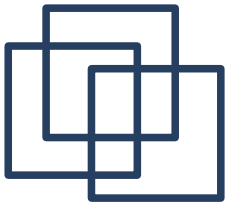
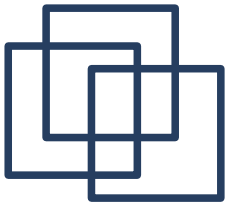


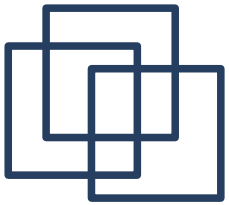
Tema 2

Abstracción de Datos



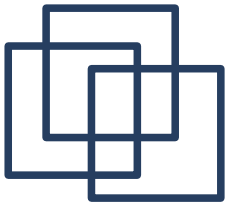
Objetivos Generales

- Abstracción en programación
- Abstracción procedimental
- Abstracción de datos (TDA)



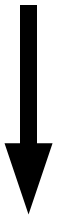
Abstracción en Programación: Objetivos

- Concepto de abstracción
- Papel de la abstracción en programación
- Tipos de abstracción
- Mecanismos de abstracción



Introducción

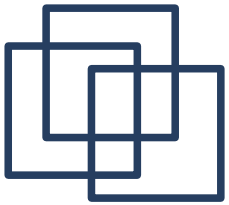
Ordenadores más avanzados



Programas más complejos

Facilita el diseño, codificación y prueba

Programación estructurada



Programación modular

Problema complejo

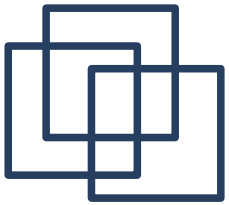
↓

Dividir en partes

↓

Resueltas con bloques
de código independientes

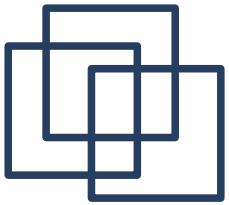
Módulo



Abstracción

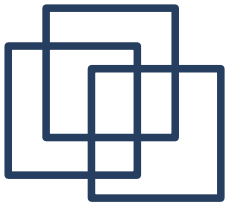
(Diccionario RAE)

abstraer.
(Del lat. *abstrahĕre*).
1. tr. Separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.



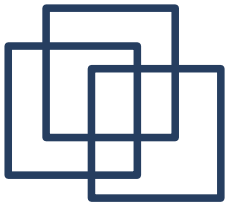
Abstracción en Programación

- *Abstracción*: Operación intelectual que ignora selectivamente partes de algo complejo para facilitar su comprensión
- Abstracción en la resolución de problemas: Ignorar detalles específicos buscando generalidades que ofrezcan una perspectiva distinta, más favorable a su resolución



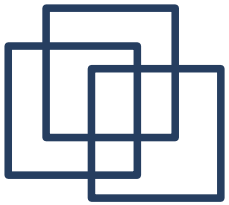
Abstracción: Descomposición útil

- *Abstracción*: descomposición en que se varía el nivel de detalle.
- Todas las partes deben estar el 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 en programación

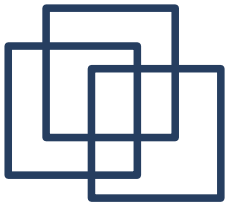
- Resolución de problemas mediante programación empleando la abstracción (a distintos niveles).
- Se aplica mediante distintos *mecanismos*.
- Productos a distinto nivel --> *Tipos* de abstracciones.



Mecanismos de abstracción en Programación

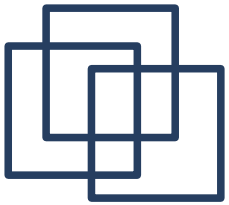
- **Abstracción por parametrización**. Se introducen parámetros para abstraer un número infinito de computaciones. Ej.: cálculo de cos(x)
- **Abstracción por especificación**. Permite ignorar la implementación concreta de un procedimiento asociándole una descripción precisa de su comportamiento.

```
double sqrt(doble a);
Requisitos:  a >= 0;
Efecto:  devuelve una aprox. de √a
```



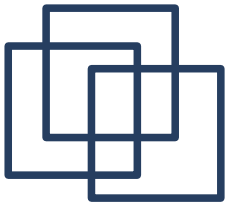
Abstracción por Especificación

- Expresada como:
  - *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*



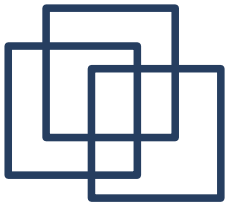
Ejemplo

```
/**
  busca_minimo: localiza el elemento mínimo de un vector
  Precondición:
        num_elem > 0
        v vector con num_elem elementos
  Postcondición:
        Devuelve la posición del mínimo elemento de v
*/

int busca_minimo(float v[], int num_elem)
```



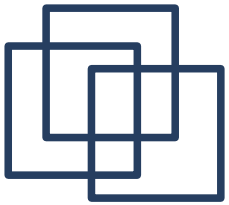
Tipos de Abstracción

- **Abstracción Procedimental**. Conjunto de operaciones que se comportan como una operación
- **Abstracción de Datos**. Conjunto de datos y conjunto de operaciones que caracterizan el comportamiento de los primeros. Las operaciones están vinculadas a los datos del tipo
- **Abstracción de Iteración**. Permite trabajar sobre colecciones de objetos sin preocuparse por la forma en que se organizan.



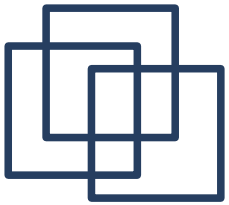
Abstracción Procedimental: Objetivos

- Concepto
- Especificación de una A.P.
- Especificación usando *doxygen*



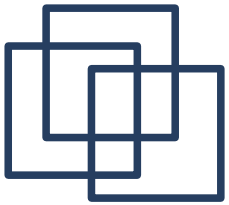
Abstracción Procedimental

- Permite considerar (y utilizar) un conjunto de operaciones de cálculo como una operación simple.
- *Formalmente*: realiza la aplicación de un conjunto de entradas en las salidas con posible modificación de las entradas.



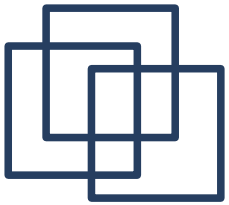
Abstracción Procedimental

- La identidad de los datos no es relevante para el diseño. Sólo interesa el número de parámetros y su tipo
- Abstracción por especificación: es irrelevante la implementación, pero NO qué hace



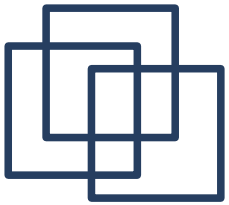
Características de la Especificación de la A.P.

- *Localidad*: Para implementar una abstracción procedimental no es necesario conocer la implementación de otras que se use, 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.



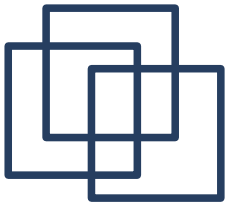
Propiedades de la especificación de la A.P.

- Útil
- Completa
- Consistente
- Indicar el comportamiento en todos los casos en que sea aplicable



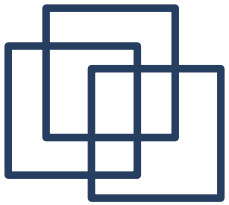
Elementos de la especificación de una A.P.

- Entradas
- Salidas
- Requisitos
- Efectos
- Elementos modificados



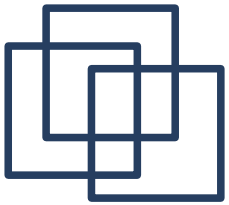
Especificación de una A.P.

- **Cabecera**: (Parte sintática) Indica el nombre el procedimiento y el número, orden y tipo de las entradas y salidas. Se suele adoptar la sintaxis de un lenguaje de programación concreto.
- **Cuerpo**: (Parte semántica) Compuesto por:
    - Argumentos
    - Requiere
    - Valores de retorno
    - Efecto



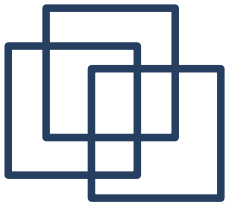
E.A.P.: Argumentos

- Explica el significado de cada parámetro de la abstracción; indica qué representa.
- Expresa las restricciones sobre le cj de datos del tipo sobre los que puede operar. Procedimiento *total* y *parcial*.
- Indica si cada argumento es modificado o no. Cuando un parámetro vaya a ser modificado se incluirá la expresión “Es MODIFICADO”.



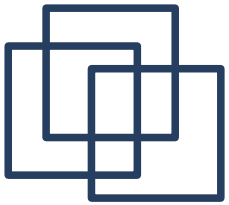
E.A.P.: Requiere

- Restricciones impuestas por el uso del procedimiento no recogidas por *Argumentos*. Ej.: que previamente se haya ejecutado otra abstracción procedimental.



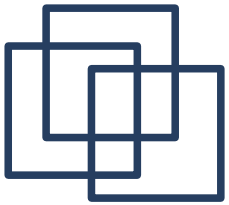
E.A.P.: Valores de retorno

- Descripción de qué valores devuelve la abstracción y qué significan.



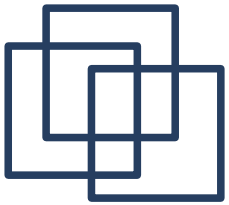
E.A.P.: Efecto

- Describe el comportamiento del procedimiento para las entradas no excluidas por los requisitos. El efecto debe indicar las salidas producidas y las modificaciones producidas sobre los argumentos marcados con “Es MODIFICADO”.



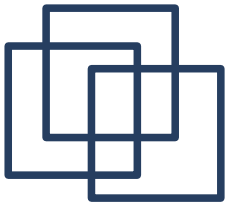
E.A.P.: Efecto (II)

- Importante:
  - Nada sobre el comportamiento del procedimiento cuando la entrada NO satisface los requisitos,
  - nada sobre cómo se lleva a cabo dicho efecto, salvo que esto sea requerido por la especificación. (Ej.: requisito expreso sobre la forma de hacerlo o sobre la eficiencia requerida)



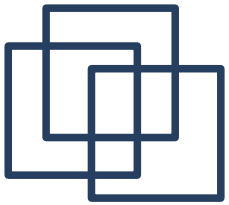
Especificación junto al código fuente

- Falta de herramientas para mantener y soportar el uso de especificaciones → Responsabilidad exclusiva del programador
- Necesidad de vincular código y especificación
- Incluirla entre comentarios en la parte de interfaz del código
- Herramienta doxygen.



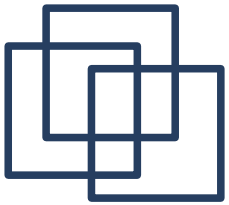
Especificación usando doxygen

- Toda la especificación se encierra entre
  / … */**
- Se pone una frase que describa toda la función precedida de **@brief**
- Cada argumento se precede de **@param**
- Los requisitos adicionales se indican tras **@pre**
- Los valores de retorno se ponen tras **@return**
- La descripción del efecto al final



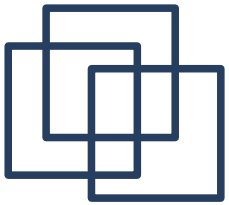
Ejemplo de uso de doxygen

```
/**
@brief Comprueba si una cadena de caracteres es un palíndromo
@param s cadena que se va a  comprobar
@return true si s es palindromo y false en caso contrario
*/
bool palindromo(const string & s)

/**
@brief Determina si un entero aparece en alguna posición de un vector
@param x elemento que se va a buscar
@param v vector con n elementos
@param n numero de elementos del vector. n>0
@return true si x aparece en alguna posición de v y false en caso contrario
*/
bool buscar(int x, int v[], int n)
```



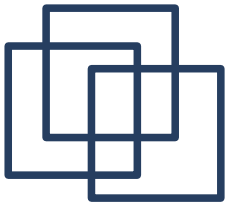
Diseño de abstracciones procedimentales

1. **Minimalidad**: Los requisitos mínimos posibles
2. **Generalidad**: Poder aplicarla en el mayor número de casos posibles
3. **Simplicidad**: Realizar una única acción concreta.
4. Carácter total o parcial.



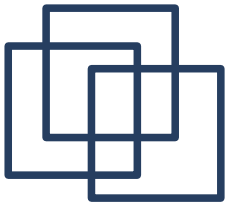
Tipo de Dato Abstracto (TDA)

- Entidad abstracta formada por un conjunto de datos y una colección de operaciones asociadas.
- Ejemplo: tipos de datos en C++.



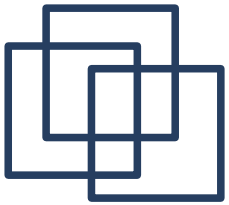
Conceptos

- *Especificación*: Descripción del comportamiento del TDA
- *Representación*: Forma concreta en que se representan los datos en un lenguaje de programación
- *Implementación*: Forma específica en que se expresan las operaciones.



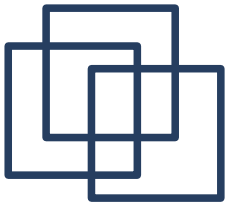
Visiones de un TDA

- Visión externa: especificación
- Visión interna: representación e implementación
- Ventajas de la separación:
  - Se puede cambiar la visión interna sin afectar a la externa
  - Facilita la labor del programador permitiéndole concentrarse en cada fase por separado



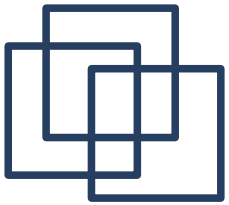
TDA y Ocultación de información

- **Ocultación de información**: Mecanismo para impedir el acceso a la representación e implementación desde fuera del tipo. Garantiza:
  - Representación inaccesible: razonamientos correctos sobre la implementación
  - Cambio de representación e implementación sin afectar a la especificación del TDA
- **Encapsulamiento**: Ocultación de información y capacidad para expresar el estrecho vínculo entre datos y operaciones



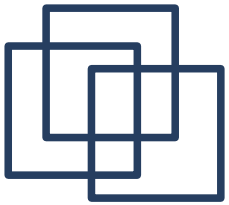
Especificación del TDA

- *Define* el comportamiento, pero no dice nada sobre su implementación
- Indica el tipo de entidades que modela, qué operaciones se les pueden aplicar, cómo se usan y qué hacen



Estructura de la especificación

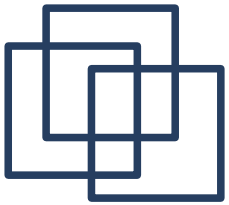
- *Cabecera*: nombre del tipo y listado de las operaciones
- *Definición*: Descripción del comportamiento sin indicar la representación. Se debe indicar si el tipo es mutable o no y dónde residen los objetos
- *Operaciones*: Especificación de las operaciones, una por una como abstracciones procedimentales



Tipos de operaciones

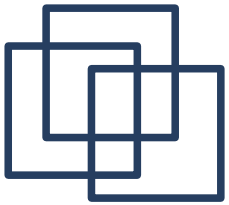
1) Constructores primitivos. Crean objetos del tipo a partir de objetos de otras clases
2) Constructores de copia. Crean objetos del tipo a partir de otros objetos del tipo
3) Modificadores o mutadores.
4) Observadores o consultores.

Es habitual la combinación de tipos. En particular de los 3 y 4.



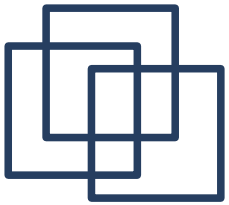
TDA Fecha

```
/**
  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.
*/
```



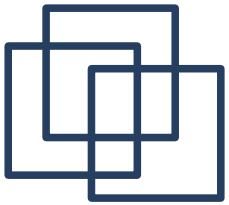
TDA Fecha

```
/**
   @brief Constructor primitivo.
   Crea un objeto fecha con valor 1/1/2000
*/
Fecha();
/**
   @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.
      Los tres argumentos deben representar una fecha válida
      según el calendario juliano.
   Crea un objeto con valor d/m/a.
*/
Fecha(int d, int m, int a);
```



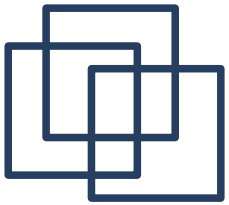
TDA Fecha

```
/**
  @brief Obtiene el día del receptor.

  @return Día del receptor.
*/
int dia() const;

/**
  @brief Obtiene el mes del receptor.

  @return Mes del receptor.
*/
int mes() const;

/**
  @brief Obtiene el año del receptor.

  @return Año del receptor.
*/
int anio() const;
```



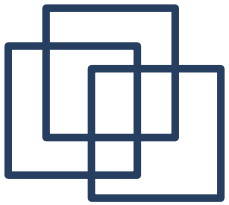
TDA Fecha

```
/**
  @brief Fecha siguiente.

  @return Fecha siguiente a la del receptor

  Modifica el receptor por la fecha siguiente al valor que tiene.
*/
Fecha operator++();

/**
  @brief Fecha anterior.

  @return Fecha anterior a la del receptor

  Modifica el receptor por la fecha anterior valor que tiene.
*/
Fecha operator--();
```



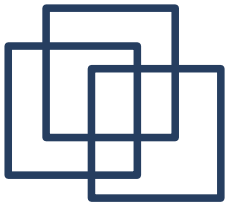
TDA Fecha

```
/**
  @brief Comparación menor para fechas
  @param f: Fecha con que se compara el receptor
  @return true: si el receptor es estrictamente anterior a f.
          false: en otro caso.
*/
bool operator<(const Fecha & f) const;

/**
  @brief Comparación menor o igual para fechas
  @param f: Fecha con que se compara el receptor
  @return true: si el receptor es estrictamente anterior o
                igual a f.
          false: en otro caso.
*/
bool operator<=(const Fecha & f) const;
```



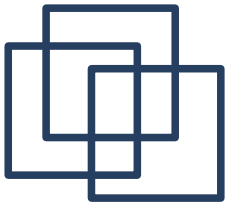
TDA Fecha

```
/**
  @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 operator=(const Fecha & f);
```



TDA Fecha

```
/**
  @brief Operador de conversión a texto (escritura).

  @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 convetido a texto y
  devuelve s.
*/
ostream & operator<<(ostream & s, const Fecha & f);
```



TDA Fecha

```
/**
  @brief Operador de lectura de fechas
  @param s: flujo del que se lee. Es MODIFICADO.
  @param f: Fecha que se lee. Es MODIFICADO.

  @return Flujo s.

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