

# Tipos de Datos Abstractos

Mauricio Avilés

# Contenido

- Abstracción
- Resolución de problemas
- Abstracción en los lenguajes de programación
  - Abstracciones de control
  - Abstracciones de datos
- Tipos de Datos Abstractos
  - Estructura
  - Ventajas
  - Especificación



# Lectura

- Capítulo 1
  - Joyanes, Aguilar, & Martínez. (2007). Estructura de datos en C++. Madrid: McGraw-Hill Interamericana.

# Abstracción

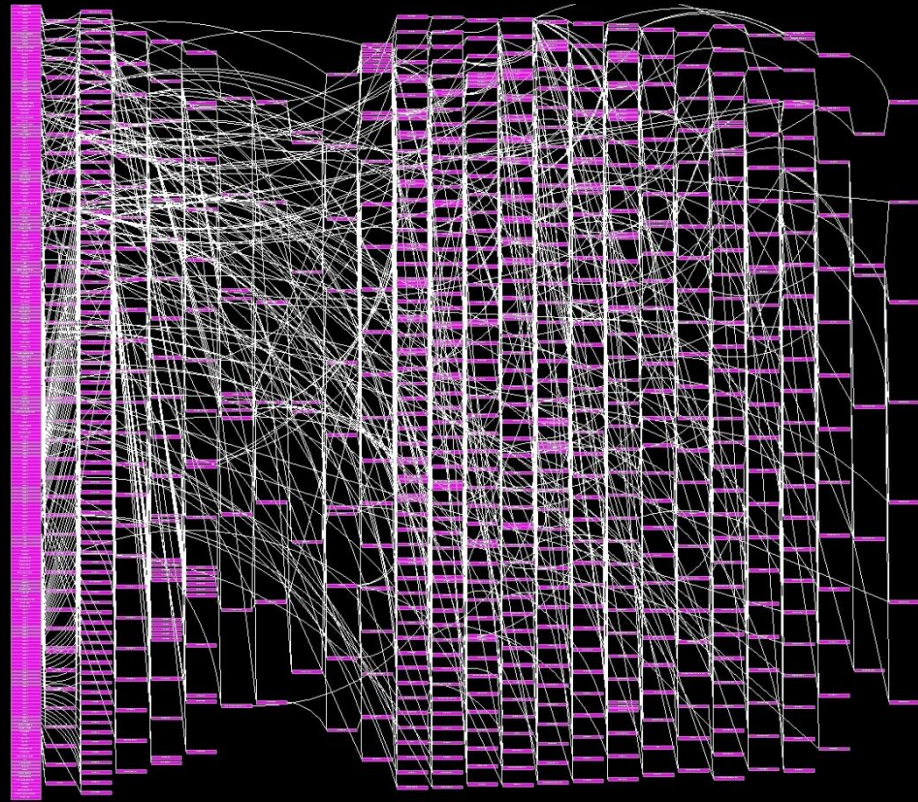
*“Los humanos hemos desarrollado una técnica excepcionalmente potente para tratar la **complejidad**: abstraernos de ella. Incapaces de dominar en su totalidad los objetos complejos, se **ignoran** los detalles **no esenciales**, tratando en su lugar con el modelo ideal del objeto y centrándonos en el estudio de sus aspectos esenciales” - Wulft*



- Humanos → **capacidad** de abstraer el pensamiento
- **Omitir detalles** para comprender sistemas
- Abstracción → proceso de **excluir información** no relevante a un problema



- Esencial para el funcionamiento de la **mente humana**
- Crucial para comprender la **complejidad** del mundo



# Proceso mental natural

---

Construcción de modelos mentales

---

Vista simplificada de cómo funcionan las cosas

---

Es lo mismo que el diseño de software

---

El modelo se puede manipular en una computadora

---

El modelo debe ser más sencillo que el real

---

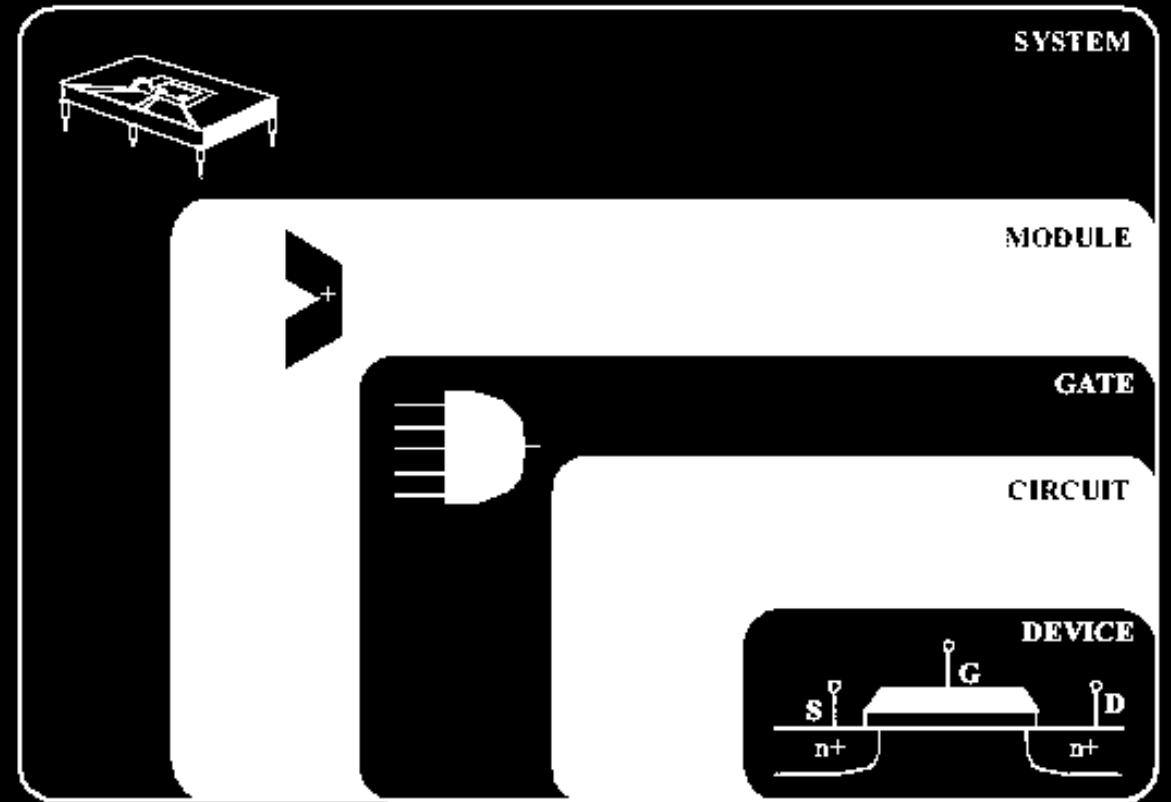
# Niveles de abstracción

Los problemas se  
pueden visualizar  
desde diferentes  
**niveles** de  
abstracción



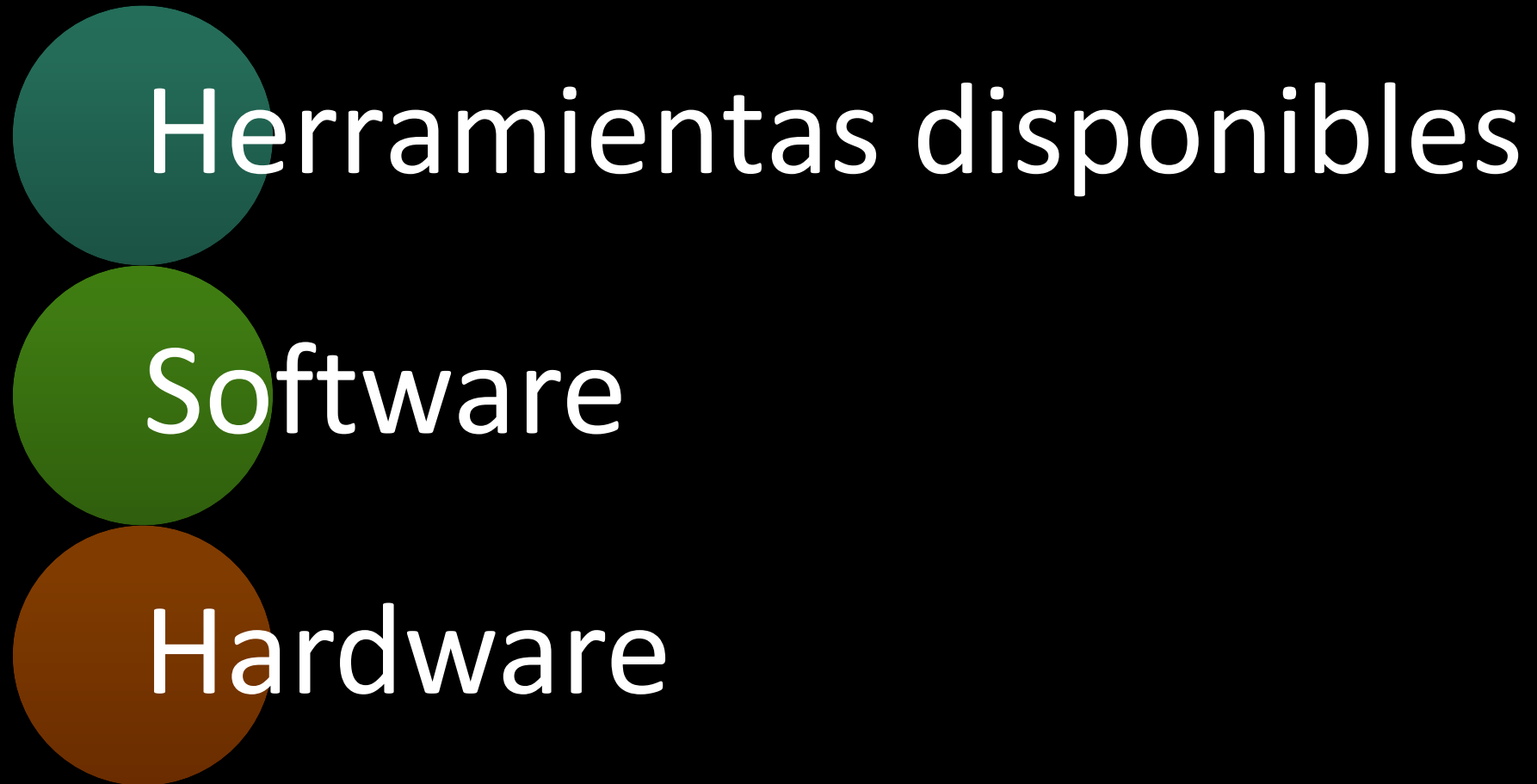


- Nace una **jerarquía** de abstracciones
- Objetivos **generales**
  - Nivel más **alto**
- Objetivos **específicos**
  - Niveles más **bajos**



Conforme se desciende de nivel los aspectos de la solución se hacen evidentes

La cantidad de niveles depende de:



Un problema se divide en **subproblemas** de complejidad razonable que se pueden resolver independientemente



# Resolución de problemas

- Proceso completo de tomar la descripción de un **problema** y desarrollar un **programa** que lo resuelva
- Solución → **algoritmos** + almacenamiento de **datos**





# Abstracción en los lenguajes de programación

Abstracciones  
de control

Abstracciones  
de datos



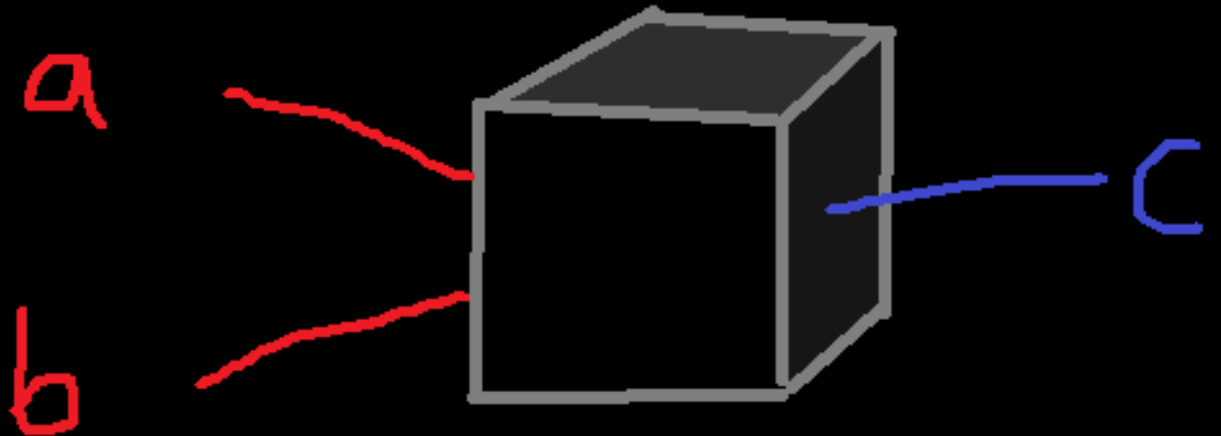
# Abstracciones de control

- Describen el **orden** en que se ejecutan sentencias o grupos de sentencias
  - Sentencias de **bifurcación** (if)
  - Sentencias de **repetición** (while/for)
  - **Unidades** de programa (subrutinas)
    - Abstracción procedimental
    - Diseño descendente
  - Manejo de **errores** (excepciones/try)
  - **Plantillas** (templates)



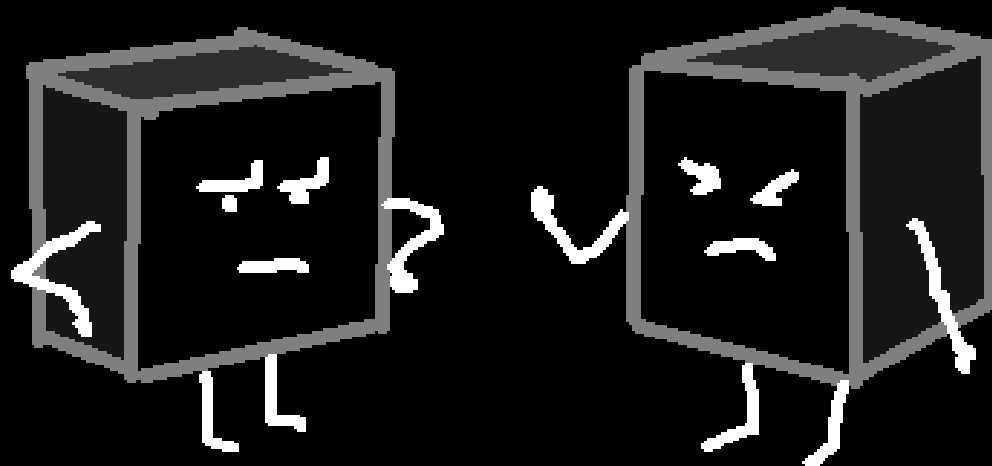
# Abstracción procedimental

- Cada unidad algorítmica puede verse como una **caja negra**
- Tiene entradas y salidas pero **no dice** cómo lo hace
- Separa el **propósito** de un programa de su **implementación**
- **Modularidad** → romper la solución en módulos
- Especificar cada módulo **antes** de programarlo
- Se pueden **cambiar los algoritmos** de un módulo sin afectar el resto de la solución



# Ocultación de información

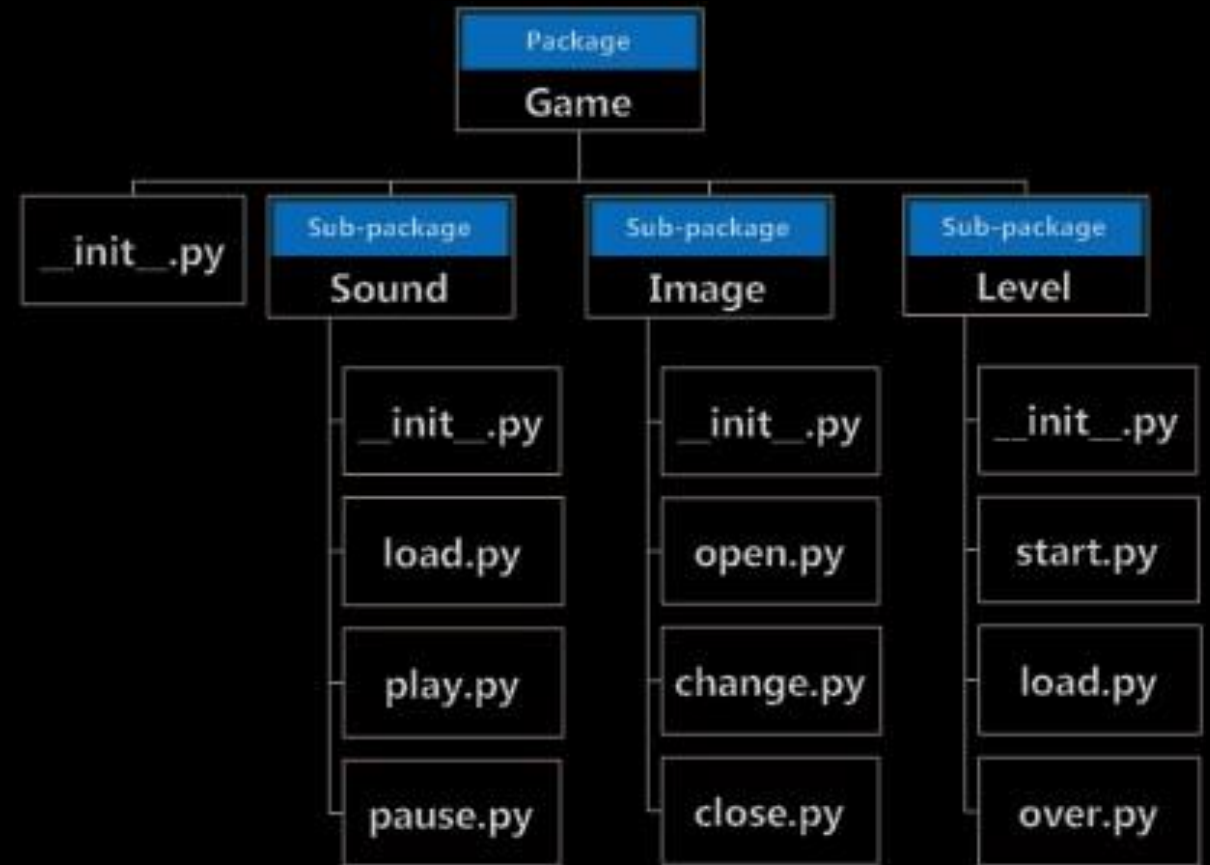
- La abstracción procedimental **oculta los detalles** de una caja negra
- La ocultación de información asegura que las cajas negras **no pueden ver los detalles** ocultos de otras cajas negras






# Diseño descendente

- Descomponer una tarea en sucesivos niveles de detalle
- Dividir el programa en módulos independientes
- Los módulos de nivel inferior realizan tareas muy específicas



En C++ la abstracción  
procedimental se logra por medio  
de métodos de clases



Programación  
Orientada a  
Objetos

# Abstracción de datos

- Técnica de programación que permite **definir nuevos tipos** de datos
- **Adecuados** a la aplicación que se está creando
- Programas cortos, legibles y flexibles
- Es un tipo de dato que se utiliza **sin considerar su implementación**

*Tipos de Datos*

**ABSTRACTOS**

**TDA**



# Tipos de Datos Abstractos (TDAs)

- Ampliar el lenguaje por medio de tipos **definidos por el usuario**
- Para crearlos, un lenguaje de programación necesita:
  - **Asociar** una construcción de datos con sus **operaciones**
  - **Ocultar detalles** del tipo de dato a quienes lo utilizan
- La **orientación a objetos** es una forma de crear tipos de datos abstractos

Representación  
interna  
Estructuras

Datos

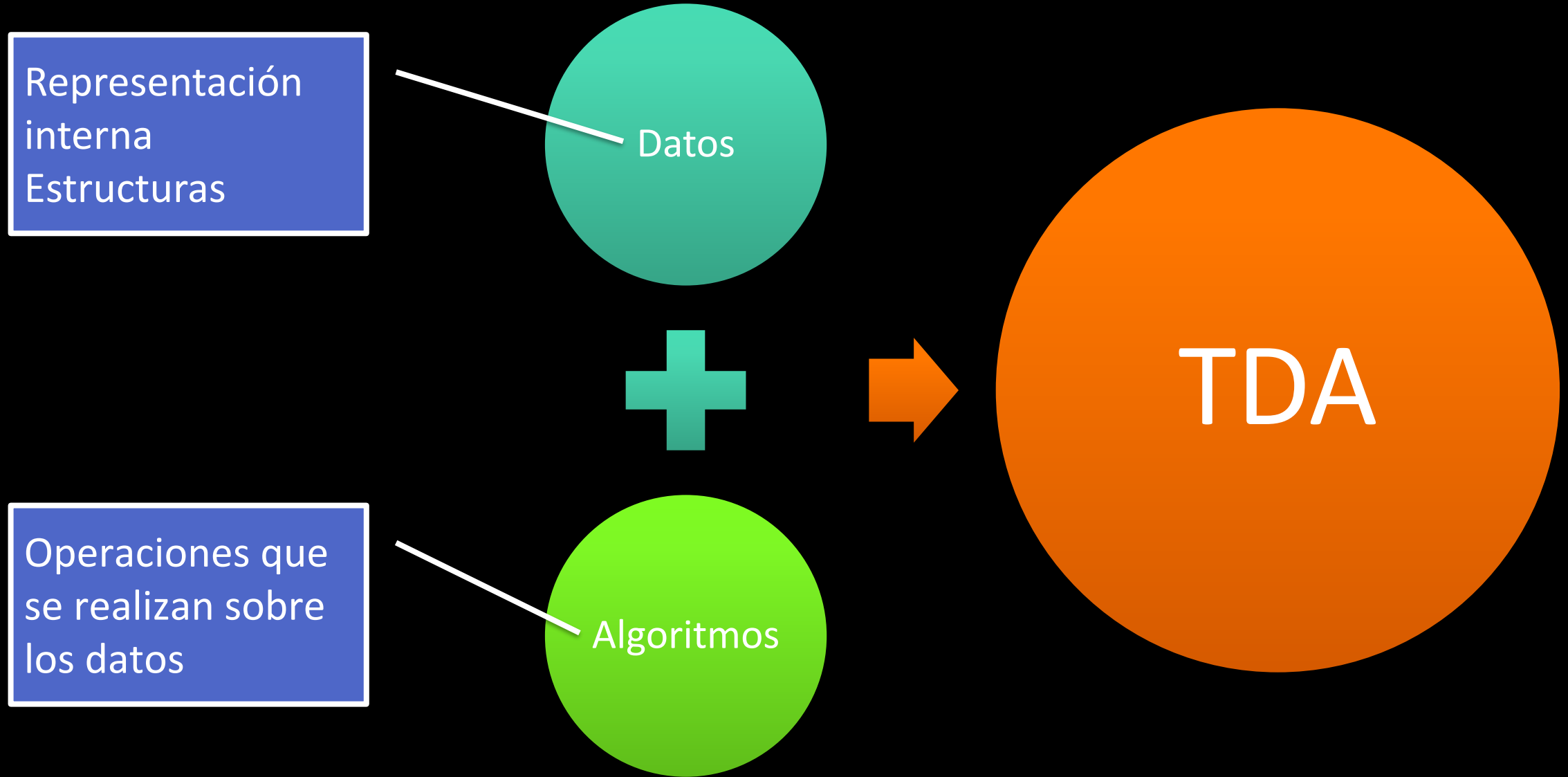


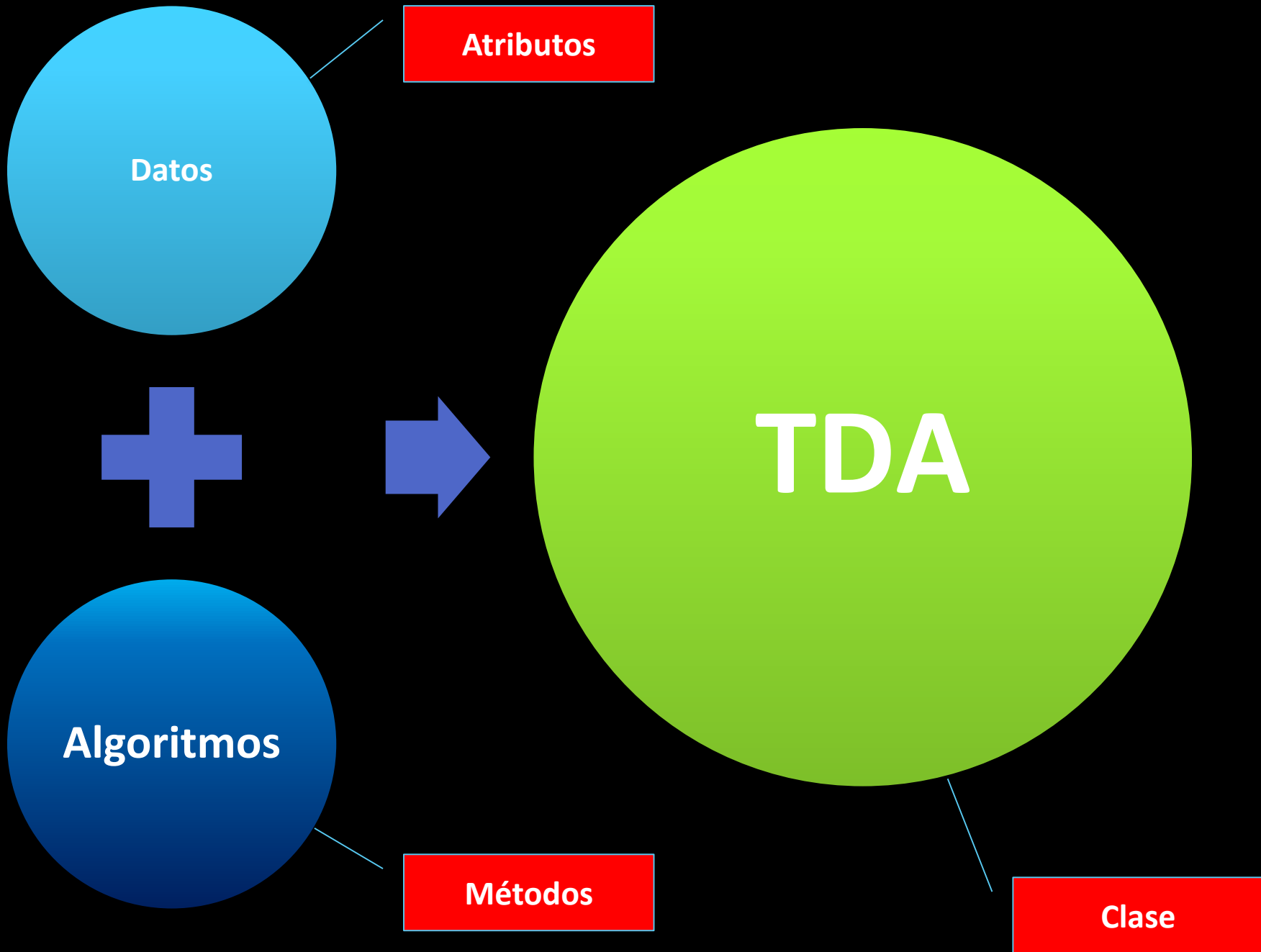
Operaciones que  
se realizan sobre  
los datos

Algoritmos



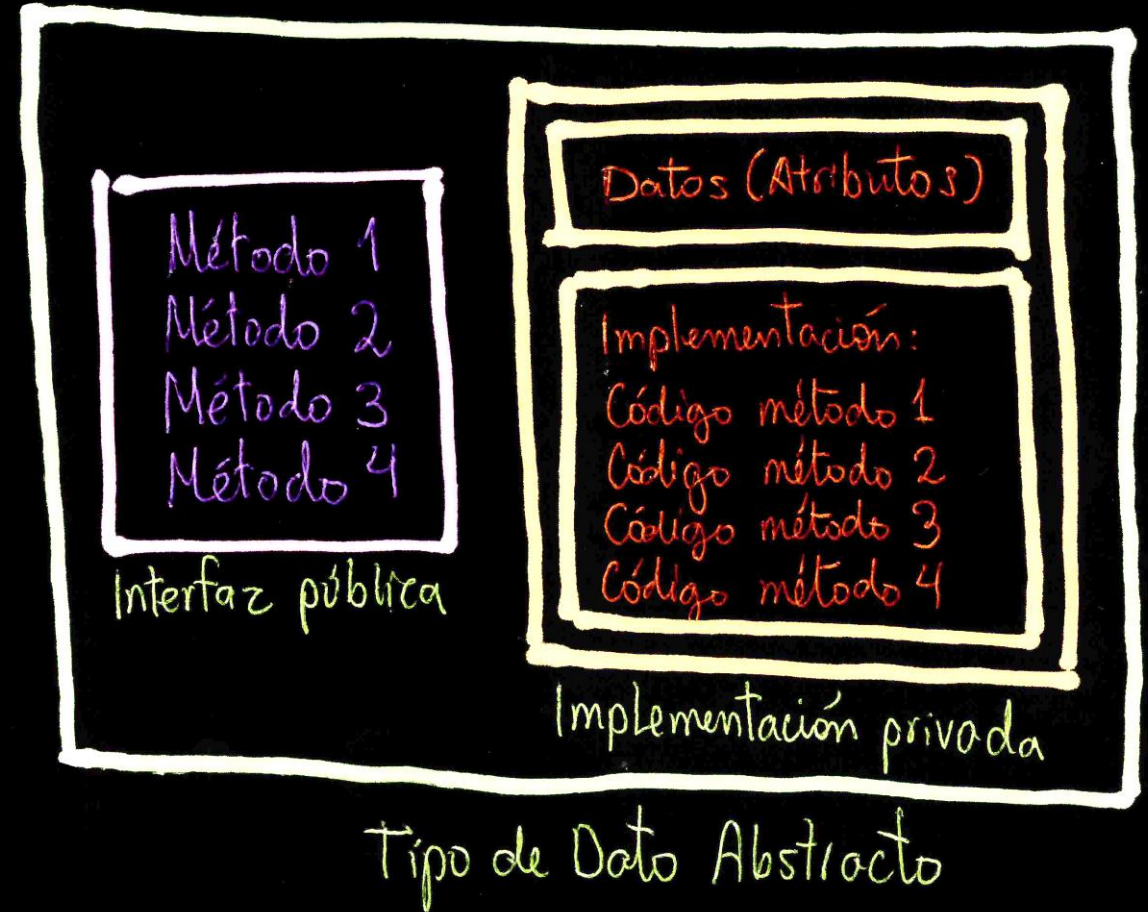
TDA





# Estructura de un TDA

- Interfaz pública: **operaciones** que se pueden realizar con el tipo
- Implementación: **estructuras y algoritmos** utilizados internamente
- Las estructuras y algoritmos utilizados internamente son **invisibles** al usuario o clientes

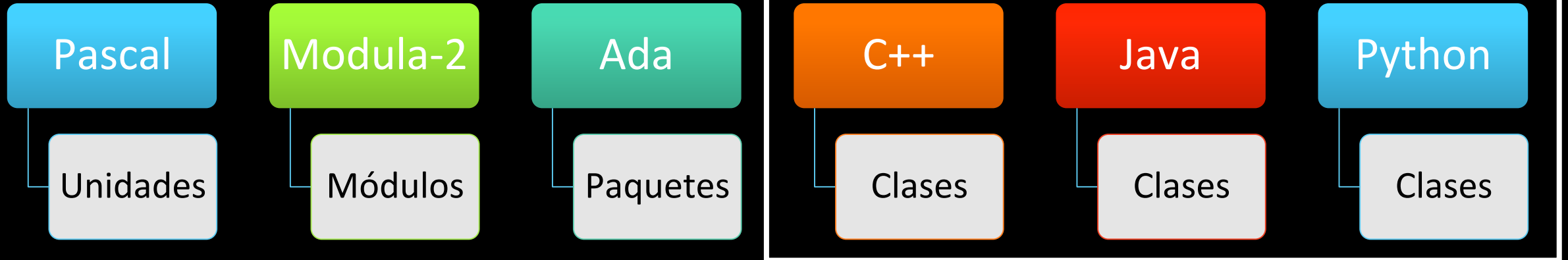




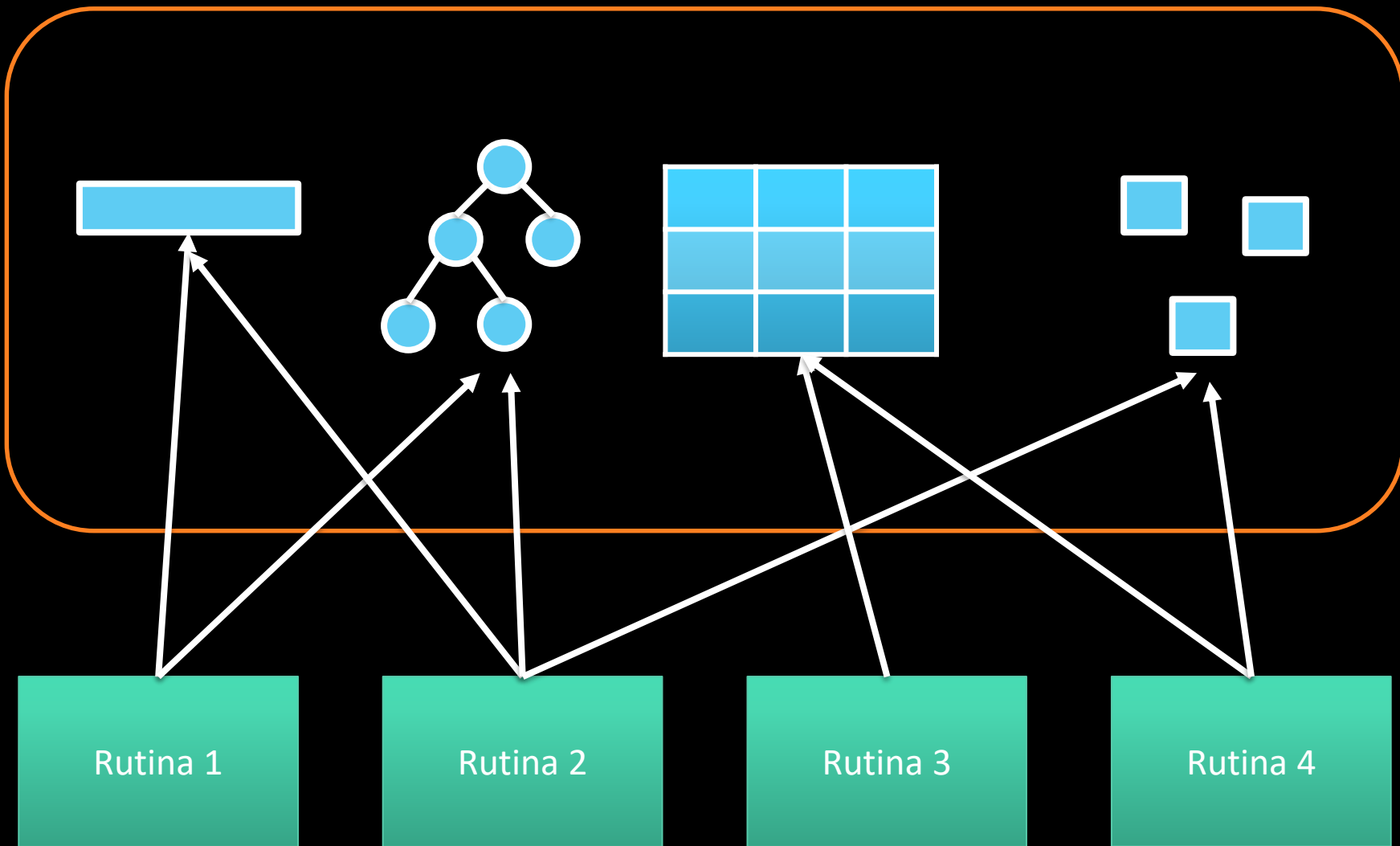
# Ventajas de los TDAs

- Facilita la conceptualización del mundo real (modelado)
- Facilita la comprobación de tipos (robustez)
- Separación de implementación (mantenibilidad)
- Reutilización de componentes (modularidad)
- Permiten extender funcionalidades (extensibilidad)
- Agrupa operaciones y representaciones (semántica)

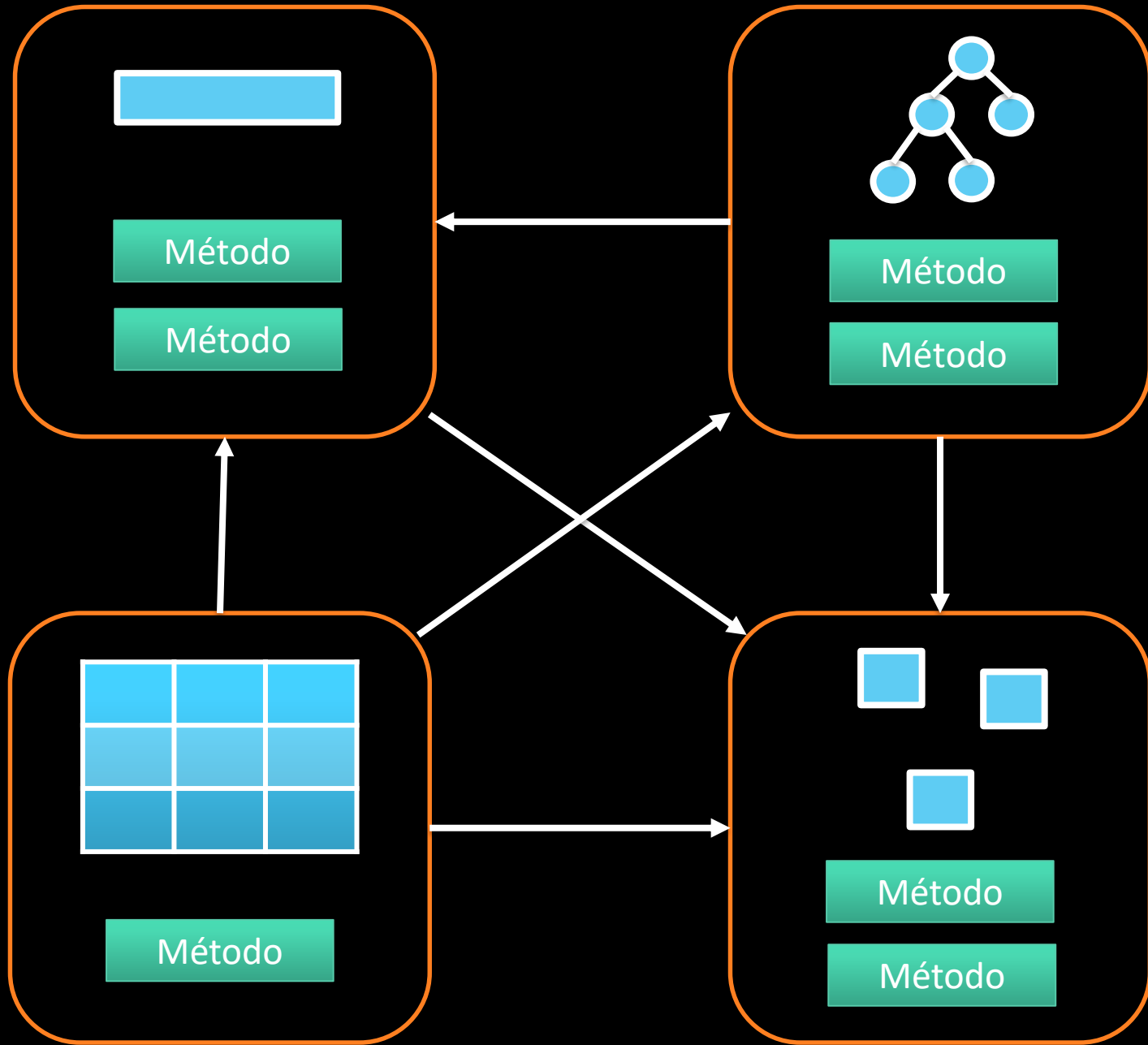
# ¿Cómo se implementan los TDAs?



# Programación imperativa



# Programación orientada a objetos



# Abstracción

```
graph LR; A[Abstracción] --- B[Encapsulamiento]; A --- C[Herencia]; A --- D[Polimorfismo]; A --- E[Modularidad];
```

Encapsulamiento

Herencia

Polimorfismo

Modularidad

# Especificación de un TDA

- Tipo de dato abstracto para representar un dado
- El dado puede ser de cualquier cantidad de lados



```
#include <stdexcept>

using namespace std;

class Dado {
private:
    int ladoActual;
    int cantidadLados;
public:
    Dado(int lados);
    int obtenerLadoActual() throw (runtime_error);
    void lanzar();
};
```

```
#include "Dado.h"
#include <time.h>
#include <cstdlib>

Dado::Dado(int lados) {
    ladoActual = 0;
    cantidadLados = lados;
    srand(time(0));
}

int Dado::obtenerLadoActual() throw (runtime_error) {
    if (ladoActual == 0) {
        throw runtime_error("El dado no ha sido lanzado.");
    }
    return ladoActual;
}

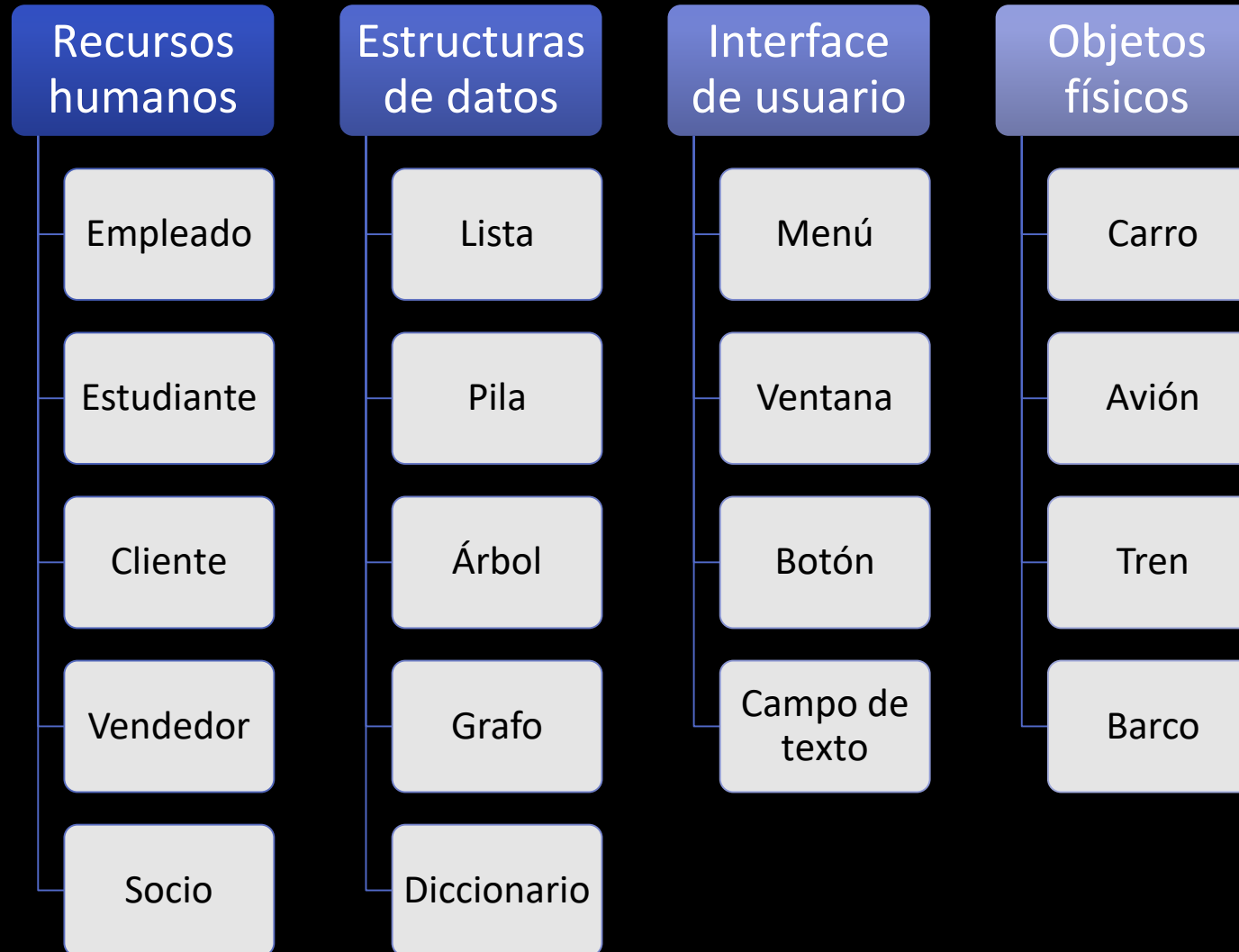
void Dado::lanzar() {
    ladoActual = rand() % cantidadLados + 1;
}
```

```
#include <iostream>
#include <limits>
#include "Dado.h"

using namespace std;

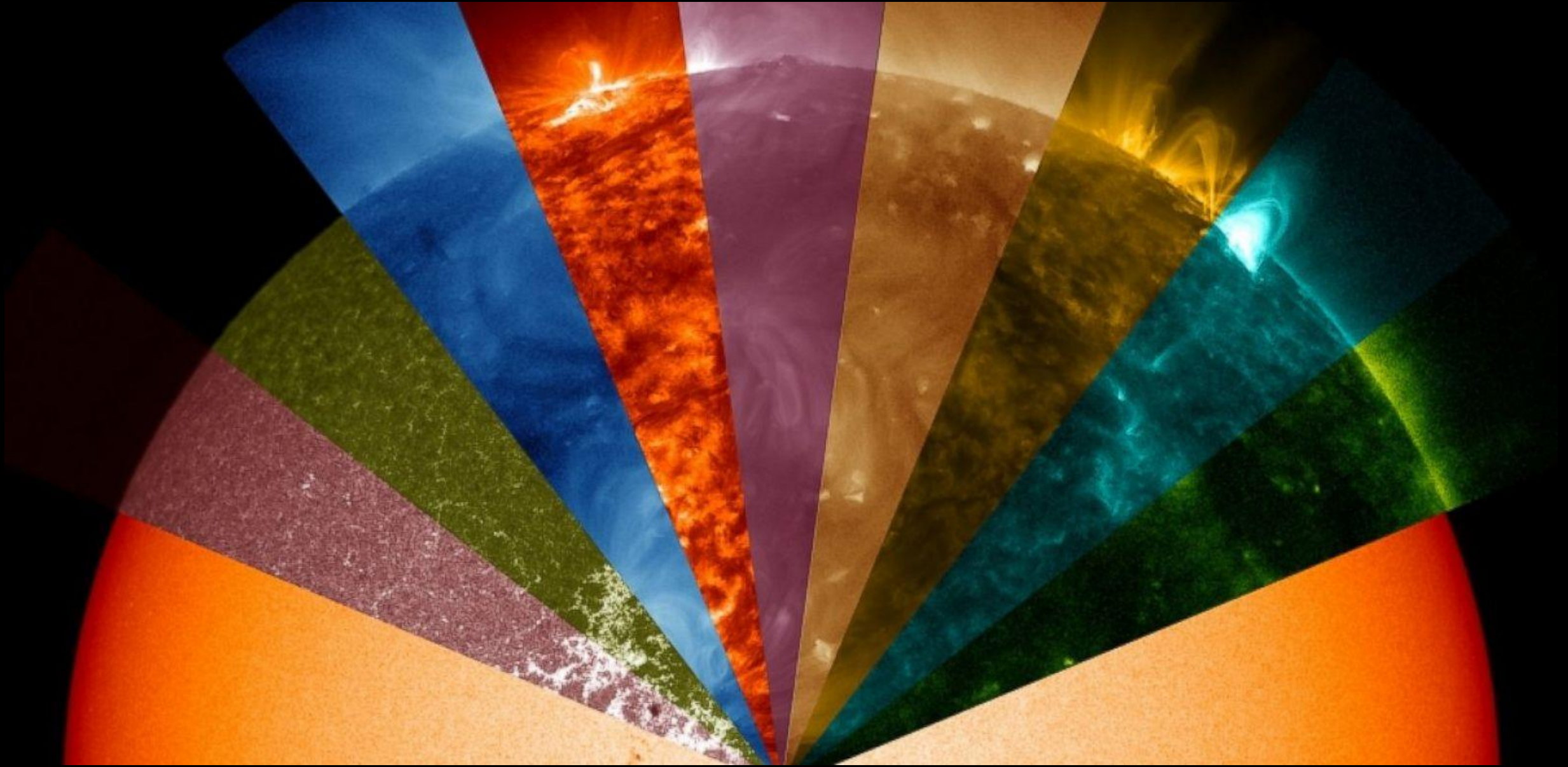
int main() {
    int lados = 0;
    int resultado;
    cout << "Digite la cantidad de lados del dado: ";
    while (!(cin >> lados) || lados < 1) {
        cout << "Debe ser un número entero positivo." << endl;
        cout << "Digite la cantidad de lados del dado: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    Dado *dado = new Dado(lados);
    dado->lanzar();
    resultado = dado->obtenerLadoActual();
    cout << "Resultado del lanzamiento: " << resultado;
    delete dado;
    return 0;
}
```

# Ejemplos de clases



# Preguntas

1. Defina con sus propias palabras el concepto de abstracción.
2. ¿Porqué es conveniente dividir un problema en subproblemas para resolverlo?
3. Compare y contraste las abstracciones de control con las abstracciones de datos.
4. Explique los conceptos de abstracción procedimental, ocultación de información y diseño descendente.
5. Defina qué es un tipo de dato abstracto.
6. Explique la estructura de un dato abstracto.
7. ¿Cómo se implementan los tipos de datos abstractos en el paradigma orientado a objetos? Explique.
8. Indique ejemplos propios de tipos de datos abstractos que usted podría crear.



# Tipos de Datos Abstractos

Mauricio Avilés