Abstracciones de diseño

Abstracciones de diseño

- Abstracción por parametrización
- Abstracción por especificación
- Especificación e implementación de TADs
- Abstracciones genéricos
- Abstracciones de control

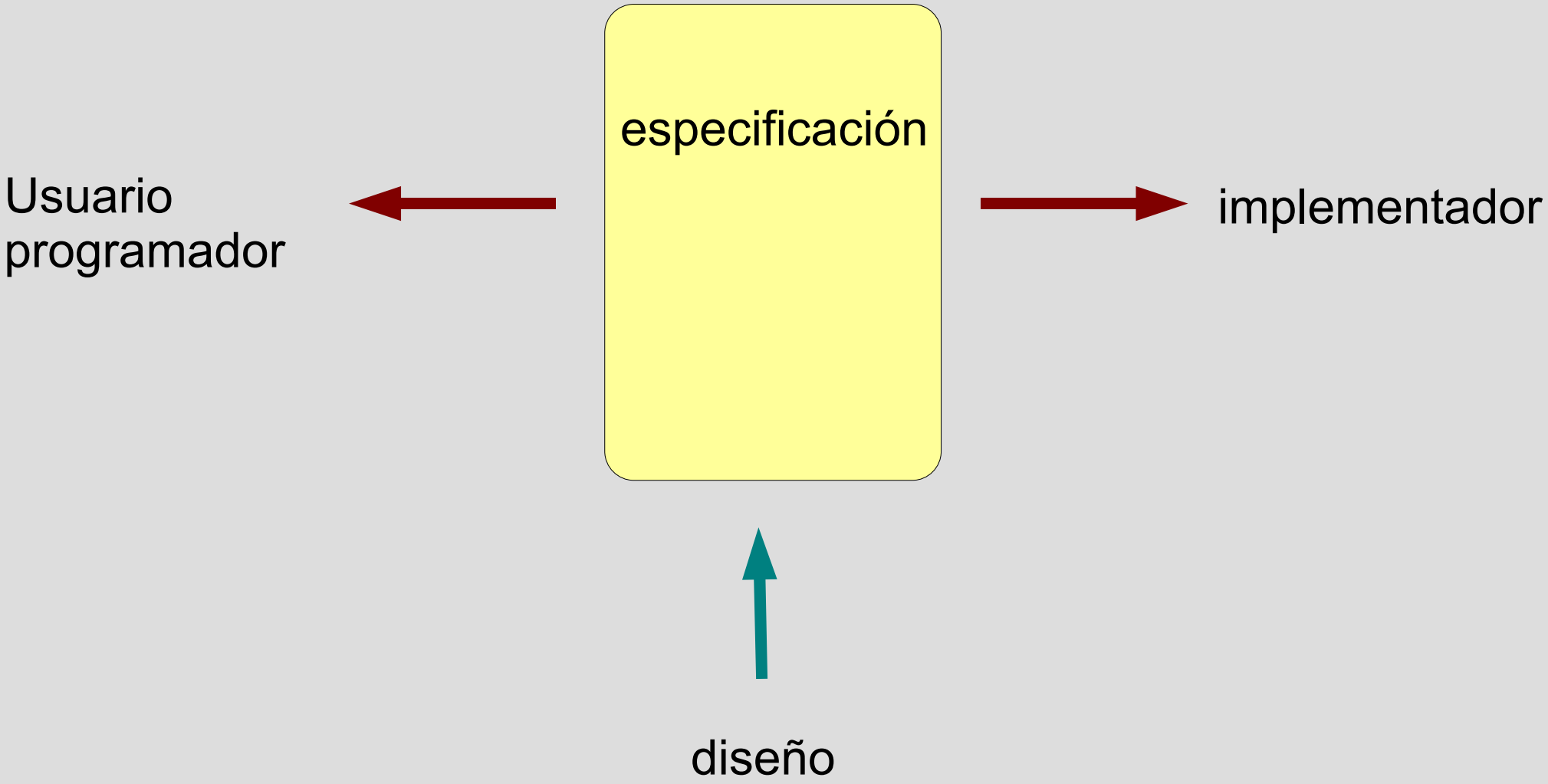
Abstracción por parametrización

- Por parametrización se obtienen las abstracciones procedimentales
- La definición de parámetros nos permite abstraer su valor concreto (actual)
- También abstraemos la particularidad de la ejecución concreta

Abstracción por especificación

- Mediante especificación abstraemos los detalles de implementación.
- Establecemos un contrato entre usuario e implementador
- Este contrato permite:
  - ◇ Al usuario centrarse en su problema
  - ◇ Al implementador despreocuparse de las posibles consecuencias de un mal uso de la implementación
  - ◇ A ambos cambiar de implementación

especificación - contrato



Especificación mediante...

- Pre-condiciones
  - Lo que debe cumplirse antes de la llamada
- Post-condiciones
  - Lo que se cumplirá después de la llamada, supuesta la precondición

¿Quién debe comprobar la precondición?

Especificación mediante...

- La precondición es responsabilidad del usuario
- La postcondición es responsabilidad del implementador

Especificación mediante...

int index(colecc X,tipo_elemento ele)

pre: *true*
post: si ele es uno de los valores contenidos en el X
devuelve el indice correspondiente, en caso
contrario devuelve el valor int(-1)

Especificación mediante...

float sqrt(float X)

pre: *X >= 0*
post: devuelve $(X)^{1/2}$

Abstracción por especificación

- *Abstracciones de datos*:
  - se especifica un TAD sin hacer referencia a la representación ni a la implementación de las operaciones
- Abstracción por especificación de procedimientos y funciones

Un estilo de especificación

Interface: int index(int ele)

requer.:
modif.:
efecto: si ele es uno de los valores contenidos en el array
devuelve el indice correspondiente, en caso contrario
devuelve el valor int(-1)

Estilo de especificación de TAD de B. Liskov
TAD: nombre
Descripción: .....
Operaciones:

- signatura -> información sintáctica

Pre-Condición: Debe cumplirla el programador usuario
Post-Condición:

Si a la entrada se cumplía la Pre-Condición, a la salida se cumplirá la Post-Condición

- int member (int ele)

requer.:
modifica:
efecto:

- .....

Estilo de implementación de Liskov

- La implementación debe asociar un tipo representación al tipo Abstracto
- Al tipo Abstracto pertenecen todos los posibles objetos compatibles con la especificación
- El tipo abstracto posee las operaciones especificadas

Estilo de implementación de Liskov

- Al tipo representación pertenecen todos los objetos representación de objetos abstractos

- El tipo rep:
  - Debe poseer implementaciones para las operaciones del tipo abstracto
  - Puede poseer algunas operaciones de uso interno en la implementación

Estilo de implementación...

- Para obtener una implementación de un TAD:
  - debe escogerse una representación
  - deben implementarse las operaciones del TAD
  - deben especificarse e implementarse las operaciones internas necesarias

Estilo de implementación..

- La elección de la representación implica:
  - Una declaración sintáctica
  - Un significado de la representación

- La implementación de las operaciones exige la elección de una política de implementación, acorde con la especificación del TAD y con el significado de la implementación.

Estilo de implementación..

- Es conveniente explicitar y documentar el significado de la representación y la política de implementación al comenzar esta;
- así podremos recordarlo a lo largo de los trabajos de implementación,
- y facilitaremos la coherencia de la implementación y de su mantenimiento

Estilo de implementación..

Función de abstracción

rep: A → *A*

asocia a cada objeto rep el objeto abstracto que representa

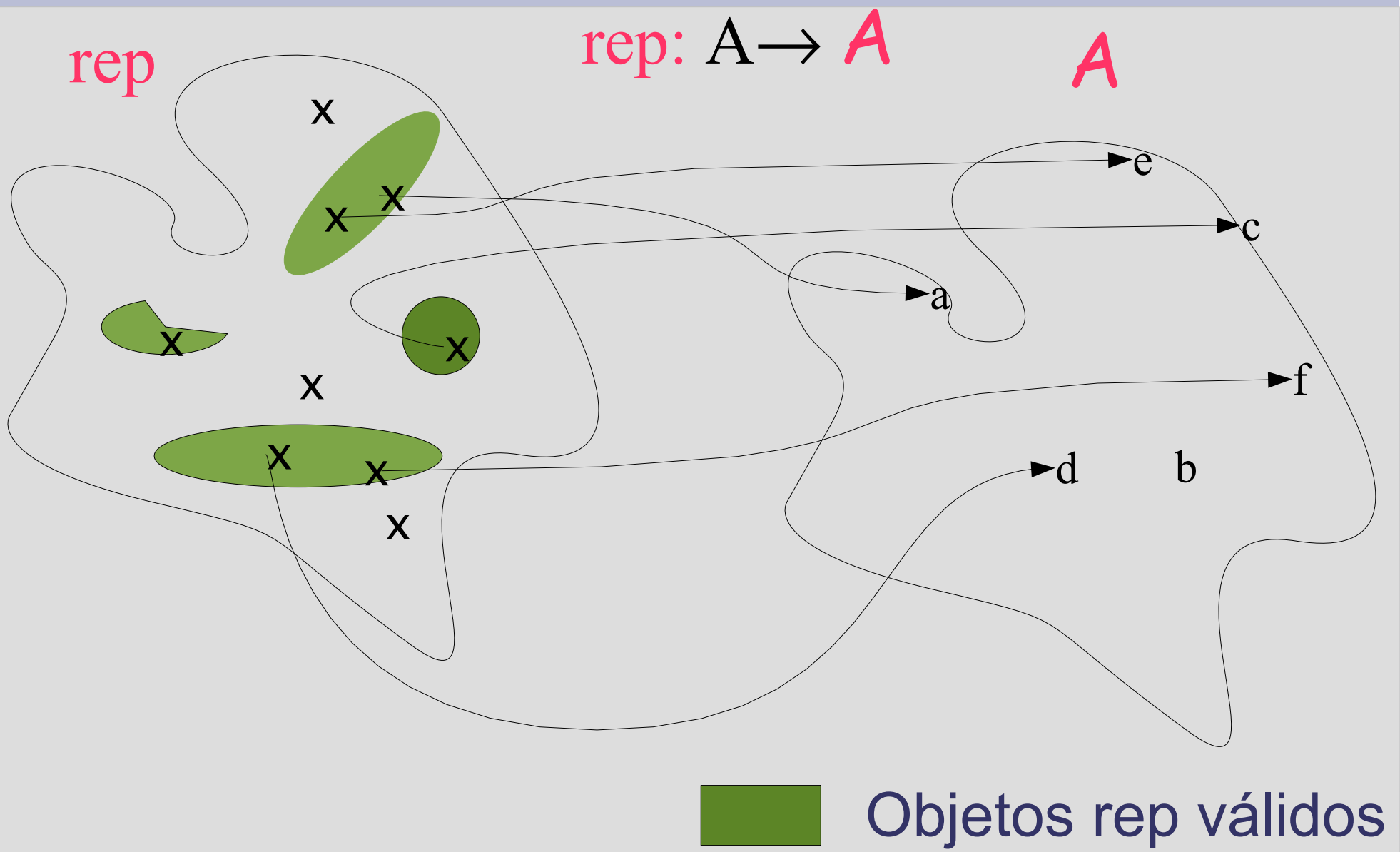
Estilo de implementación..

Invariante de la representación

I: rep $\rightarrow$ {*true* , *false*}

es una propiedad que deben cumplir todos los objetos rep válidos

Estilo de implementación..



Estilo de implementación..

- Un buen sitio para indicar la función de abstracción y el invariante de la representación es la implementación
- No es necesario adoptar un estilo formal para definirlas
- Sólo se trata de explicar el sentido de la representación y la política de implementación

Tad: int_set_100

Representacion:

Rep = int_set_100 data members:

{int cursize;

int members[100];}

Rep r;

F.Abstraccion: r representa al conjunto de todos los valores enteros de los primeros r.cursize elementos de r.members

I.Representacion: entre los primeros r.cursize valores de r.members no hay repeticiones

Abstracciones genéricas

- Los dos mecanismos de abstracción pueden combinarse a distintos niveles
- Las abstracciones genéricas se obtienen introduciendo un nivel extra de parametrización, p.e. mediante la definición de *tipos parámetro*
- Un tipo parámetro es un parámetro formal cuyos valores actuales son tipos

Abstracciones genéricas

- Abstrayendo los valores concretos de los tipos parametro damos generalidad a nuestras abstracciones

- Para hacer uso de las abstracciones genericas sera necesario instanciarlas, dando valores concretos a los par

Abstracciones genéricas

- La obtencion de codigo generico se basa en el uso de plantillas (templates)
- La generacion de codigo objeto eficiente antes de la instanciacion es problematica
- En un primer momento se pueden comprobar la coherencia del codigo generico, y las restricciones que tendra su uso
- Despues, tras comprobar las restricciones e instanciar los parametros se puede generar el codigo objeto

Abstracciones genéricas

- Al especificarlas se deben indicar las restricciones que ha de cumplir la instanciación
- Al especificar TAD genericos con el estilo de Liskov podria introducirse una clausula general de requerimientos de la instanciacion, p.e. a continuacion de

TAD: nombre<class tip, ...>
requer: que tip resida en el stack, y ....

Abstracciones de control

- Se obtienen abstrayendo los detalles del flujo de control,
- mediante parametrización y especificación
- Ejemplos
  - Iteradores
  - Manejo de excepciones
  - Mecanismos de concurrencia

iteradores

```
arbol<personas> arbol_genealogico;
...

foreach ele in arbol_genealogico
do
{
...
}
```

- Liskov los incorporó al lenguaje CLU

Tipos generadores

- Una forma de implementar iteradores
- A las colecciones (conjuntos, bags, árboles, listas, ...) se les asocian tipos generadores de iteraciones (*iteradores*)
- Puede haber varios para una misma colección, para llevar a cabo recorridos de distintos tipos (primero en profundidad, priemero en anchura)
- A veces, las iteraciones de llevan a cabo sobre copias de las colecciones, o se requiere su no modificación durante la iteración

Tipos generadores

- Operaciones:
  - Constructor (asocia el objeto iterador con el objeto colección)
  - Inicialización: begin(), end()
  - Avance:
    - ++ , next(), next_n()
    - -- , previous, previous_n()
    - object_at(posición) (posición dentro de la iteración)
  - Obtención del elemento
  - Detección del final atEnd()

```
template <class tipel,int deftam>

tipel Pila<tipel,deftam>::pop()
{ if (tope == 0) throw Vacia(); return v[--tope]; }
```

```
try
{ for(int i=1;i<20;i++) cout << p2.pop()<<'\n'; }
catch(...)
{ cout<<"problemas"<<'\n'; }
```