Abstracción

Abstracción
Objetivos

- Comprender la función de la abstracción en programación.
- Comprender los diferentes tipos de abstracción
- Comprender la abstracción procedimental.
- Comprender la abstracción de datos (TDA).
- Comprender el concepto de función de abstracción.
- Comprender el concepto de invariante de representación.

Abstracción

Comprender la función de la abstracción en programación

Ordenadores más avanzados

Programas más complejos

Programación estructurada

(Facilita el diseño, codificación y prueba)

Abstracción
Comprender la función de la abstracción en programación

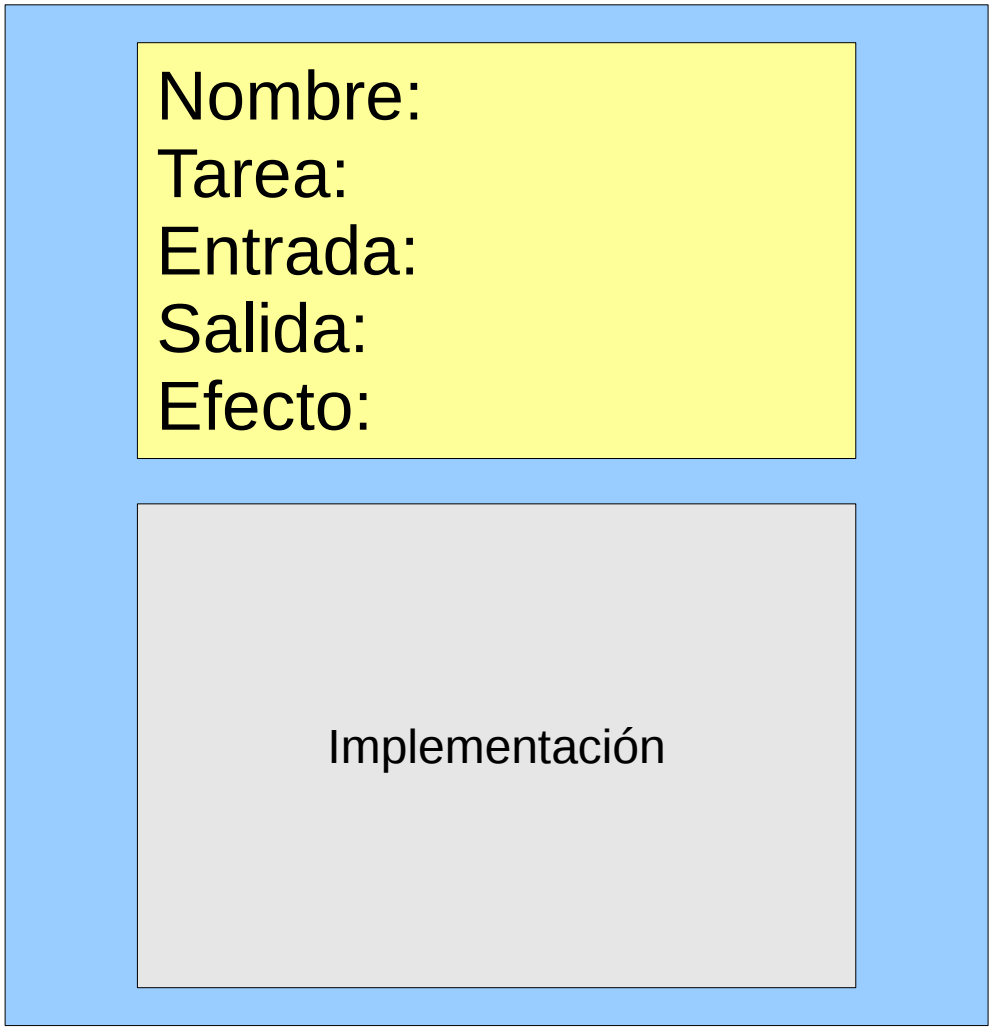
Problema complejo

Dividir en partes

Resolver cada parte con un bloque de código independiente

Abstracción
Comprender la función de la abstracción en programación

Módulo



Abstracción
Comprender la función de la abstracción en programación

¿Qué es Abstracción? Ignorar detalles específicos buscando generalidades que ofrezcan una perspectiva distinta, más favorable a su resolución y generalización.

Buscamos una descomposición útil del problema:

- Todas las partes deben estar al mismo nivel.
- Cada parte debe poder ser abordada por separado
- La solución de cada parte debe poder unirse al resto para obtener la solución final.

Abstracción
Comprender los diferentes tipos de abstracción

Abstracción por especificación: Permite ignorar la implementación de un método asociándole una descripción precisa de su comportamiento, en base a:

- **Precondición** (condiciones necesarias y suficientes para que el procedimiento se comporte correctamente)
- **Postcondición** (enunciados que se suponen ciertos tras la ejecución del procedimiento, si se cumplió la precondición).
- **Valor devuelto** (valor que devolverá el método).

```
//Requisitos: a >= 0.
//Devuelve: una aproximación de la raíz cuadrada de a.
double sqrt(double a);
```

Abstracción
Comprender los diferentes tipos de abstracción

Abstracción por parametrización: Se introducen parámetros para abstraer un número infinito de computaciones.

Ejemplo, las funciones:

- rellenar_vector_ceros(int v[],int n).

- rellenar_vector_cincos(int v[],int n).

Pueden ser reemplazadas por:

- rellenar_vector(int v[],int n, int val).

Abstracción
Comprender los diferentes tipos de abstracción

Abstracción procedimental: Conjunto de operaciones que se comportan como una operación. ~= método/función.

```
int sumatoria(int v[],int n) {
  int i;
  int suma;
  for (i = 0;i < n;i++) {
    suma = suma + v[i];
  }
  return suma;
}
```

Abstracción
Comprender los diferentes tipos de abstracción

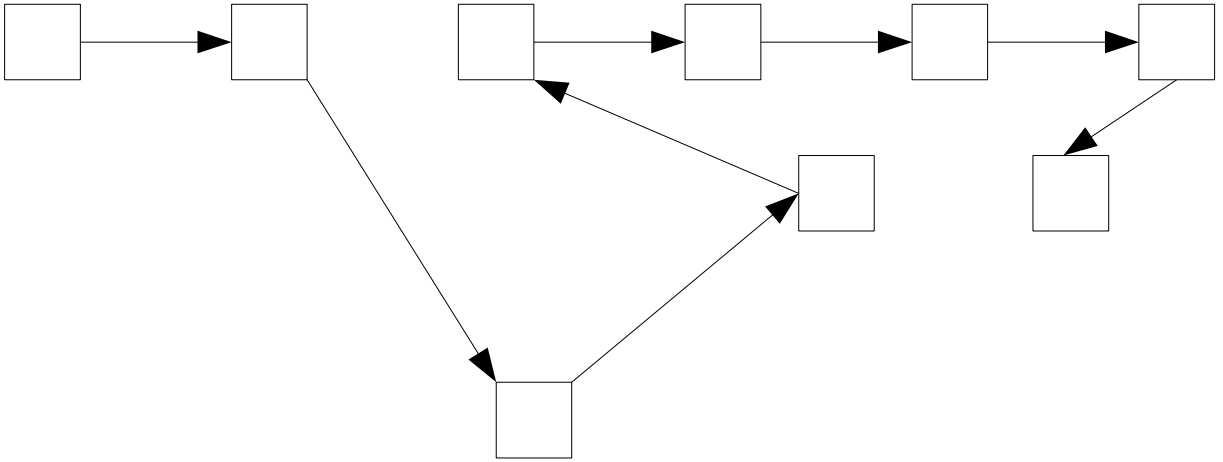
Abstracción de datos: Conjunto de datos y conjunto de operaciones que caracterizan el comportamiento de los datos. Las operaciones están vinculadas a los datos del tipo. ~= clase o estructura con métodos.

```
class Linea {
  Punto inicio;
  Punto fin;
  Void cambiarPuntoInicio(Punto nuevo);
  Void cambiarPuntoFin(Punto nuevo);
  Punto ObtenerPuntoInicio();
  Punto ObtenerPuntoFin();
}
```

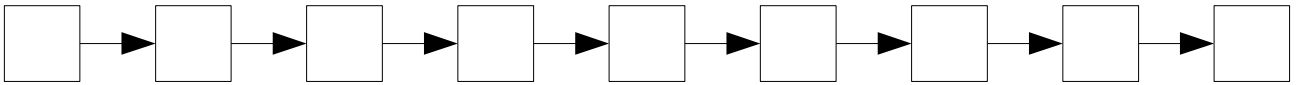
Abstracción
Comprender los diferentes tipos de abstracción

Abstracción de iteración: Permite trabajar sobre colecciones de objetos sin preocuparse por la forma en que se organizan. ~= iteradores.

Lista enlazada (puede estar repartida en memoria):



Vista desde iterador (a la hora de recorrerla):



Abstracción
Comprender la abstracción procedimental

Hay que tener en cuenta que para la especificación es **irrelevante la implementación**, pero **no qué hace**.

Propiedades:

- **Localidad**: Para implementar una abstracción procedimental no es necesario conocer la implementación de otras que se usen, sólo su especificación.
- **Modificabilidad**: Se puede cambiar la implementación de una abstracción procedimental sin afectar a otras abstracciones que la usen, siempre y cuando no cambie la especificación.

Abstracción
Comprender la abstracción procedimental

Propiedades de especificación de abstracción procedimental:

- Útil.
- Completa.
- Consistente.
- Indica el comportamiento en todos los casos aplicables.

Elementos de especificación de abstracción procedimental:

- Entradas.
- Salidas.
- Requisitos.
- Efectos.
- Elementos modificados.

Abstracción
Comprender la abstracción procedimental

Especificación de abstracción procedimental:

Cabecera (parte sintáctica): Indica el nombre del procedimiento, el número, orden y tipo de las entradas y salidas. Se suele adoptar la sintaxis de un lenguaje de programación completo.

Cuerpo (parte semántica): Argumentos, requisitos (precondiciones), valores de retorno y efecto (postcondiciones).

Abstracción
Comprender la abstracción procedimental

Especificación de abstracción procedimental:

```
/**
  * @brief Rellena un vector de cualquier longitud con ceros.
  * @param v Vector a rellenar.
  * @param n Longitud del vector.
  * @param val Valor con el que se desea rellenar el vector.
  * @pre El vector está inicializado.
  * @pre El vector tiene longitud mayor o igual a n.
  * @post Las n primeras posiciones del vector valen ahora val.
  *
  * Rellena el vector v con val, en las posiciones 0 a n-1 inclusive.
  */
void rellena_vector(int v[],int n,int val);
```

Abstracción
Comprender la abstracción procedimental

Argumentos:

- Explican el significado de cada parámetro de la abstracción.
- Expresan las restricciones sobre el conjunto de datos del tipo sobre los que se puede operar (ej: > 0).
- Indica si cada argumento se va a modificar o no, con el texto “Es MODIFICADO”.

Precondiciones/requisitos:

- Restricciones impuestas por el uso del procedimiento no recogidas por los argumentos. (ej: que previamente se haya ejecutado otro procedimiento).

Abstracción
Comprender la abstracción procedimental

Valores de retorno:

- Descripción de los valores que devuelve la abstracción y sus significados.

Postcondiciones/efectos:

- Describe el comportamiento del procedimiento para las entradas no excluidas por los requisitos.
- Debe indicar las salidas producidas y las modificaciones producidas sobre los argumentos marcados como que se van a modificar.
- Se suele indicar en pasado (ej: “el vector ha quedado vacío”).

Abstracción
Comprender la abstracción procedimental

Importante:

- No se debe indicar nada sobre el comportamiento del procedimiento cuando la entrada no satisfaga los requisitos.
- No se debe indicar cómo se llevan a cabo los efectos del procedimiento, salvo que sea requerido por la especificación (es decir, salvo que exista un requisito expreso sobre la forma de hacerlo o sobre la eficiencia requerida).

Abstracción
Comprender la abstracción procedimental

La especificación **se debe dar** junto al código fuente:

- En nuestro entorno no hay herramientas para mantener y soportar el uso de especificaciones, es responsabilidad exclusiva del programador.
- Necesidad de vincular código y especificación.
- Incluir la especificación entre comentarios en la parte de interfaz del código.
- Herramienta doxygen.

Abstracción
Comprender la abstracción procedimental

Para usar doxygen, la especificación se encierra entre /** */.

Ejemplo:

```
/**
  * @brief Frase breve que describe la función.
  * @param v Explicación del argumento v.
  * @param n Explicación del argumento h.
  * @pre Precondición 1.
  * @pre Precondición 2.
  * @post Postcondición 1.
  * @post Postcondición 2.
  * @return Valor devuelto.
  *
  * Descripción detallada del efecto.
  *
  */
int nombre(argumento 1,argumento 2);
```

Abstracción
Comprender la abstracción procedimental

Para usar doxygen, la especificación se encierra entre /** */.

Ejemplo:

```
/**
  * @brief Sumatoria de elementos de vector.
  * @param v Vector de elementos.
  * @param n Longitud del vector v.
  * @pre v es un vector inicializado.
  * @pre v contiene al menos n elementos.


  * @return Sumatoria de los elementos del vector.
  *
  * Calcula la sumatoria de los elementos de v y la
  * devuelve.
  */
int sumatoria(int v[],int n);
```

Abstracción
Comprender la abstracción procedimental

Factores importantes en abstracciones procedimentales:

- **Minimalidad**: Los requisitos deben ser los mínimos posibles.
- **Generalidad**: Que se pueda aplicar en el mayor número de casos posibles.
- **Simplicidad**: Que realice una única acción concreta.

Se conoce como **abstracción total** a aquella que sucede sin requisitos, cuando vale cualquier entrada.

Así mismo se conoce como **abstracción parcial** a aquella que sucede con requisitos, es decir, cuando sólo valen algunas entradas.

Abstracción
Comprender la abstracción de datos (TDA)

Un **TDA** o **Tipo de Dato Abstracto** es una entidad abstracta formada por un conjunto de datos y una colección de operaciones asociadas. Ejemplo: Tipos de datos, estructuras o clases en C++.

Especificación: Descripción de comportamiento.

Representación: Forma en la que se representan los datos.

Implementación: Forma en la que se hacen las operaciones.

Ventajas de la separación: se puede cambiar la visión interna sin tener que cambiar la externa; facilita la labor del programador al permitirlo concentrarse en cada fase por separado.

Vista Externa: Especificación. Vista Interna: Representación + Implementación.

Abstracción
Comprender la abstracción de datos (TDA)

Encapsulamiento: Mecanismo para impedir el acceso a la representación e implementación desde fuera del tipo.

Garantiza:

- Representación inaccesible (no se pueden modificar datos incoherentemente).
- Cambio de representación e implementación sin afectar a la especificación del TDA.

Constructores/Destructores: Mecanismo que nos abstrae de tener que controlar desde fuera de un TDA la creación o destrucción del mismo.

Abstracción
Comprender la abstracción de datos (TDA)

La **especificación** del TDA define su comportamiento, pero no dice nada sobre la implementación.

Indica el tipo de entidades que modela, qué operaciones se les pueden aplicar, cómo se usan y qué hacen.

Cabecera: Nombre del tipo y listado de operaciones.

Definición: Descripción del comportamiento sin indicar la representación. Se debe indicar si el tipo es mutable o no y dónde residen los objetos.

Operaciones: Especificación de las operaciones, una por una como abstracciones procedimentales (constructores primitivos (creación a partir de objetos de otras clases), constructores de copia, modificadores / mutadores, observadores / consultores y mixtos entre estos dos últimos.

Abstracción
Comprender el concepto de función de abstracción

La **función de abstracción** indica qué tipo de información del mundo real estamos representando mediante qué datos.

Esto es, cuando nosotros almacenamos unos datos (enteros, reales, vectores), qué significado tienen en el mundo real.

Debe proporcionarse en la especificación de cada TDA.

Ejemplo:

TDA Fecha: fA: tipo_rep → Q

{int dia, int mes, int anio} → día/mes/año

TDA Polinomio: fA: tipo_rep → Q

{int r[0..n]} → r[0] + r[1]x + … +r[n]x^n

Abstracción
Comprender el concepto de invariante de representación

El **invariante de representación** es una condición que siempre se va a cumplir en nuestros datos, porque están representando valores reales que también contienen ese invariante.

Debe proporcionarse en la especificación de cada TDA.

Ejemplo:

TDA Fecha:

Si mes == 4,6,9,11 → dia entre 1 y 31.

Si mes == 1,3,5,7,8,10,12 → dia entre 1 y 30.

Si mes == 2 y bisiesto(año) → dia entre 1 y 29.

Si mes == 2 y !bisiesto(año) → dia entre 1 y 28.

Abstracción
¿Qué hemos aprendido?

- **Función de la abstracción en la programación.**
- **Diferentes tipos de abstracción.**
- **Concepto de abstracción procedimental.**
- **Concepto de abstracción de datos.**
- **Concepto de Tipo de Datos Abstracto.**
- **Concepto de función de abstracción.**
- **Concepto de invariante de representación.**

Abstracción
Ejemplos

```
/**
  TDA Fecha
  Fecha::Fecha, dia, mes anio, ++, --, <, <=
         ==, <<, >>, dia_semana, fase_lunar

  Definición:
  Fecha representa fechas según el calendario
    Occidental.
  Son objetos mutables.
  Residen en memoria estática.
*/
```

Abstracción
Ejemplos

```
/**
  @brief Constructor primitivo.
  @post Objeto inicializado.

  Crea un objeto fecha con valor 1/1/2000
*/
Fecha::Fecha();
```

Abstracción
Ejemplos

```
/**
  @brief Constructor primitivo.
  @param d: dia de la fecha. 0 < dia <= 31
  @param m: mes de la fecha. 0 < mes <= 12
  @param a: año de la fecha.
  @pre Los tres argumentos deben representar
       Una fecha válida según el calendario
       Occidental.
  @post Objeto inicializado.

  Crea un objeto fecha con valor d/m/a.
*/
Fecha::Fecha(int d,int m,int a);
```

Abstracción
Ejemplos

```
/**
  @brief Obtiene el día.
  @return Día.

  Obtiene el día.
*/
int Fecha::dia() const;
```

Abstracción
Ejemplos

```
/**
  @brief Obtiene el mes.
  @return Mes.

  Obtiene el mes.
*/
int Fecha::mes() const;
```

Abstracción
Ejemplos

```
/**
  @brief Obtiene el año.
  @return Año.

  Obtiene el año.
*/
int Fecha::anio() const;
```

Abstracción
Ejemplos

```
/**
  @brief Devuelve fecha y la aumenta.
  @return Fecha actual.

  Modifica la fecha por la siguiente.
*/
Fecha Fecha::operator++();
```

Abstracción
Ejemplos

```
/**
  @brief Devuelve fecha y la decrementa.
  @return Fecha actual.

  Modifica la fecha por la anterior.
*/
Fecha Fecha::operator--();
```

Abstracción
Ejemplos

```
/**
  @brief Comparar si la fecha es menor.
  @param f Fecha con la que se compara.
  @return true si esta fecha es estrictamente
          menor a f, false en otro caso.

  Comparar si la fecha es menor que otra.
*/
bool Fecha::operator<(const Fecha & f) const;
```

Abstracción
Ejemplos

```
/**
  @brief Comparar si la fecha es menor o =.
  @param f Fecha con la que se compara.
  @return true si esta fecha es menor o igual
          a f, false en otro caso.

  Comparar si la fecha es menor o igual que
  otra.
*/
bool Fecha::operator<=(const Fecha & f)
const;
```

Abstracción
Ejemplos

```
/**
  @brief Asignación de fechas.
  @param f Fecha que se asigna.
  @return El valor de f.

  Asigna al receptor el valor de f y lo
  devuelve.
*/
Fecha Fecha::operator=(const Fecha & f);
```

Abstracción
Ejemplos

```
/**
  @brief Operador de conversión a texto.
  @param s Flujo al que se envía el texto.
         Es MODIFICADO.
  @param f Fecha que se escribe.
  @return El flujo s.

  Escribe en el flujo s el valor de f
  Convertido a texto y devuelve s.
*/
ostream & operator<<(ostream & s,
                     const Fecha & f);
```

Abstracción
Ejemplos

```
/**
  @brief Operador de lectura de fechas.
  @param s Flujo del que se lee.
         Es MODIFICADO.
  @param f Fecha hacia donde se lee.
         Es MODIFICADO.
  @return El flujo s.

  Lee una fecha del flujo s, en formato
  “dd/mm/aaaa” y pone el valor en f. Devuelve
  s.
*/
istream & operator>>(istream & s,
                     Fecha & f);
```

Abstracción
Ejemplos

```
/**
  @brief Operador de conversión a texto.
  @param s Flujo al que se envía el texto.
         Es MODIFICADO.
  @param f Fecha que se escribe.
  @return El flujo s.

  Escribe en el flujo s el valor de f
  Convertido a texto y devuelve s.
*/
ostream & operator<<(ostream & s,
                     const Fecha & f);
```