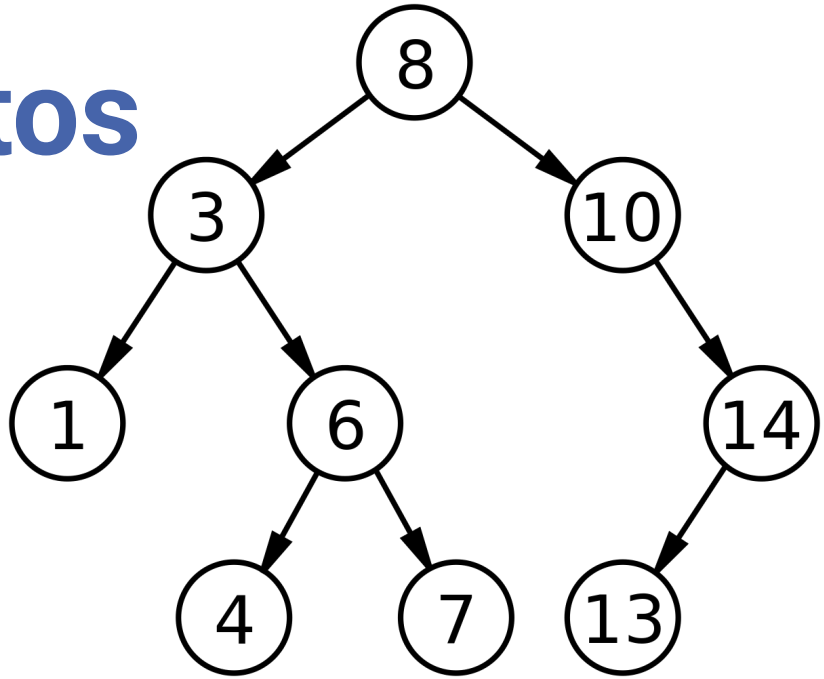
Estructuras de datos

Tipos de datos abstractos

Abstracción y ejemplos



Franco Cerda

Créditos: Sebastián Torrealba

Tipos de datos abstractos

Son una forma de encapsular datos y las operaciones que se pueden realizar sobre ellos en una única unidad. Estos proporcionan una interfaz clara y definida, ocultando los detalles internos de implementación y permitiendo que los datos se manipulen de manera segura y eficiente

Abstracción

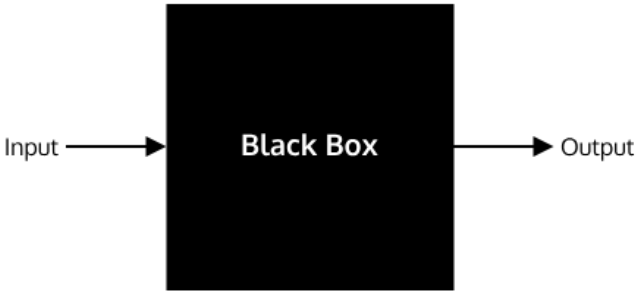
Se aísla conceptualmente una propiedad o función concreta de un objeto, y se piensa qué es, ignorando otras propiedades del objeto en cuestión.

¿Por qué es útil?

Permite concentrarse en **que hace** en vez del **como lo hace**

¿Por qué son MUY utiles en estructuras de datos?

- Permiten crear estructuras modulares con fines especificos que permiten ser reutilizadas.
- Aseguras un buen rendimiento si el **TDA** está bien implementado
- La estandarización permite que sea más rapido el desarrollo de un programa



¿Cómo programar un tipo de dato abstracto?

En **C++** tenemos acceso a las clases. Las clases nos permiten encapsular, abstraer, crear métodos, *herencia y polimorfismo**

```
class TDA {
  private:
    int ejemplo;
  public:
    void hola() {
      cout << "Método" << endl;
    }
};
```

** Herencia y polimorfismo son conceptos que verán en el ramo Lenguajes de Programación*

Vocabulario importante sobre las clases

- **Clase:** Es una plantilla para crear objetos, similares a los *structs*
- **Objeto:** Es la instancia de una clase
- **Atributo:** Variables que representan el estado de un objeto
- **Método:** Funciones que realizan operaciones **sobre** el objeto
- **Constructor:** Un método especial que se llama al crear el objeto
- **Destructor:** Un método especial que se llama cuando se destruye un objeto
- **Modificador de acceso:** Controlan la visibilidad de los miembros de la clase

Ejemplo de un tipo de dato abstracto

En caso de que queramos crear una fracción, una opción sería esta:

```
class Fraccion {
  private:
    int numerador, denominador;
  public:
    int obtenerNumerador() { return this->numerador; }
    int obtenerDenominador() { return this->denominador; }
    void multiplicar(Fraccion b) {
      this->numerador *= b.numerador;
      this->denominador *= b.denominador;
    }
  };
```

¡Fin!