

# Comenzando Orientación a Objeto en C++

Profesor: Juan Bekios Calfa

<http://jbekios.ucn.cl>

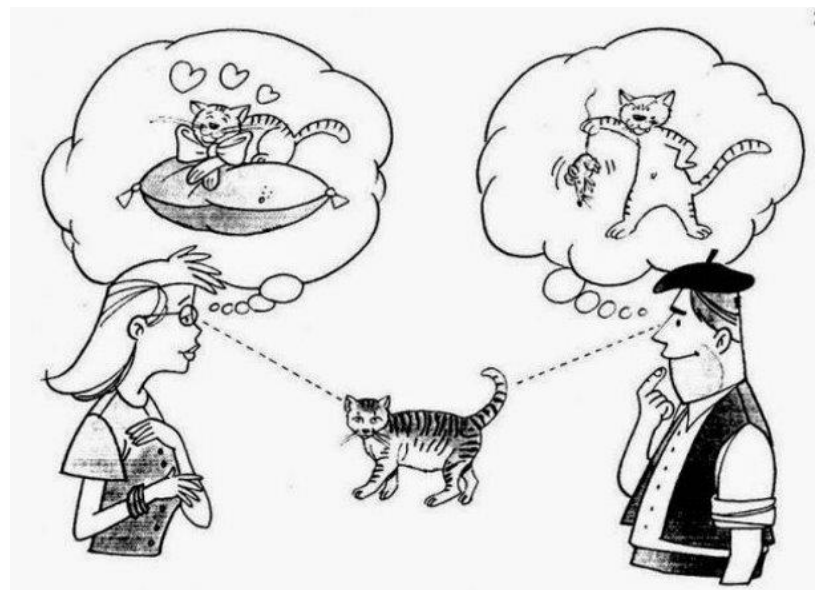
[juan.bekios@ucn.cl](mailto:juan.bekios@ucn.cl)

4/1/2022

1

## Abstracción

- La abstracción se centra en las características esenciales de un objeto, en relación a la perspectiva de un observador.
- Un **tipo de dato abstracto** (TDA) o **tipo abstracto de datos** (TAD) es un modelo matemático compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo.  
(fuente:wikipedia)



# Tipo de Datos Abstractos

- Un **Tipo de Abstracto de Datos (TDA)** es un mecanismo, de un lenguaje de programación, diseñado para imitar las propiedades abstractas de un ***nuevo tipo de datos incorporado***.
- Debe incluir una especificación de las operaciones que pueden aplicarse a los datos.
- Debe ocultar los detalles de la implementación del código para el cliente.
- Estas propiedades, generalmente, se llaman **encapsulación** y **ocultamiento de información** (con diferentes énfasis).

3

Dr. Juan Bekios Calfa

## Encapsulación

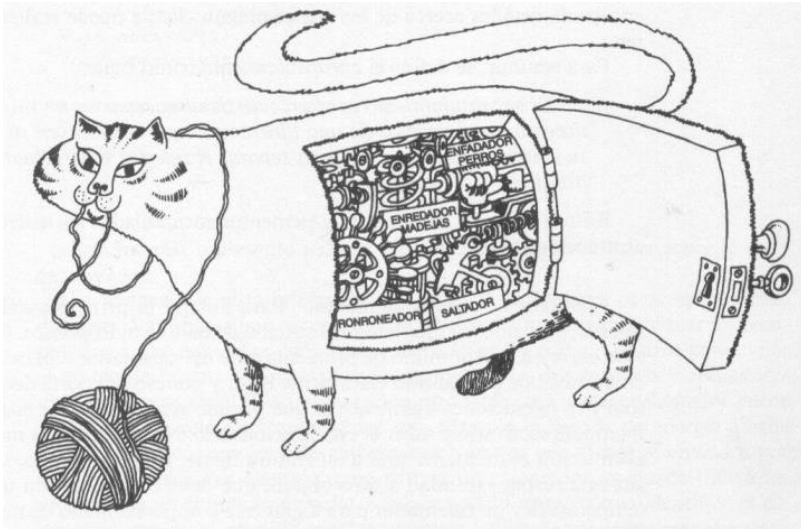


- Recolección de una sola localización de todas las definiciones relacionadas con **un tipo de datos**.
- Restringir el uso del tipo a las operaciones definidas para dicha localización.

4

Dr. Juan Bekios Calfa

# Ocultamiento de Información




- Oculta los detalles de la implementación de un objeto.
- Separación de los detalles de implementación de esas definiciones y la supresión de dichos detalles en el uso del tipo de datos.

5

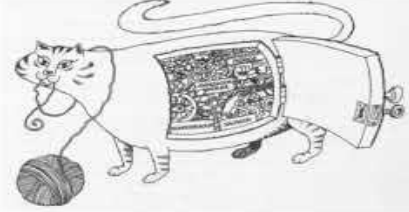
Dr. Juan Bekios Calfa

## TDA (Tipo Abstracto de Datos)



Universidad Católica del Norte  
Departamento de Ingeniería de Sistemas y Computación  
Antofagasta, Chile

### Ocultamiento de Información



- Oculta los detalles de la implementación de un objeto.
- Separación de los detalles de implementación de esas definiciones y la supresión de dichos detalles en el uso del tipo de datos.

Dr. Juan Bekios Calfa

Para más información: <https://youtu.be/CFE7S5g8XPA>

# TDA: Clase

```
1  class nombreClase
2  {
3      // Atributos: Definición
4      // Métodos: Definición e implementación
5  };
6
7  // Líneas de código
8
9  nombreClase objeto1;
10 //objeto1 es una instancia de la clase nombreClase.
11
12 nombreClase objeto2(parámetros);
13 /*objeto2 es una instancia de la clase, la cual contiene
14 un conjunto de parámetros de construcción de la clase.
15 El término construcción se refiere a la inicialización
16 de la clase.
17 */
18
19 // Líneas de código
```

7

## Constructor y Destructor

En C++, los constructores y destructores se declaran como parte de la definición de una clase:

- **Constructor:** Es una operación que crea un objeto y/o inicializa su estado
- **Destructor:** Es una operación que libera el estado de un objeto (es decir, memoria ... solo en c++)

8

# Ejemplos de C++

```
#include<string>
#include<iostream>

using namespace std;

class Persona {

string nombre;
int edad;

public:

Persona() {
    nombre = "NO NAME";
    edad = -1;
}

Persona(string nombre, int edad) {
    this->nombre = nombre;
    this->edad = edad;
}

void decirHola() {
    cout<<"Hola"<<this->nombre<<endl;
}

~Persona() {
    cout<<"Llamando al destructor"<<nombre<<endl;
}
};
```

```
int main() {

    Persona juan = Persona();
    Persona pedro = Persona("Pedro", 45);

    juan.decirHola();
    pedro.decirHola();

    return 0;
}
```

- No hay constructores por defecto, todos se deben implementar
- Los objetos creados (sin new) se destruyen cuando la función actual termina la ejecución

<https://repl.it/@PaulLeger/clase30>

9

## OOP en C++

```
int main() {

    Persona juan = Persona();
    Persona pedro = Persona("Pedro", 45);

    juan.decirHola();
    pedro.decirHola();

    Persona* maria = new Persona("maria", 35);
    maria->decirHola();

    delete maria;

    return 0;
}
```

- Todo elemento creado con **new**, debe ser destruido con **delete**.
- Todo objeto iniciado con punteros, usa “->” para llamar a sus elementos
- Ejemplo: **maria->decirHola();** tiene el mismo efecto que **(maria\*) .decirHola();**

<https://repl.it/@PaulLeger/clase31>

# ¿Cuándo estos objetos se destruyen?

NO NAME, PEDRO, PEDRO (FOO), MARIA

```
void foo() {
    Persona pedro = Persona("Senor F00", 45);
}

int main() {
    Persona juan = Persona();
    Persona pedro = Persona("Pedro", 45);

    juan.decirHola();
    pedro.decirHola();

    Persona *maria = new Persona ("Maria", 45);
    maria->decirHola();
    delete maria;

    foo();
    printf("ADIOS!!!!\n");

    return 0;
}
```

4/1/2022

¿Respuestas?

11

# ¿Qué muestra por pantalla?

```
void cambiarPersona(Persona p) {
    p.nombre = "Thanos";
}

int main() {
    Persona juan = Persona();
    Persona pedro = Persona("Pedro", 45);

    juan.decirHola();
    pedro.decirHola();

    cambiarPersona(pedro);
    pedro.decirHola();

    return 0;
}
```

4/1/2022

¿Respuestas?

12

**Ejemplo:** Se desea construir una clase que sea capaz de leer cadenas y desplegarlas por pantalla

```
#include<stdio.h>
#include<string.h>
#include<iostream>

using namespace std;

class Cadena {

    //private: (Por defecto privados)

    char cadena[80];

public:
    Cadena() {
        cout << "Llamando constructor VACIO" << endl;
    };

    Cadena(const char* string) {
        configurarCadena(string);
        cout << "Llamando constructor CON PARAMETRO" << endl;
        cout << cadena << endl;
    }

    void mostrar() {
        cout << "Llamando Mostrar" << endl;
        cout << cadena << endl;
    }

    void configurarCadena(const char* string) {
        strcpy(cadena,string);
    }
};
```

```
int main() {
    Cadena cadenaVacía;
    Cadena cadenaTexto("Hola");

    cout << endl << endl << "Mostrando" << endl;
    cadenaVacía.mostrar();
    cadenaTexto.mostrar();

    cout << endl << endl << "Configurando" << endl;
    cadenaVacía.configurarCadena("Nueva");
    cadenaTexto.configurarCadena("Chao");

    cout << endl << endl << "Mostrando" << endl;
    cadenaVacía.mostrar();
    cadenaTexto.mostrar();

    return 0;
}
```

Salida del Programa

Llamando constructor VACIO  
Llamando constructor CON PARAMETRO  
Hola

Mostrando  
Llamando Mostrar  
`?Q??  
Llamando Mostrar  
Hola

Configurando

Mostrando  
Llamando Mostrar  
Nueva  
Llamando Mostrar  
Chao

# Otra forma de escribir una clase

```
class Cadena {
    //private: (Por defecto privados)

    char cadena[80];

public:
    Cadena();
    Cadena(const char* string);
    void mostrar();
    void configurarCadena(const char* string);
};

Cadena::Cadena() {
    cout << "Llamando constructor VACIO" << endl;
};

Cadena::Cadena(const char* string) {
    configurarCadena(string);
    cout << "Llamando constructor CON PARAMETRO" << endl;
    cout << cadena << endl;
}

void Cadena::mostrar() {
    cout << "Llamando Mostrar" << endl;
    cout << cadena << endl;
}

void Cadena::configurarCadena(const char* string) {
    strcpy(cadena,string);
}
```

Declaración de la clase (header de la clases)

Definición de la clase

<https://repl.it/@PaulLeger/clase34>

# Encapsulación

- Recuerde que:
  - Una **abstracción de un objeto** debe ocultar su implementación a sus clientes, según el principio "**black box**".
  - De acuerdo a la teoría de lenguajes, existe encapsulación *débil* y *fuerte*. Principalmente, se usan los modificadores de métodos para moverse entre ambos tipos de encapsulación. C++ está entre débil y fuerte.

15

# Encapsulación

```
1 class RegPersonal{
2     public:
3
4     char *empleadoNombre();
5     int empleadoNombreSeguroSocial();
6     char *empleadoDepartamento();
7
8     protected:
9     void inicEmpleadoNombre(char *nombre);
10    void inicEmpleadoNumSeguroSocial(int numero);
11    void inicEmpleadoDepartamento(char *depto);
12    void inicEmpleadoSalario(float salario);
13    float empleadoSalario();
14
15    private:
16    char *nombre[100];
17    int numeroSeguroSocial;
18    char *departamento[10];
19    float salario;
20 };
```

Similar a Java, C++ soporta **public**, **protected** y **private**.

Poner atención en la sintaxis para su uso.

Cuando no se escribe el "*modifier*", se asume **private**

¿Cómo es en Java cuando no se escribe el modifier?

16



# Encapsulamiento

¿Cuál es el objetivo de tener “cosas” públicas y otras privadas?

## Sección Privada (*private*):

- Es una declaración que está visible solamente a la misma clase y a sus “friends” (ver próxima slide).
- Especifica a los métodos que son accesibles solamente dentro del alcance de la clase.

## Sección Pública (*public*):

- Es una declaración que está visible a todos los clientes.
- Usualmente especifica la interface a los métodos que forman la base para la reusabilidad de la clase.
- Estos métodos pueden ser invocados desde el exterior del alcance de la clase enviando mensaje a los objetos de una clase dada.

## Sección Protegida (*protected*):

- Es una declaración que está visible solamente a la misma clase, a sus subclases y a sus “friends”.

17

# Tiempo de Vida de un Objeto

## Crear explícitamente un objeto: **New**

- En C++, **new** crea un objeto a partir de almacenamiento disponible en el **heap**.
- Siempre que un objeto es creado, el constructor definido en la clase del objeto, es invocado en forma automática.
- Un objeto existe aunque no tenga referencias a él (**¡MEMORY LEAK OTRA VEZ!**)
- En Java, los objetos son destruidos automáticamente durante el proceso de “**garbage collection**” si no existen referencias.

18

# Tiempo de Vida de un Objeto

## Destruir explícitamente un objeto: Delete

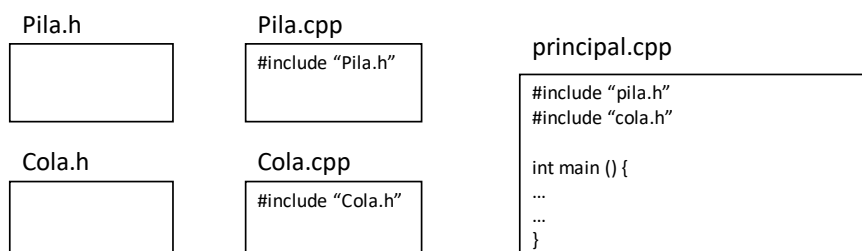
- En C++, siempre que un objeto es destruido, el destructor definido en la clase del objeto, es invocado en forma automática.
- El método **delete** se puede aplicar sólo a los objetos creados explícitamente en el **heap**.

19

# Modularidad

- La **modularización** consiste en dividir un programa en módulos que se puedan crear, mantener y potencialmente por separado, pero que tienen conexiones con otros módulos.
- En C++, la práctica usual es situar las interfaces de los módulos en archivos con extensión .h (archivos de cabecera) y las implementaciones de los módulos se sitúan en archivos con extensión .cpp.

## Ejemplo



20

# Diferencias entre “.h” y “.cpp”

## Persona.h

```
#ifndef PERSONA_H
#define PERSONA_H

class Persona{
private:
    char* nombre;
    int rut;

public:
    Persona(char*, int);
    char* getNombre();
    int getRut();
};
#endif
```

## Persona.cpp

```
#include <iostream.h>
#include "Persona.h"

Persona::Persona(char* n, int r) {
    nombre = n;
    rut = r;
}

char* Persona::getNombre() {
    return nombre;
}

int Persona::getRut() {
    return rut;
}
```

## main.cpp

```
#include <iostream.h>
#include "Persona.h"
#include <conio.h>

int main() {

    //Alternativa 1 para crear un objeto
    Persona p = Persona ("Juan", 123);

    //Alternativa 2 para crear un objeto
    Persona* punt = new Persona ("Loreto", 456);

    cout << p.getNombre() << endl;
    cout << (*punt).getNombre() << endl;
    cout << punt ->getNombre() << endl;
    cout << punt ->getRut() << endl;
    getch();
    return 0;
}
```

## Salida del Programa

```
Juan
Loreto
Loreto
456
```

<https://repl.it/@PaulLeger/clase36>

21

# Friend functions

“friend functions” o (“class friend”) son funciones que no pertenecen a una clase, pero pueden acceder a todos sus elementos. **Generalmente, no se recomienda este uso, pues “rompe” la encapsulación.**

¿Cuándo usarla? Cuando tengas dos tipos de objetos que no son del mismo tipo, pero trabajan de manera cercana. Ejemplos, amigos entre todos los elementos de un auto.

```
class Cadena {
    //private: (Por defecto privados)
    char cadena[80];

public:
    Cadena();
    Cadena(const char* string);
    void mostrar();
    void configurarCadena(const char* string);
    friend void eliminarEspacios(Cadena *c);
};
```

```
void eliminarEspacios(Cadena *c) {
    char nuevoTexto[80];
    int j = 0;

    for(int i = 0; i < strlen(c->cadena); ++i) {
        if (c->cadena[i] != ' '){
            nuevoTexto[j++] = c->cadena[i];
        }
    }

    nuevoTexto[j] = '\0';
    strcpy(c->cadena, nuevoTexto);
}
```

```
int main() {
    Cadena cadenaVacía;
    Cadena cadenaTexto("Hola");

    cout << endl << endl << "Mostrando" << endl;
    cadenaVacía.mostrar();
    cadenaTexto.mostrar();

    cout << endl << endl << "Configurando" << endl;
    cadenaVacía.configurarCadena("Nueva");
    cadenaTexto.configurarCadena("Hola Mundo");

    cout << endl << endl << "Mostrando" << endl;
    cadenaVacía.mostrar();
    cadenaTexto.mostrar();

    cout << endl << endl << "Eliminando espacios en blancos" << endl;
    eliminarEspacios(&cadenaTexto);
    cadenaTexto.mostrar();

    return 0;
}
```

<https://repl.it/@PaulLeger/clase37>

22

# Ejercicio: “Adivina el número”

Implementar el **juego adivina el número** en C++. Las reglas del juego son:

- El computador genera un número aleatorio.
- El computador pregunta al Jugador Humano si conoce el número.
- El Jugador Humano acierta al número, se termina el juego y el jugador gana 100 puntos.
- Si el Jugador Humano, no acierta:
  - Si el número es más bajo del que generó el computador, se le indica que no acertó y que el número debe generado es más grande.
  - Si el número es más grande del que generó el computador, se le indica que no acertó y que el número generado debe ser menor.
  - Por cada intento el jugador Humano pierde | Número Jugador Humano - Número generado por computador| puntos.
- El jugador Humano puede realizar varios intentos hasta acertar el número.
- El jugador puede obtener puntaje negativo.

4/1/2022

23

# Ejercicio: “Adivina el número”



<https://replit.com/team/EstructuraDeDatos/ClasesC02Adivina-el-numero-parte-I>

4/1/2022

24