Diseño I: Elección de estructuras

Clase práctica

Algoritmos y Estructuras de Datos II

Diferencias entre especificación y diseño

- En la etapa de especificación:

Diferencias entre especificación y diseño

- En la etapa de especificación:
  - Nos ocupamos del '¿Qué?'

Diferencias entre especificación y diseño

- En la etapa de especificación:
  - Nos ocupamos del '¿Qué?'
  - Lo explicamos usando TADs

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

Diferencias entre especificación y diseño

- En la etapa de diseño:

Diferencias entre especificación y diseño

- En la etapa de diseño:
  - Nos ocupamos del '¿Cómo?'

Diferencias entre especificación y diseño

- En la etapa de diseño:
  - Nos ocupamos del '¿Cómo?'
  - Lo explicamos con *módulos de abstracción*

Diferencias entre especificación y diseño

- En la etapa de diseño:
  - Nos ocupamos del '¿Cómo?'
  - Lo explicamos con *módulos de abstracción*
  - Ese '¿Cómo?' se explica usando el paradigma imperativo

Diferencias entre especificación y diseño

- En la etapa de diseño:
  - Nos ocupamos del '¿Cómo?'
  - Lo explicamos con *módulos de abstracción*
  - Ese '¿Cómo?' se explica 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:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - Signatura: que recibe y que devuelve.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - Signatura: que recibe y que devuelve.
  - Precondición: condiciones sobre los input.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - 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:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - Signatura: que recibe y que devuelve.
  - Precondición: condiciones sobre los input.
  - Postcondición: lo que debe cumplirse después de la ejecución.
  - Complejidad Temporal...

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - Signatura: que recibe y que devuelve.
  - Precondición: condiciones sobre los input.
  - Postcondición: lo que debe cumplirse después de la ejecución.
  - Complejidad Temporal...
  - Descripción (Importante!).

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - Signatura: que recibe y que devuelve.
  - Precondición: condiciones sobre los input.
  - Postcondición: lo que debe cumplirse después de la ejecución.
  - Complejidad Temporal...
  - Descripción (Importante!).
  - Aliasing: cuando tenemos más de un nombre que hace referencia a lo mismo, lease: 2 punteros o referencias hacia el mismo objeto.

Partes de un módulo

Un módulo de abstracción se divide en tres secciones:

- Interfaz: Es la sección accesible para los usuarios del módulo (Que bien pueden ser otros módulos). Aquí se detallan los servicios exportados. Cada operacion del módulo viene acompañado de:
  - Signatura: que recibe y que devuelve.
  - Precondición: condiciones sobre los input.
  - Postcondición: lo que debe cumplirse después de la ejecución.
  - Complejidad Temporal...
  - Descripción (Importante!).
  - Aliasing: cuando tenemos más de un nombre que hace referencia a lo mismo, lease: 2 punteros o referencias hacia el mismo objeto.

- Representación: Esta sección no es accesible a los usuarios del módulo. Aquí se detalla la elección de estructura de representación y los algoritmos. Se justifica la elección de las estructuras así como la complejidad de los algoritmos.

Partes de un módulo

- Servicios usados: Es la parte del módulo donde se detallan todas los supuestos sobre los servicios que exportan los otros módulos. Estos requisitos son los que justifican los análisis de complejidad que se hacen dentro de la sección Representación que a su vez justifican las promesas hechas en la interfaz.

Partes de un módulo

- Servicios usados: Es la parte del módulo donde se detallan todas los supuestos sobre los servicios que exportan los otros módulos. Estos requisitos son los que justifican los análisis de complejidad que se hacen dentro de la sección Representación que a su vez justifican las promesas hechas en la interfaz.

Por ejemplo, supongamos que implementamos un conjunto utilizando un arreglo:

- Para chequear la pertenencia al conjunto utilizamos la operación *está?* de secuencia. El módulo secuencia nos permite, con complejidad lineal, saber si un elemento está en la misma o no.

Vinculación especificación y diseño

¿Cómo se relaciona todo esto con la especificación? Necesitamos saber dos cosas:

Vinculación especificación y diseño

¿Cómo se relaciona todo esto con la especificación? Necesitamos saber dos cosas:

- ¿Qué valores pueden tomar las variables dentro de nuestra estructura para que esta sea válida?
  - Invariante de Representación (Rep): Es un predicado que expresa la sanidad de la estructura. Tiene que valer siempre en el momento inicial y final de cada función descrita en la interfaz.

$$\text{Rep}: \widehat{estr} \rightarrow boolean$$

Vinculación especificación y diseño

¿Cómo se relaciona todo esto con la especificación? Necesitamos saber dos cosas:

- ¿Qué valores pueden tomar las variables dentro de nuestra estructura para que esta sea válida?
  - Invariante de Representación (Rep): Es un predicado que expresa la sanidad de la estructura. Tiene que valer siempre en el momento inicial y final de cada función descrita en la interfaz.

$$\text{Rep}: \widehat{estr} \rightarrow boolean$$

- ¿Con qué instancia del TAD que estoy diseñando se vincula mi instancia de estructura de representación?

Vinculación especificación y diseño

¿Cómo se relaciona todo esto con la especificación? Necesitamos saber dos cosas:

- ¿Qué valores pueden tomar las variables dentro de nuestra estructura para que esta sea válida?
  - Invariante de Representación (Rep): Es un predicado que expresa la sanidad de la estructura. Tiene que valer siempre en el momento inicial y final de cada función descrita en la interfaz.

$$\text{Rep}: \widehat{estr} \rightarrow boolean$$

- ¿Con qué instancia del TAD que estoy diseñando se vincula mi instancia de estructura de representación?
  - Función de Abstracción (Abs): Vincula la imagen abstracta de una estructura con el valor abstracto al que representa.

$$\text{Abs}: \widehat{\texttt{estr}} \rightarrow tipo\_abstracto\_representado$$

Recordar: La función $\widehat{\ }$ toma una estructura del mundo del diseño y nos devuelve automágicamente su correspondiente instancia abstracta del mundo de los TADs.

Ejemplo completo

- 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

Ejemplo completo

- 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

- Se pide tiempo de acceso sublineal en la cantidad total de tarjetas (en caso promedio) para la operación usarTarjeta

TAD

TAD SUBITE

observadores básicos

tarjetas : subite $\longrightarrow$ conj(tarjeta)

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

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

generadores

Crear : $\longrightarrow$ subite

IncorporarTarjeta : subite s $\times$ tarjeta t $\times$ nat c $\longrightarrow$ subite

$\{t \notin \text{tarjetas}(s) \land c \in \{1,2,5,10,30\}\}$

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

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

Fin TAD

- Y ahora cómo la seguimos?

- Y ahora cómo la seguimos?
- Para elegir la estructura definamos correctamente la interfaz...
- La interfaz está compuesta por sus operaciones, hay que *definirlas*, y además brindar:
  - **Descripción**
  - Precondición
  - Postcondición
  - Complejidad
  - Aliasing

- Y ahora cómo la seguimos?
- Para elegir la estructura definamos correctamente la interfaz...
- La interfaz está compuesta por sus operaciones, hay que *definirlas*, y además brindar:
  - **Descripción**
  - Precondición
  - Postcondición
  - Complejidad
  - Aliasing

- Cómo elegimos las operaciones?

Preguntas?

Antes que nada arrancamos con la cabecera:

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

- Creación:
  - Crear() $\rightarrow$ res: Subite
  - Pre: Ninguna... o bien { true }
  - Post:

- Creación:
  - Crear() $\rightarrow$ res: Subite
  - Pre: Ninguna... o bien $\{$ true $\}$
  - Post: $\{$ res $=$ Crear $\}$
  - Complejidad: El ejercicio no lo aclara, pero podría ser: O(1)
  - Descripción: Crea una nueva instancia
  - Aliasing: -

- Agregar tarjeta:
  - NuevaTarjeta(inout s: Subite, in crédito: Nat) $\rightarrow$ res: Tarjeta
  - Pre:

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

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

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

- Agregar tarjeta:
  - NuevaTarjeta(inout s: Subite, in crédito: Nat) $\rightarrow$ res: Tarjeta
  - Pre: $\{\ s = s_0 \land \text{crédito} \in \{1, 2, 5, 10, 30\}\ \}$
  - Post: $\{\ res \notin tarjetas(s_0) \land s = \text{IncorporarTarjeta}(s_0, res, \text{crédito})\ \}$
  - Complejidad: Libre, al no tener restricciones podemos elegirla más adelante
  - Descripción: Agrega una nueva tarjeta al sistema

- Agregar tarjeta:
  - NuevaTarjeta(inout s: Subite, in crédito: Nat) $\rightarrow$ res: Tarjeta
  - Pre: $\{\ s = s_0 \land \text{crédito} \in \{1, 2, 5, 10, 30\}\ \}$
  - Post: $\{\ res \notin tarjetas(s_0) \land s = \text{IncorporarTarjeta}(s_0, res, \text{crédito})\ \}$
  - Complejidad: Libre, al no tener restricciones podemos elegirla más adelante
  - Descripción: Agrega una nueva tarjeta al sistema
  - Aliasing:

- Agregar tarjeta:
  - NuevaTarjeta(inout s: Subite, in crédito: Nat) $\rightarrow$ res: Tarjeta
  - Pre: $\{\ s = s_0 \wedge \text{crédito} \in \{1, 2, 5, 10, 30\}\ \}$
  - Post: $\{\ res \notin tarjetas(s_0) \wedge s = \text{IncorporarTarjeta}(s_0, res, \text{crédito})\ \}$
  - Complejidad: Libre, al no tener restricciones podemos elegirla más adelante
  - Descripción: Agrega una nueva tarjeta al sistema
  - Aliasing: res es un puntero a la nueva tarjeta, alcanzable desde s

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Usar tarjeta
  - UsarTarjeta(inout s: Subite, in i: Tarjeta, **out** crédito: Nat) $\rightarrow$ res: Bool

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Usar tarjeta
  - UsarTarjeta(inout s: Subite, in i: Tarjeta, **out** crédito: Nat) $\rightarrow$ res: Bool
  - Pre:

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Usar tarjeta
  - UsarTarjeta(inout s: Subite, in i: Tarjeta, **out** crédito: Nat) $\rightarrow$ res: Bool
  - Pre: $\{ s = s_0 \}$

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Usar tarjeta
  - UsarTarjeta(inout s: Subite, in i: Tarjeta, **out** crédito: Nat) $\rightarrow$ res: Bool
  - Pre: $\{ s = s_0 \}$
  - Post:

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Usar tarjeta
  - UsarTarjeta(inout s: Subite, in i: Tarjeta, **out** crédito: Nat) $\rightarrow$ res: Bool
  - Pre: $\{\ s = s_0\ \}$
  - Post: $\{\ (res \iff i \in tarjetas(s_0) \land \text{Crédito}(s_0, i) > 0) \land$
    $(s = \text{if res then UsarTarjeta}(s_0, i) \text{ else } s_0 \text{ fi}) \land$
    $(\text{res} \Rightarrow \text{crédito} = \text{Crédito}(\text{UsarTarjeta}(s_0, i)))\ \}$

- Supongamos que tanto para utilizar la tarjeta como para ver los viajes, quiero prescindir de conocer previamente si la tarjeta estaba en el sistema

- Usar tarjeta
  - UsarTarjeta(inout s: Subite, in i: Tarjeta, **out** crédito: Nat) $\rightarrow$ res: Bool
  - Pre: $\{ s = s_0 \}$
  - Post: $\{ (res \iff i \in tarjetas(s_0) \land \text{Crédito}(s_0, i) > 0) \land$
    $(s = \text{if res then UsarTarjeta}(s_0, i) \text{ else } s_0 \text{ fi}) \land$
    $(\text{res} \Rightarrow \text{crédito} = \text{Crédito}(\text{UsarTarjeta}(s_0, i))) \}$
  - Complejidad: tiene que ser $< O(n)$
  - Descripción: Si la tarjeta pasada por parámetro está registrada y puede utilizarla, entonces actualiza su crédito y retorna true
  - Aliasing: -

- Ver viajes:
  - VerViajes(in s: Subite, in i: Tarjeta, out viajes: Secu(FechaHora)) $\rightarrow$ res: Bool

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

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

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

- Ver viajes:
  - VerViajes(in s: Subite, in i: Tarjeta, out viajes: Secu(FechaHora)) $\rightarrow$ res: Bool
  - Pre: $\{$ true $\}$
  - Post: $\{$ $(res \iff i \in tarjetas(s)) \wedge$
    $(\text{res} \Rightarrow \text{viajes} = \text{Viajes}(s, i))$ $\}$

- Ver viajes:
  - VerViajes(in s: Subite, in i: Tarjeta, out viajes: Secu(FechaHora)) $\rightarrow$ res: Bool
  - Pre: $\{$ true $\}$
  - Post: $\{$ $(res \iff i \in tarjetas(s)) \wedge$
    $(\text{res} \Rightarrow \text{viajes} = \text{Viajes}(s, i))$ $\}$
  - Complejidad: Libre, al no tener restricciones podemos elegirla más adelante
  - Descripción: Si la tarjeta pasada por parámetro está registrada, retorna true y asigna en *viajes* los viajes realizados
  - Aliasing: -

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

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

- Subite se podría representar con 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...

```
function iUsarTarjeta(inout s: Subite, in i: Tarjeta, out crédito: Nat)
return res
    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
end function
```

- Las operaciones definido, obtener y agregar tienen que ser sublineales
- Qué estructuras conocemos?

- Las operaciones definido, obtener y agregar tienen que ser sublineales
- Qué estructuras conocemos?

- Si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)

- Las operaciones definido, obtener y agregar tienen que ser sublineales
- Qué estructuras conocemos?

- Si implementamos la secuencia con una lista enlazada, el agregar al final cuesta O(1)
- Y el diccionario?

- Posibles implementaciones:
  - Vectores de tuplas

- Posibles implementaciones:
  - 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 $O(log\ n)$

- Posibles implementaciones:
  - 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...

```
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
```

- Finalmente...

- Dicc(Tarjeta, DatosTarjeta) se representa con AVL(Nodo)
  Donde Nodo es <Tarjeta, DatosTarjeta>

- Secuencia(FechaHora) (en DatosTarjeta) se representa con ListaEnlazada(FechaHora)

- Finalmente...

- Dicc(Tarjeta, DatosTarjeta) se representa con AVL(Nodo)
  Donde Nodo es <Tarjeta, DatosTarjeta>

- Secuencia(FechaHora) (en DatosTarjeta) se representa con ListaEnlazada(FechaHora)

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

Fin.