

Patrones de diseño

Electrónica IV

Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

<https://facetvirtual.facet.unt.edu.ar/course/view.php?id=165>

Resumen de Objetos

- ▶ El análisis, diseño y programación orientados a objetos plantean una alternativa en la comprensión del problema y el desarrollo de la solución.
- ▶ Los objetos son abstracciones de entidades del mundo real que participan en el problema

Análisis orientado a objetos

- ▶ Los sustantivos del enunciado se corresponden con las clases en el diseño.
- ▶ Los adjetivos del enunciado se corresponden con los atributos de la clase.
- ▶ Los verbos del enunciado se corresponden con los métodos de la clase.

Patrones de diseño

- ▶ Son técnicas para resolver problemas comunes en el desarrollo de software.
- ▶ Para que una solución sea considerada un patrón debe poseer ciertas características:
 - ▶ Debe haber comprobado su efectividad resolviendo problemas similares anteriormente
 - ▶ Debe ser reutilizable, es aplicable a diferentes problemas de diseño en distintas circunstancias

Patron de programación ADT

- ▶ Las declaraciones y definiciones publicas se hacen en los archivos de cabecera (.h)
- ▶ Las declaraciones y definiciones privadas se hacen en los archivos de código fuente (.c)
- ▶ Por cada clase, en el archivo de cabecera, se declara una estructura y se define un tipo de datos como un puntero a esa estructura.
- ▶ Los métodos públicos de la clase se declaran como funciones en el archivo de cabecera.

Patron de programación ADT

- ▶ El constructor crea un nuevo objeto en un estado conocido y devuelve un puntero a la instancia, es decir al conjunto de valores que representa su estado.
- ▶ Los métodos reciben como primer parámetro el puntero a la instancia del objeto.
- ▶ La estructura de datos con los atributos de un objeto es privada y su definición se completa en el archivo de código (.c).
- ▶ Se pueden declarar y definir métodos privados, esto se hace también en el archivo de código (.c)

Llamadas a funciones

- ▶ Al llamar a una función:
 - ▶ Se almacena el valor actual del PC para poder continuar la ejecución al terminar la rutina
 - ▶ Se cambia el valor del PC para ejecutar la primera instrucción de la rutina
- ▶ Se puede pensar el nombre de una función como un puntero, constante, a la primera instrucción
- ▶ Normalmente, un punto concreto del código, se invoca siempre a la misma función

Punteros a funciones

- ▶ Sin embargo se puede definir un puntero a funciones variable
- ▶ El puntero debe ser tipado con los argumentos que recibe y el valor de retorno de la función
- ▶ Esto permite cambiar la función que se invoca en tiempo de ejecución
- ▶ También permite resolver las dependencias de código en forma externa (dependencias explícitas)

Funciones de Callback

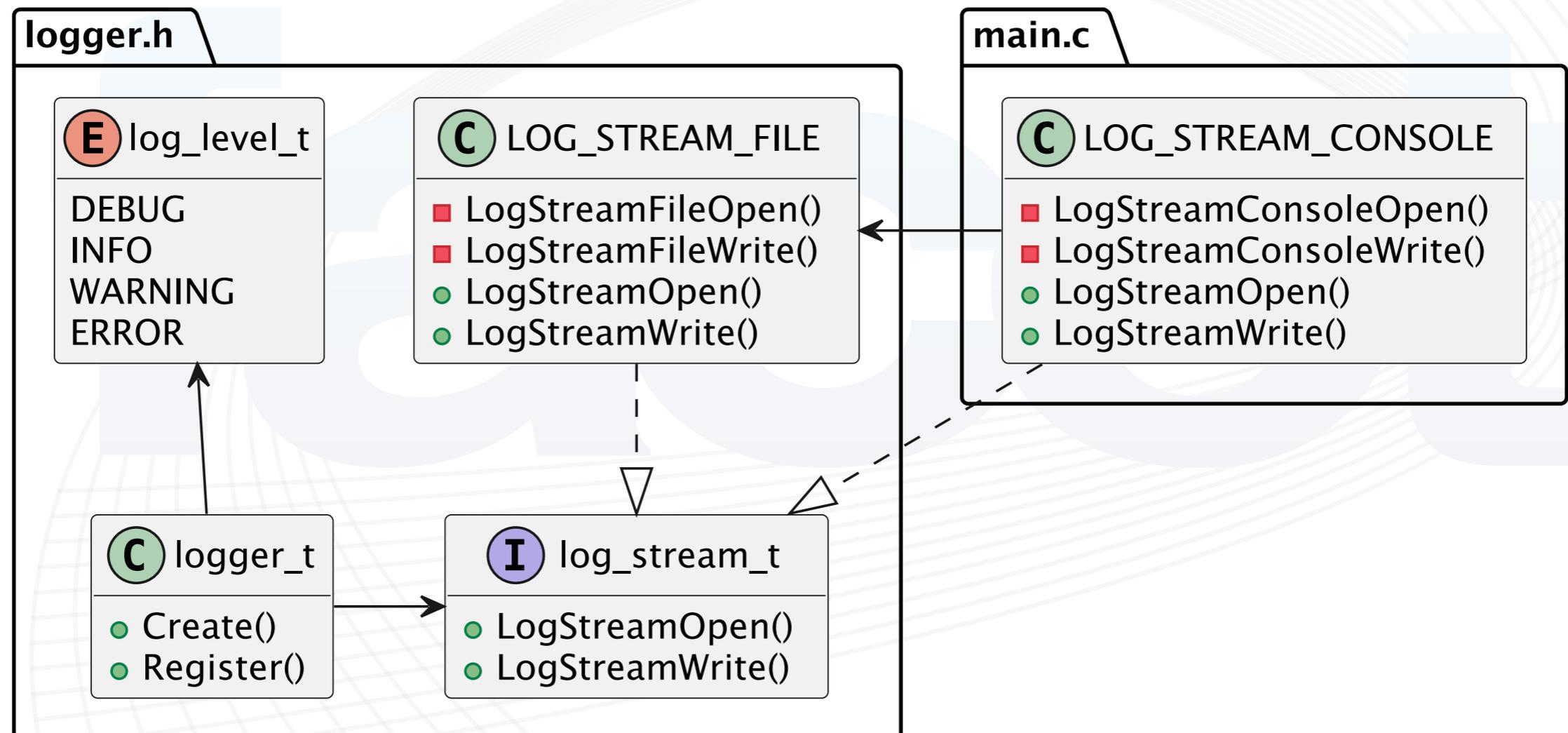
- ▶ Es un mecanismo que permite utilizar una función como argumento en el llamado a otra función
- ▶ La referencia a la función de callback se puede almacenar en una variable para usarla después
- ▶ Esto permite modificar el comportamiento en tiempo de ejecución y/o en forma externa al código que ejecuta la función de callback

Interface

- ▶ Una interface es un conjunto de métodos que permiten que objetos no relacionados puedan interoperar
- ▶ Se define un puntero a función por cada tipo de operación
- ▶ En una estructura se agrupan todos los punteros definidos para establecer una API

Ejemplo

- ▶ Una clase para registrar eventos que permite cambiar el medio físico usado para el registro

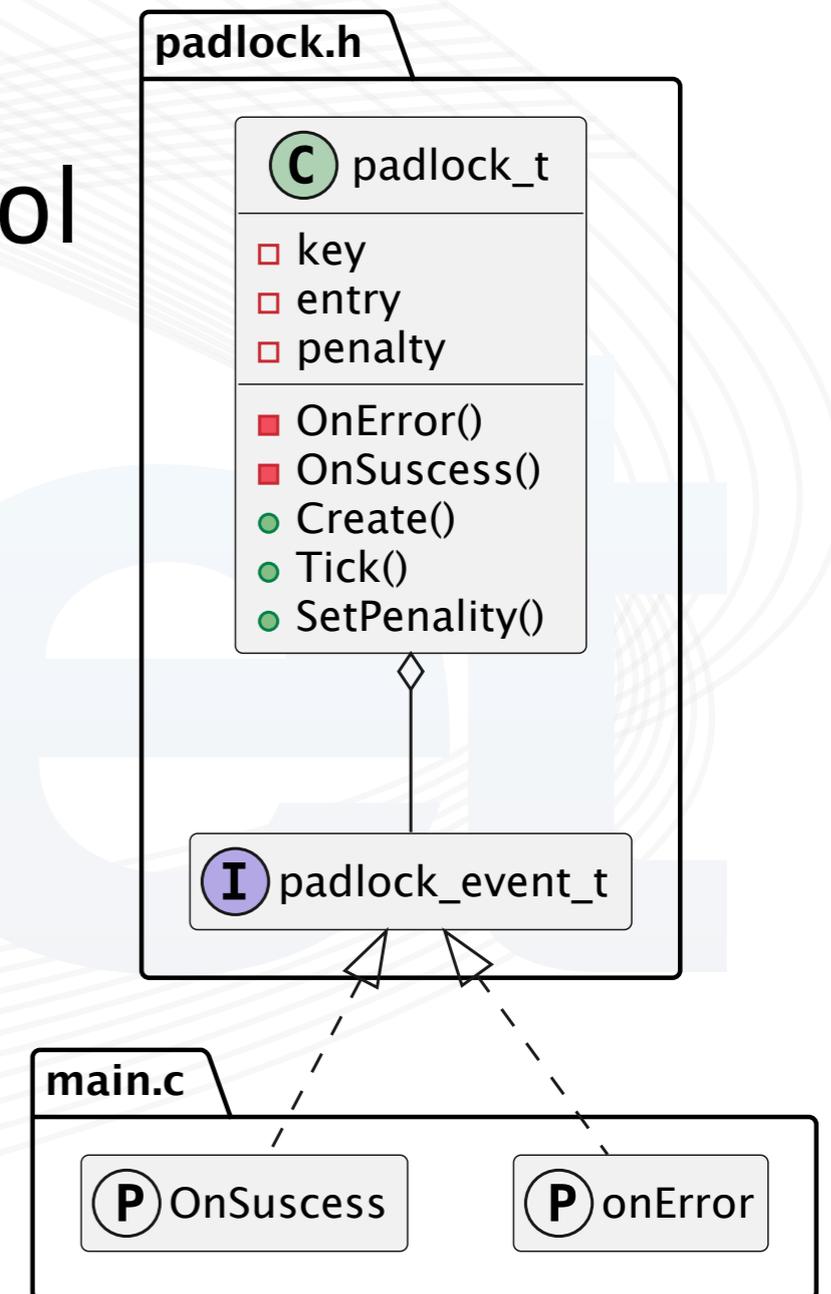


Eventos

- ▶ Un componente puede utilizar las funciones de callback para implementar eventos
- ▶ El componente define un conjunto de situaciones internas que quiere comunicar el exterior
- ▶ El usuario proporciona las funciones que serán llamadas por el componente cuando el evento interno ocurra

Ejemplo: un candado reutilizable

- ▶ La clase candado implementa el ingreso tecla por tecla y el control con una clave preestablecida
- ▶ Permite gestionar tiempos de penalización por errores en el ingreso
- ▶ Las acciones ante un ingreso correcto y un error en la clave se gestionan como eventos

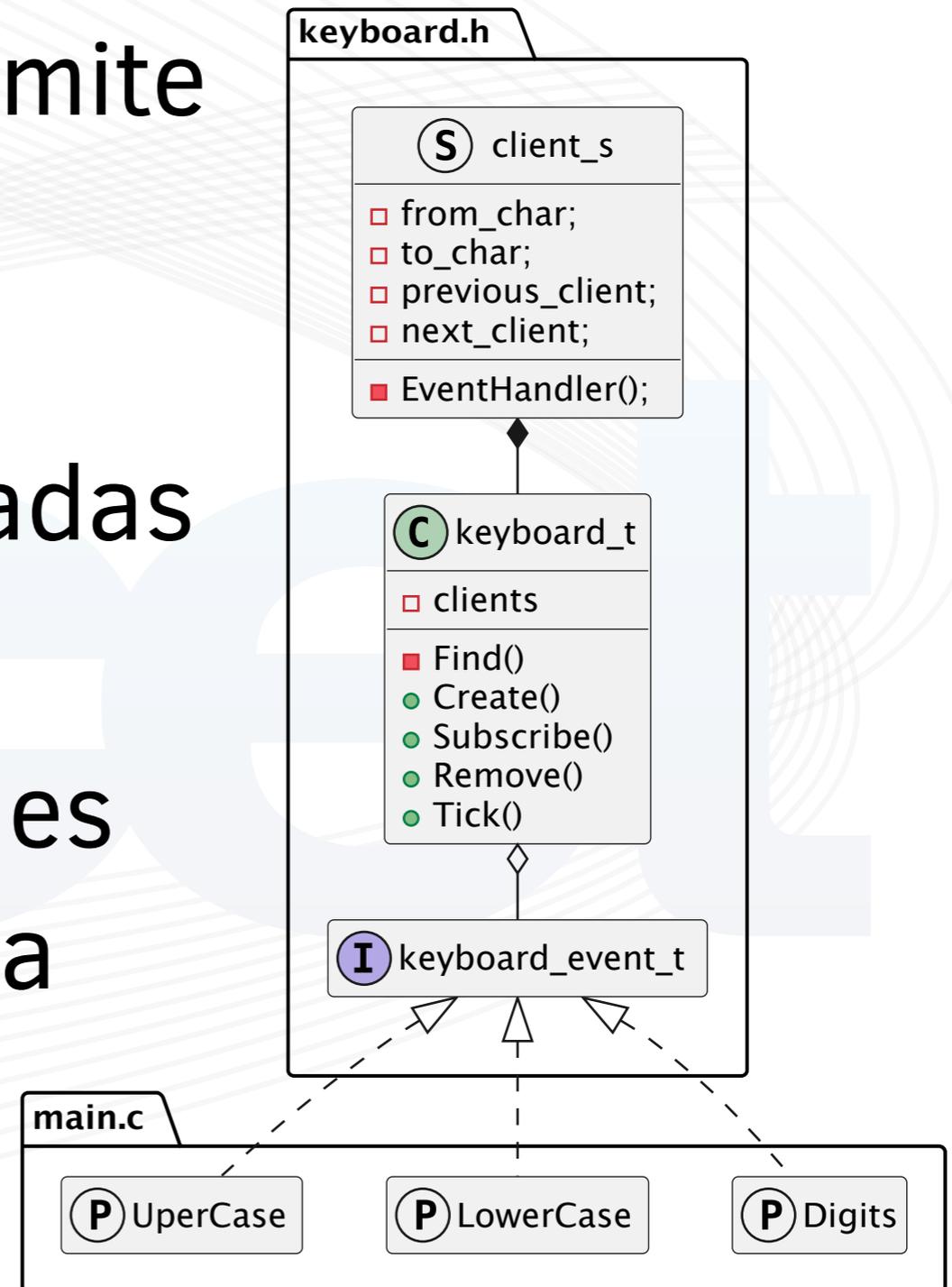


Patron observador

- ▶ Un componente que genera eventos
- ▶ Varios componentes están interesados en esos eventos.
- ▶ El generador implementa un mecanismo de suscripción y cancelación.
- ▶ Eventualmente se pueden incluir filtros para recibir notificaciones selectivas de cierto grupo de eventos.

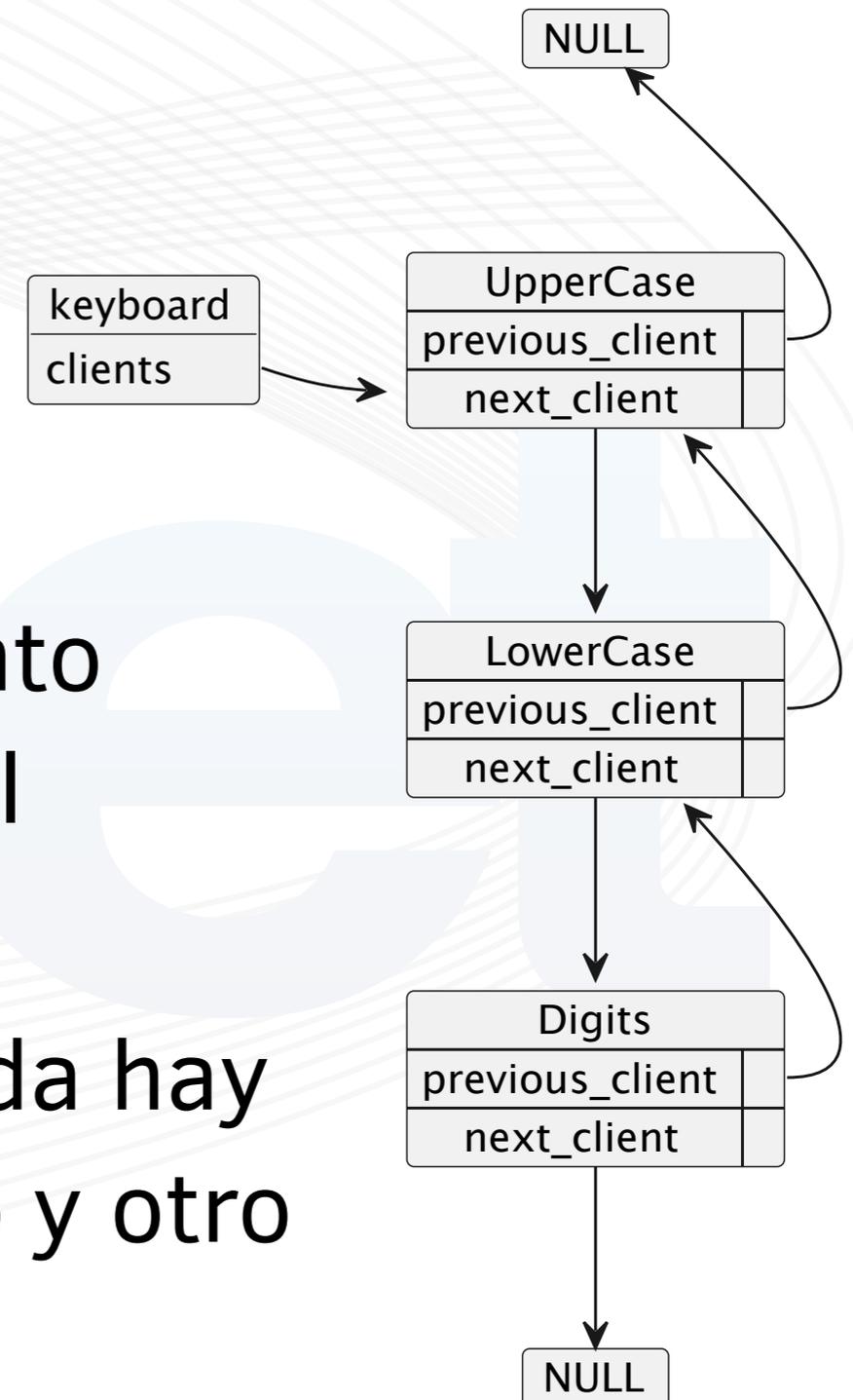
Ejemplo: un gestor de teclado

- ▶ Un gestor de teclado permite a los clientes suscribirse para recibir los eventos generados por determinadas teclas
- ▶ Como la lista de clientes es indefinida se necesita una lista enlazada en lugar de un arreglo



Listas enlazadas

- ▶ Las listas enlazadas son una alternativa a los vectores
- ▶ Permiten variar la cantidad de elementos en forma dinámica
- ▶ En la versión simple cada elemento tiene un puntero al siguiente, y el ultimo elemento apunta a NULL
- ▶ En la versión doblemente enlazada hay un puntero al siguiente elemento y otro al anterior



Patron estrategia

- ▶ Un componente debe completar una misma tarea de diferentes formas
- ▶ El componente define una interfaz para la tarea
- ▶ El componente puede o no implementar un conjunto de implementaciones para la tarea
- ▶ El usuario del componente puede extender el funcionamiento aportando implementaciones independientes

Ejemplo: una calculadora

- ▶ Una calculadora que permite definir nuevas operaciones
- ▶ Esto permite agregar operaciones nuevas a una calculadora existente

