# PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

# OBJECT ORIENTED PROGRAMMING (OOP)

curso 21/22

# PROGRAMACIÓN CON TIPOS ABSTRACTOS DE DATOS

# Ejemplo: lanzar unos dados

```
int d1, d2, sum, r1, r2;

srand(seed); // pseudo-random integer seed
r1 = rand();    // in the range 0-RAND_MAX
r2 = rand();
d1 = (r1 % 6) + 1;
d2 = (r2 % 6) + 1;
sum = d1 + d2;
printf("d1 = %d\n", d1);
printf("d2 = %d\n", d2);
printf("sum = %d\n", sum);
```

¿Es este código entendible fácilmente, modificable, ... mantenible?

Necesitamos mayor nivel de abstracción para enfrentarnos a problemas grandes

PROGRAMACIÓN CON
TIPOS ABSTRACTOS DE DATOS (TAD)

(es el concepto más importante de la POO)

# LA ABSTRACCIÓN

- Es un concepto clave de la POO (y de la Informática)
- Definición: *Abstraer es formar mediante una operación intelectual una idea mental o noción de un objeto extrayendo de los objetos reales particulares los rasgos esenciales, comunes a todos ellos.*
- Desarrollo de Software: manejar la complejidad de un problema ocultando detalles innecesarios en cada etapa del desarrollo del software mediante el uso de TAD.

## DE LA ABSTRACCIÓN AL... DISEÑO

1. Analizar un problema.
2. Identificar cada componente a nivel abstracto y sus funciones.
3. Diseñar una solución con dichos componentes.

La implementación es una etapa posterior.

# El Diseño

- Proceso:
  - Elaborar un plan de lo que se quiere hacer
    (en un paso posterior: detalle de cada paso)
- Resultado:
  - Producto de ingeniería de calidad
  - Producto de ingeniería informática: software de calidad

PROGRAMACIÓN CON
TIPOS ABSTRACTOS DE DATOS:

- Hay una etapa previa de DISEÑO

# TAD / CLASE en POO

**TAD** Dados

Representa el lanzamiento de dos dados.

**OPERACIONES**

- lanzamiento: simula el lanzamiento de 2 dados.
- getDado1: devuelve el valor del primer dado.
- getDado2: devuelve el valor del segundo dado.
- getSuma: devuelve la suma de los dos dados.

# Class Dados

*dados.h*

```
class Dados{
  private:
    int d1_, d2_;
  public:
    void lanzamiento();
    int getDado1();
    int getDado2();
    int getSuma();
    . . .
  };
```

*juego.cc*

```
Dados d;
d.lanzamiento();
cout << “d1 = “ << d.getDado1();
cout << “d2 = “ << d.getDado2();
cout << “sum = “ << d.getSuma();
```

# Ejemplo: lanzar unos dados

*juego.c*

```
int d1, d2, sum, r1, r2;
srand(seed);
r1 = rand();
r2 = rand();
d1 = (r1 % 6) + 1;
d2 = (r1 % 6) + 1;
sum = d1 + d2;
printf("d1 = %d\n", d1);
printf("d2 = %d\n", d2);
printf("sum = %d\n", total);
```

*juego.cc*

```
Dados d;
d.lanzamiento();
cout << “d1 = “ << d.getDado1();
cout << “d2 = “ << d.getDado2();
cout << “sum = “ << d.getSuma();
```

# dados.cc

```
Dados::Dados()
{
 srand(time(NULL));
}
void Dados::lanzamiento()
{
 d1_=(rand()%6)+1;
 d2_=(rand()%6)+1;
}
```

```
int Dados::getDado1()
{
 return d1_;
}
int Dados::getDado2()
{
 return d2_;
}
int Dados::getSuma()
{
 return d1_+d2_;
}
```

# Programación ED vs TAD

ED: Bajo Nivel de Abstracción

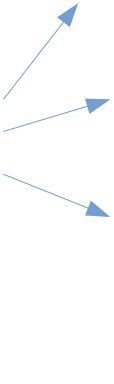
vs

TAD: Alto Nivel de Abstracción

# Clase y objeto

```
class Book{
private:
  string title_;
  string author_;
  . . .

public:
  string getTitle();
  string getAuthor();
  bool setTitle(string new_title)
  . . .

};
```

Clase Book



a
"C++ Programming Language"
Bjarne Stroustrup

b
"C++ A Beginner's Guide"
Herbert Schildt

c
"Construcción de Software OO"
Bertrand Meyer

a, b y c son objetos de la clase Book

(también se denominan instancias de la clase *Book*)

```
b.getAuthor()
```

Herbert Schildt

```
c.getAuthor()
```

Bertrand Meyer

Métodos (también se denominan funciones u operaciones) de la clase *Book*

# Cliente o usuario de una clase

```
#include “book.h”
int main(void)
{
  Book b;
  . . .
  b.getTitle();
  . . .
}
```

**cliente**/usuario

Este programa hace de **cliente** de la clase Book

El autor de este programa (el cliente):

- No tiene por qué ser el mismo que el autor de la clase Book.
- Desconoce los datos internos de la clase Book (tampoco los necesita...).
- Desconoce cómo funciona por dentro la clase Book (desconoce cómo está implementada internamente).
- Ni siquiera debe preocuparle como esté hecha por dentro la clase Book.
- Solo debe preocuparle el programa que está haciendo en ese momento.
- Se conecta a cada objeto a través de su interfaz o parte **“public”** de la clase.
- No puede acceder a la parte **“private”** de la clase.

# Encapsulamiento

- Una clase tendrá unos datos internos que solo podrán ser accedidos mediante las operaciones definidas para ello.
- Los datos internos irán en la **sección "private"** y no podrá accederse a ellos si no es con las operaciones/funciones correspondientes de la **sección "public"**.
- Las operaciones irán en la sección "public" y serán el único método para acceder al objeto.
- Se dice que los datos internos quedan ocultos, encapsulados
- Los datos internos de un objeto también se denomina "estado de un objeto".

# Encapsulamiento. Venjatas

- Impide acceso directo al estado interno de un objeto:
  - Impidiendo operaciones no permitidas.
  - Simplifica su comprensión ya que no será necesario conocer como está hecho internamente.
  - Si en el futuro se modifican sus datos internos, no afectará a ningún programa que use dicho objeto ya que nunca se accede a ellos.
- Solo podrá interactuarse con el objeto mediante su interfaz pública.

# Encapsulamiento. Ejemplos

```
class Dados{
private:
  int d1_;
  Int d2_;
  . . .

public:
  . . .

};
```

```
class Fecha{
private:
  int d_;
  int m_;
  int a_;
  . . .

public:
  . . .

};
```

Son operaciones no permitidas, estados erróneos del objeto: NO SE DEBEN PERMITIR
Los objetos son protegidos mediante el ENCAPSULAMIENTO

# Primera práctica de C++

- El objeto, una variable más con su memoria interna asignada.
- Nuevo tipo “bool” en C++ (true 1, false 0)
- Nuevo tipo string
- g++
- namespace
- Objetos `cin` y `cout`
- Constructor de una clase
- includes
- Extensiones de archivos .cc .h
- Name conventions...

# Naming conventions

- Files: lowercase and can include underscores (_) or dashes (-)
- Class and types names: start with a capital letter and have a capital letter for each new word (MyClass). This is named: mixed case
- Variables: lowercase with underscore between words (mid_age)
- Class members: with trailing underscore (d1_, d2_)
- Constants: leading “k” mixed case
- MACRO: all capitals with underscore
- `More: https://google.github.io/styleguide/cppguide.html#Naming`

# Diseño Orientado a Objetos

- ¿Quién y qué?
- Dejamos a un lado el ¿cómo?
- ¿Quién?: Las clases
- ¿Qué?: Los métodos de las clases
- También diseñamos: pruebas, patrones de diseño, etc.


Un modelo en cascada (WATERFALL)[Royce, 1970]

# Software de calidad

- Calidad funcional (externa) y estructural (interna) del software
- Factores de calidad:
  - Factores de calidad externos
  - Factores de calidad internos

# Factores de calidad del software

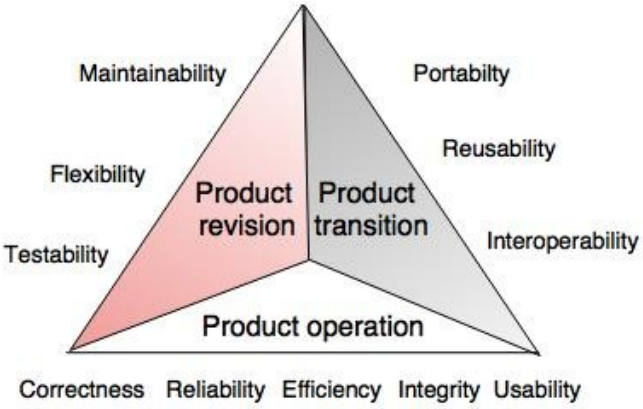



**Fig. - McCall's quality factors**

*[McCall y colaboradores, 1977]*

**Product operation:**
rel. funcionalidad

**Product revision:**
rel. mantenimiento

**Product transition:**
rel. adaptación a otros entornos

# Factores de calidad del software

| Quality Categories | Quality Factors | Broad Objectives |
|---|---|---|
| **Product operation** | Correctness | Does it do what the customer wants? |
| | Reliability | Does it do it accurately all of the time? |
| | Efficiency | Does it quickly solve the intended problem? |
| | Integrity | Is it secure? |
| | Usability | Can I run it? |
| **Product revision** | Maintainability | Can it be fixed? |
| | Testability | Can it be tested? |
| | Flexibility | Can it be changed? |
| **Product transition** | Portability | Can it be used on another machine? |
| | Reusability | Can parts of it be reused? |
| | Interoperability | Can it interface with another system? |

*(Estudiar las definiciones en los apuntes del profesor)*

# TEMA 2: DESCOMPOSICIÓN, ABSTRACCIÓN, ESPECIFICACIÓN

# Objetos del mundo real

- En mundo real hay objetos/entidades: personas, productos (libros, ropa, etc.)
- Es habitual pensar en términos de objetos
- Pensamos en ellos como abstracciones
  - Cada uno tiene su comportamiento
  - Cada uno tiene sus atributos
- Los entendemos mejor centrándonos exclusivamente en su comportamiento y sus atributos de interés.
- Otros aspectos: tienen parecidos, unos se relacionan con otros.

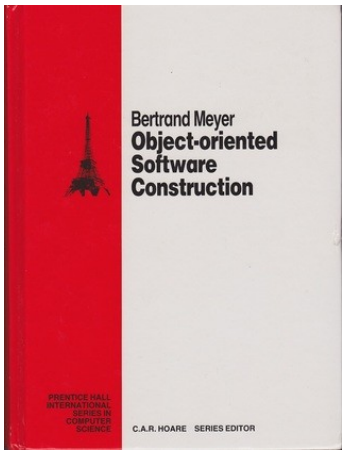
# Objetos de la POO

- Representan un objeto del mundo real en el programa
- Nos centrámos en atributos y comportamiento
- Se parecen y/o relacionan entre sí
- Cada clase y cada objeto es independiente
- La clase como **unidad de descomposición** de un problema real

# Descomposición

Bertrand Meyer (Francia, 1950). *Construcción de Software Orientado a Objetos*. 1999:

- Criterios/requisitos
- Reglas
- Principios



# Descomposición 1/3. Criterios

1) Descomposición modular
2) Composición modular
3) Comprensibilidad modular
4) Continuidad modular
5) Protección modular

# Descomposición 2/3. Reglas

1) Correspondencia directa
2) Pocas interfaces
3) Pequeñas interfaces
4) Interfaces explícitas
5) Ocultación de la información

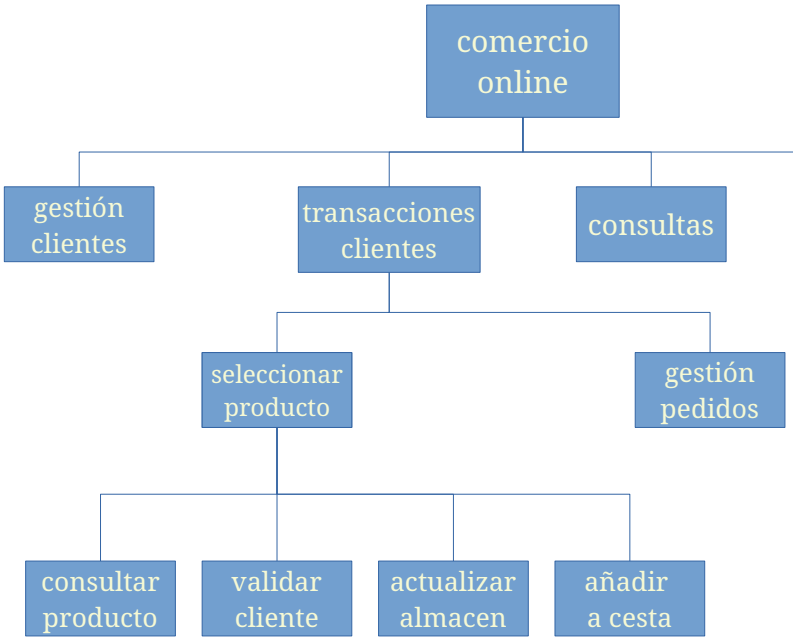
# Descomposición 3/3. Principios

1) Unidades modulares lingüísticas
2) Auto-documentación
3) Acceso uniforme
4) Principio abierto-cerrado
5) Elección única

# TDD vs OOD

- Descomposición funcional **vs** descomposición orientada a objetos
- Descomposición arriba-abajo (top-down design, TDD) **vs** descomposición orientada a objetos (object oriented design, OOD)
- Ejemplo: aplicación de comercio online

# TDD con Estructuras de Datos



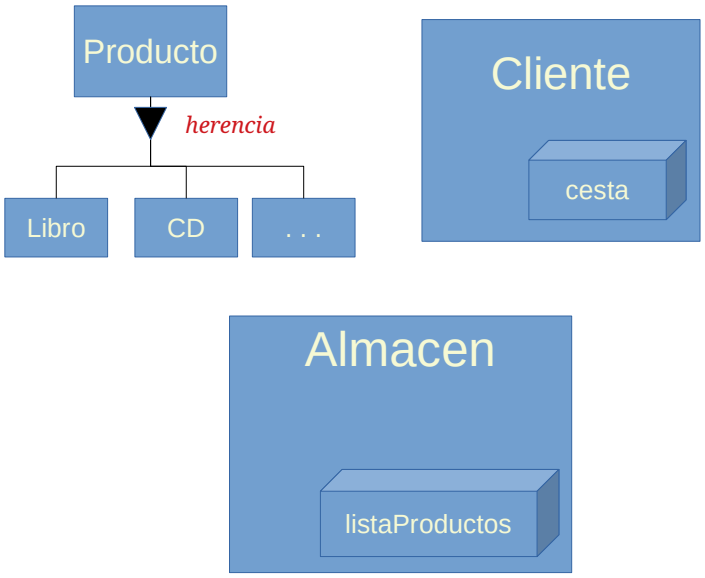

- Hierarchic
- Tightly coupled
- Shared data structures
- Loosely reusable
- Dependency between modules
- Difficulty working in a team
- Etc...

# OOD con Tipos Abstractos de Datos

Clases:

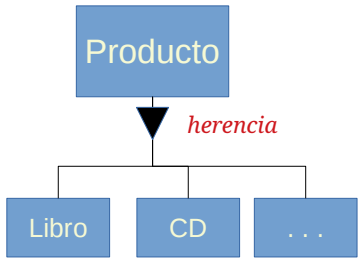
# OOD

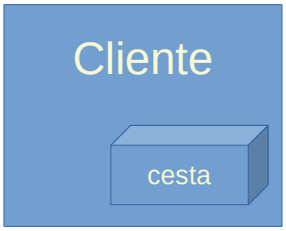
```
Algoritmo  Seleccionar Producto (Cliente cliente)

Producto producto

Almacen almacen

ListaProductos lp

if (lp = almacen.filtar("Memoria USB")) then

  lp.show() #muestra lista al usuario

  id = lp.selected() # usuario selecciona producto
                     # de la lista.

  producto=almacen.get(id)

  producto.show() # muestra detalles del producto
                  # al usuario.

if (producto.selected() AND cliente.ok()) then

  cliente.cesta.insert(producto)

  almacen.reserva(producto, 1) # reserva 1 unidad del
                               # producto.

Fin
```



# OOD

```
Algoritmo  Gestión Pedido (Cliente cliente)
Producto producto
Almacen almacen
ListaProductos lp
GestorPago gp
lp=cliente.cesta.getProductos()
for i in lp:
  cantidad = cliente.cesta.getCantidad(i)
  almacen.salida(i, cantidad, cliente.getId())
gp.iniciaPago(cliente)
gp.confirmaPago(cliente)
Fin
```





# Ejemplo: el fichero estadístico

123 -5609 -7889 33 -1
674 8998 456434 2 334 -
234 98 6754 -456721
1098 789 66 5437 98
199 -590643 8667 67
899 1009 234 34 134 45
67 5634 -25 467 4674 2
45 34567 -89 90 56 78
456 2 341 34 -2354 -567
567 867 890 54 83 562
546 34567 546 73 -67
45678 -56 783 567 -356
78…..

?

# Ejemplo: el fichero estadístico

```
Algoritmo estadistica(fichero f)
Tabla t
mientras no fin(f) hacer
   t.añadir(leer(f))
fin_mientras
para i de 1 a t.total() hacer
    x=t.infonum(i)
    frecx=t.infofrec(x)
    escribir("entero ",x, "frecuencia =", frecx)
  fin_para
Fin
```

# Especificación de TAD

**TAD** Tabla
**DESCRIPCIÓN**
Gestiona un conjunto de enteros y sus estadísticas

**OPERACIONES**

- **PROC** añade(entero i) **DEV** ()
  **REQUIERE:** True
  **MODIFICA: Ø**
  **EFECTOS:** incrementa la frecuencia del entero "*i*" en una unidad.
- **PROC** total() **DEV** (entero n)
  **REQUIERE:** True
  **MODIFICA : Ø**
  **EFECTOS:** devuelve en "n" el número de enteros distintos en la tabla.

# Especificación de TAD

**TAD** Tabla
**DESCRIPCIÓN**
Gestiona un conjunto de enteros y sus estadísticas

**OPERACIONES**

- **PROC** infonum(entero i) **DEV** (entero x)
  **REQUIERE:** True
  **MODIFICA: Ø**
  **EFECTOS:** devuelve el entero “x” que ocupa la posición i-ésima. Si la posición “i-ésima” no existe, lanza la excepción FUERA_DE_RANGO
- **PROC** infofrec(entero x) **DEV** (entero f)
  **REQUIERE:** True
  **MODIFICA : Ø**
  **EFECTOS:** devuelve en “f” el número de veces que se repite “x” en la tabla.

# Especificación de TAD

La especificación informa de manera precisa de los detalles relevantes del TAD:

- **RELEVANTE:** ¿qué? (elementos subrayados)
- **IRRELEVANTE:** ¿cómo?

**TAD** TAD's name
**DESCRIPTION** short description of the TAD

**OPERATIONS**

**PROC** name (parameters) **DEV** (return value)
**REQUIRES:** condition
**MODIFIES:** list of modified parameters
**EFFECTS:** short description of procedure effects

. . .

# Especificación de TAD

Más especificaciones de procedimientos: breves, concisas, precisas.

- **PROC** concat(string a, string b) **DEV** (string ab)

  **REQUIRES:** True

  **MODIFIES: Ø**

  **EFFECTS:** “ab” results in characters in “a” (same order than in “a”) followed by characters in “b” (same order than in ”b”)

- **PROC** deleteDuplicate(int a[]) **DEV** ()

  **REQUIRES:** True

  **MODIFIES:** a

  **EFFECTS:** deletes duplicate elements in “a” leaving the first one in his original position. Example: if a=[3, 13, 3, 6] before the call, the resultin a will be a=[3, 13, 6].

# Excepciones

Más especificaciones de procedimientos: breves, concisas, precisas.

- **PROC** binarySearch(int a[], int x) **DEV** (int i) **EXCEPTIONS** NotFoundException
  **REQUIRES:** “a” ordered ascending

  **MODIFIES: Ø**

  **EFFECTS:** if “x” is not in “a”, the procedure **throw** the exception “NotFoundException”.
  If “x” is in “a”, then “i” is such that a[i]=x.

- **PROC** factorial(int n) **DEV** (int i) **EXCEPTIONS** NotPositiveException
  **REQUIRES:** True

  **MODIFIES: Ø**

  **EFFECTS:** if “n” is not positive, the procedure **throw** the exception “NotPositiveException”.
  If “n” is positive, then “i” is the factorial of “n”.

# Excepciones

```
try {
    proc(i,j)
    }
catch(A)
    {
    . . .
    }
catch(B)
    {
    . . .
    }
catch(C)
    {
    . . .
    }
```

puede lanzar (*throw*) una excepción

código de manejo/captura (*catch*) de excepciones

# Implementación de TAD

- Requisitos de la implementación:
  - Pequeña (la menor/más-simple que cumpla la especificación)
  - cerrada/abierta
  - Etc. (criterios, reglas y principios de descomposición modular de *Bertrand Meyer*)
- STUBS
  - Función/procedimiento pendiente de implementar pero *callable*
  - “estoy aquí”
  - Posteriormente se completa.
  - Prototipado rápido

# Operaciones con TADS

- Constructores (A::A() en C++)
- Observadores: *getters...*
- Modificadores: *setters...*
- Destructores (~ en C++)