver en la teórica y en el labo):
    Permite buscar «casi» en $O(1)$
  - Árbol ABB:
    Si agregan con una distribución uniforme permite buscar en $O(log\ n)$
  - Árbol AVL:
    Permite buscar en $O(log\ n)$
- Si elegimos implementar la secuencia con una lista enlazada y el diccionario con un árbol AVL, entonces... touché!

```
function iUsarTarjeta(inout s: Subite, in i: Tarjeta, out crédito: Nat)
return res
    if ¬definido(e, i) then                                          ▷ O(log n)
        res ← false
    else
        datos ← obtener(e, i)                                        ▷ O(log n)
        if datos.credito = 0 then
            res ← false
        else
            datos.credito ← datos.credito–1
            agregar(datos.viajes, FechaHoraActual)                   ▷ O(1)
            credito ← datos.credito
            res ← true
        end if
    end if
end function
```

# Subite: estructura elegida

Finalmente:

- Subite se representa con `estr`, donde:
  - `estr` es `DiccAVL(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: ListaEnlazada(FechaHora)>`
- Y donde `DiccAVL(Tarjeta, DatosTarjeta)` se representa con `AVL(Nodo)` donde
  - Nodo es <Tarjeta, DatosTarjeta>

# Subite: estructura elegida

Finalmente:

- Subite se representa con `estr`, donde:
  - `estr` es `DiccAVL(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: ListaEnlazada(FechaHora)>`
- Y donde `DiccAVL(Tarjeta, DatosTarjeta)` se representa con `AVL(Nodo)` donde
  - Nodo es <Tarjeta, DatosTarjeta>
- Deberíamos diseñar `DiccAVL`, pero eso lo damos por hecho... ;-)

# Subite: estructura elegida

Finalmente:

- `Subite` se representa con `estr`, donde:
  - `estr` es `DiccAVL(Tarjeta, DatosTarjeta)`
  - `Tarjeta` es `Nat`
  - `DatosTarjeta` es `Tupla<Crédito: Nat, Viajes: ListaEnlazada(FechaHora)>`
- Y donde `DiccAVL(Tarjeta, DatosTarjeta)` se representa con `AVL(Nodo)` donde
  - Nodo es <Tarjeta, DatosTarjeta>
- Deberíamos diseñar `DiccAVL`, pero eso lo damos por hecho... ;-)

- Restan completar el resto de los algoritmos y...
- Nos faltó el Invariante de Representación y la Función de Abstracción (tarea)

# Fin