UNIAGRARIA

Fundación Universitaria Agraria de Colombia

LA **U VERDE** DE COLOMBIA

Institución Universitaria Personería Jurídica Nº 2599-86 M.E.N.

Institución de Educación Superior sujeta a inspección y vigilancia por parte del M.E.N.

Bogotá
Calle 170 Nº 54A - 10
Línea de atención
PBX: 601 667 1515

Facatativá
Carrera 2 Nº 4 - 21
Líneas de atención
601 890 07 37 - 601 890 07 32

www.**uniagraria**.edu.co

informes@uniagraria.edu.co

UNIAGRARIA

Educación Virtual

Carlos Andrés Gómez Vasco

Aplicación de POO en Proyectos Simples

UNIAGRARIA

Calculadora con POO

Objetivo:

Desarrollar una aplicación que simule el comportamiento de una calculadora física tradicional.

Características clave:

Introducción secuencial de dígitos para construir números.

Uso de operadores básicos para realizar operaciones.

Visualización en una pantalla pequeña de números y resultados.

Funcionamiento paso a paso según se pulsan los botones (⚠ no interpreta expresiones completas).

Meta del proyecto:

Recrear el funcionamiento real de una calculadora…

¡Tal como la usarías en la vida cotidiana! 💡

Arquitectura del Proyecto — Calculadora

Antes de programar → análisis y diseño

Claves del diseño de software:

- Analizar y diseñar antes de programar
- Usar patrones para resolver problemas comunes

Modelo (Lógica)

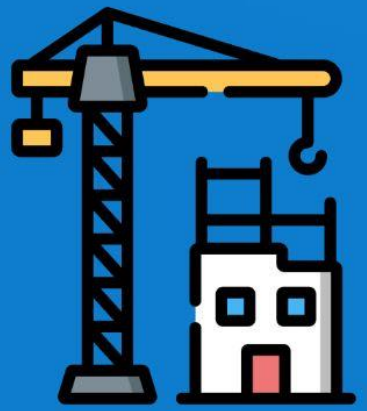
- Mantiene el estado (números y operadores)
- Realiza los cálculos
- **No debe conocer a la interfaz**, para reducir acoplamiento y aumentar cohesión

¿QUÉ SON LOS PATRONES DE DISEÑO?

Son técnicas para resolver problemas recurrentes de diseño de software.

CREACIONALES

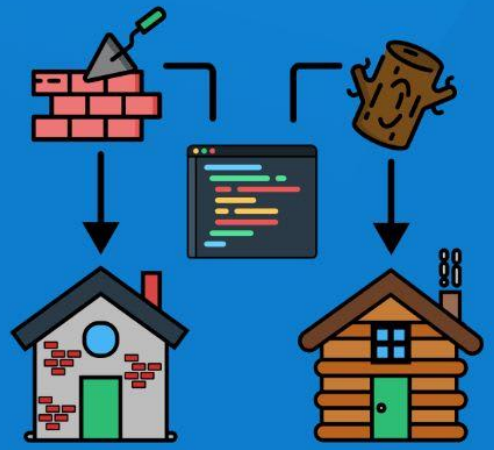
Soluciona la creación de objetos, hace un sistema independiente de cómo sus objetos son creados.



Ej. Singleton, Factory.

ESTRUCTURALES

Describe cómo los objetos se componen para formar estructuras complejas.



Ej. Adapter, Decorator.

COMPORTAMENTALES

Establece soluciones relacionados con el comportamiento de la aplicación respecto a la interacción entre objetos y clases.

Ej. Observer, State.

Si tienes un problema de diseño es muy probable que ya exista un patrón que lo solucione, no reinventes la rueda.

Alejandro Rodriguez
Backend Developer en EDteam

- ErichGamma
- RichardHelm
- RalphJohnson
- JohnVlissides

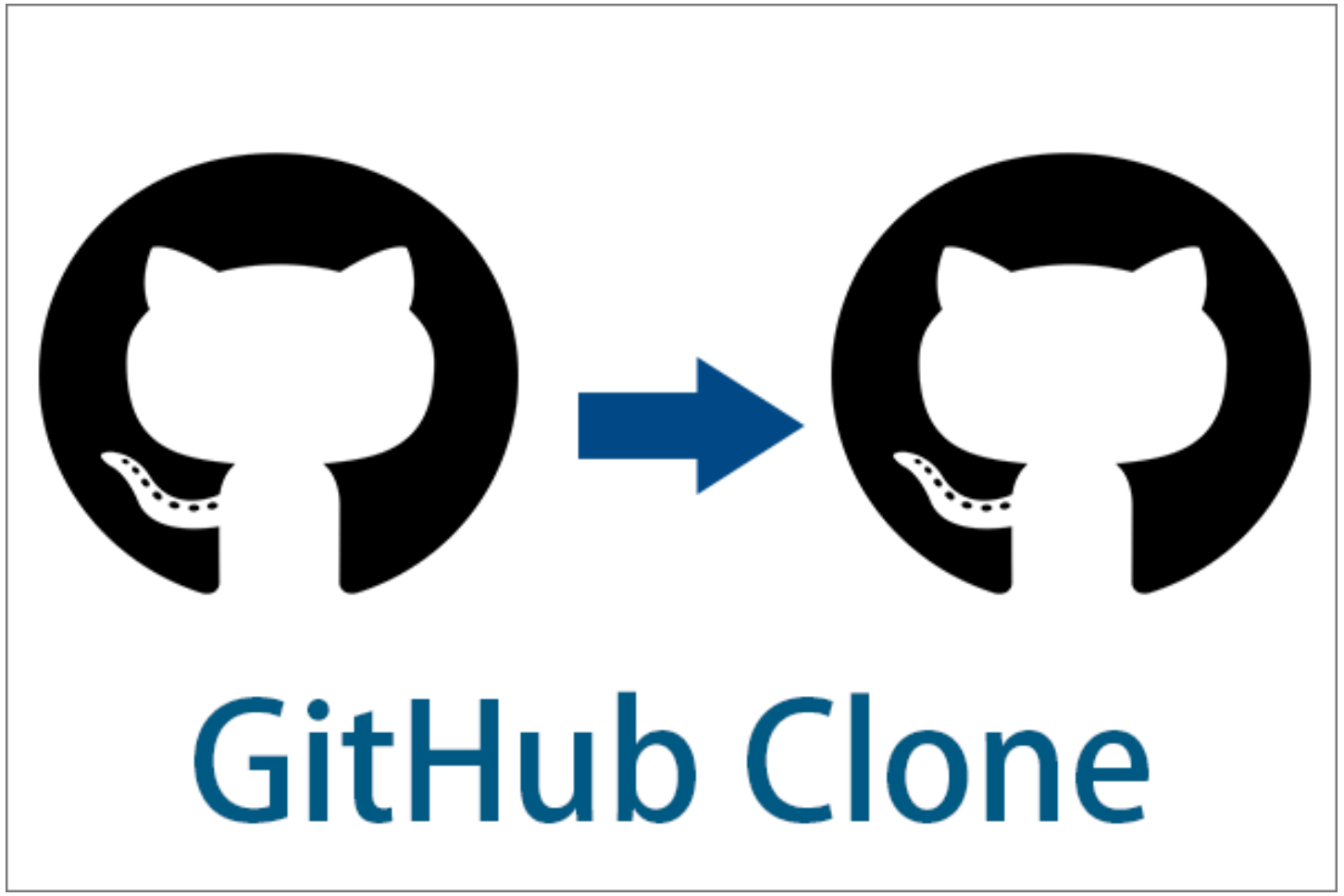
REFACTORING
·GURU·

⚡ El reto

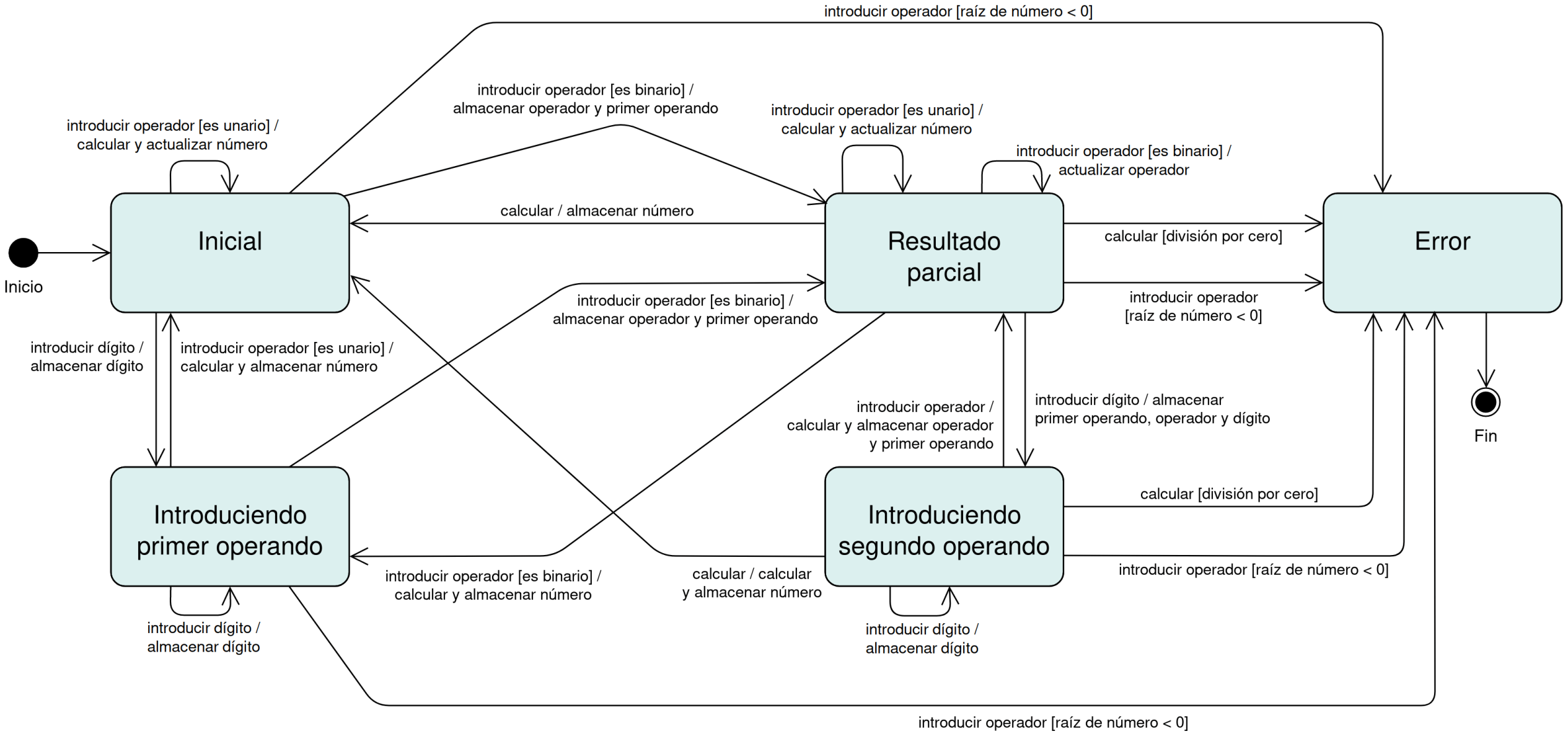
•Muchas acciones posibles: dígitos, operadores, borrar, resultado…
•Acciones en **cualquier orden**
•Calculadora pasa por **estados internos**: inicial, introduciendo operando, error, etc.
•Esto genera **código complejo, lleno de condicionales**

💡 La solución: Patrón Estado

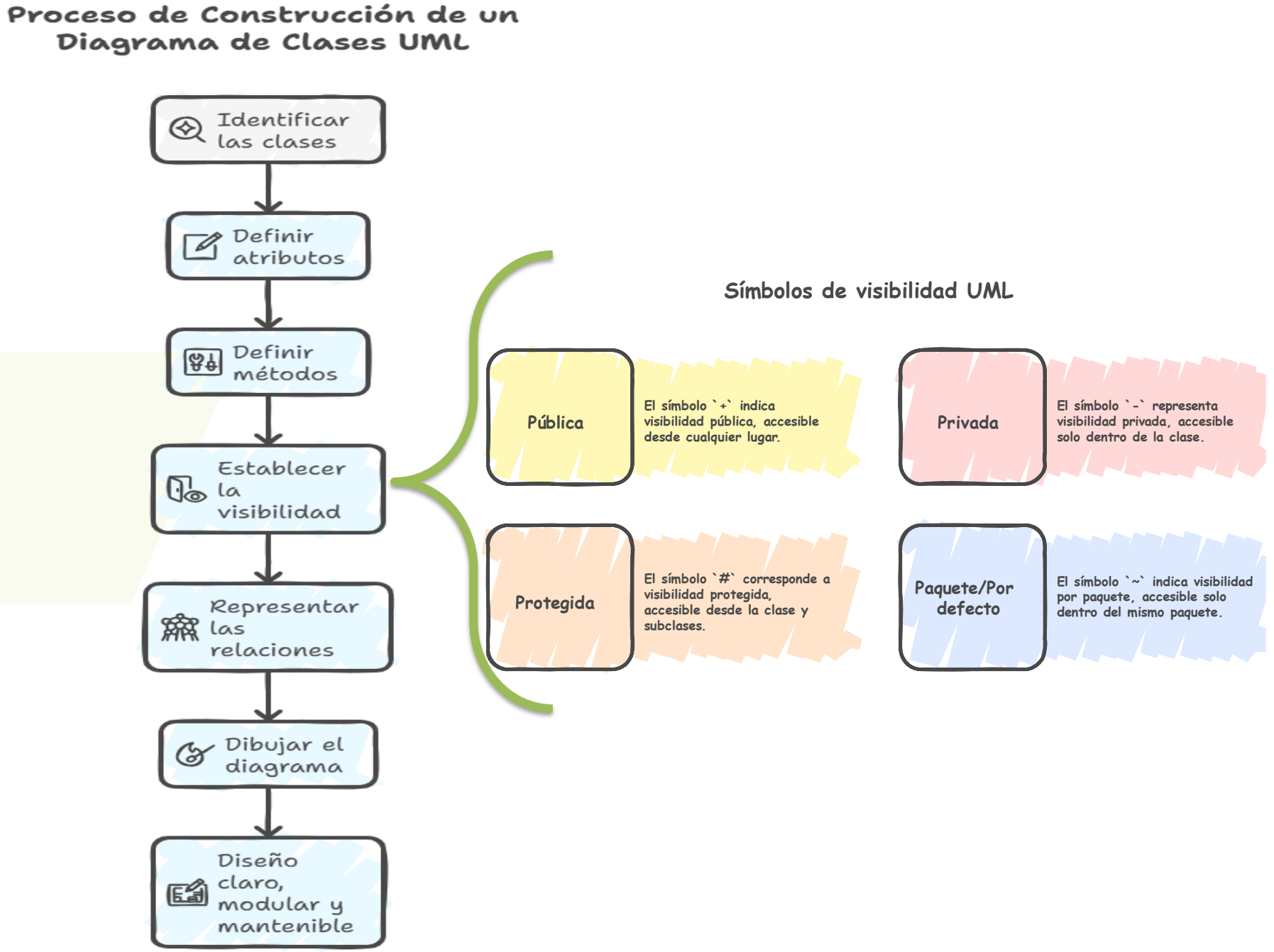
•Cambia el **comportamiento según el estado actual**
•Cada estado define cómo responder a las acciones del usuario
•Simplifica el código → más **legible, mantenible y escalable**
•Permite **añadir nuevos estados sin romper el resto**

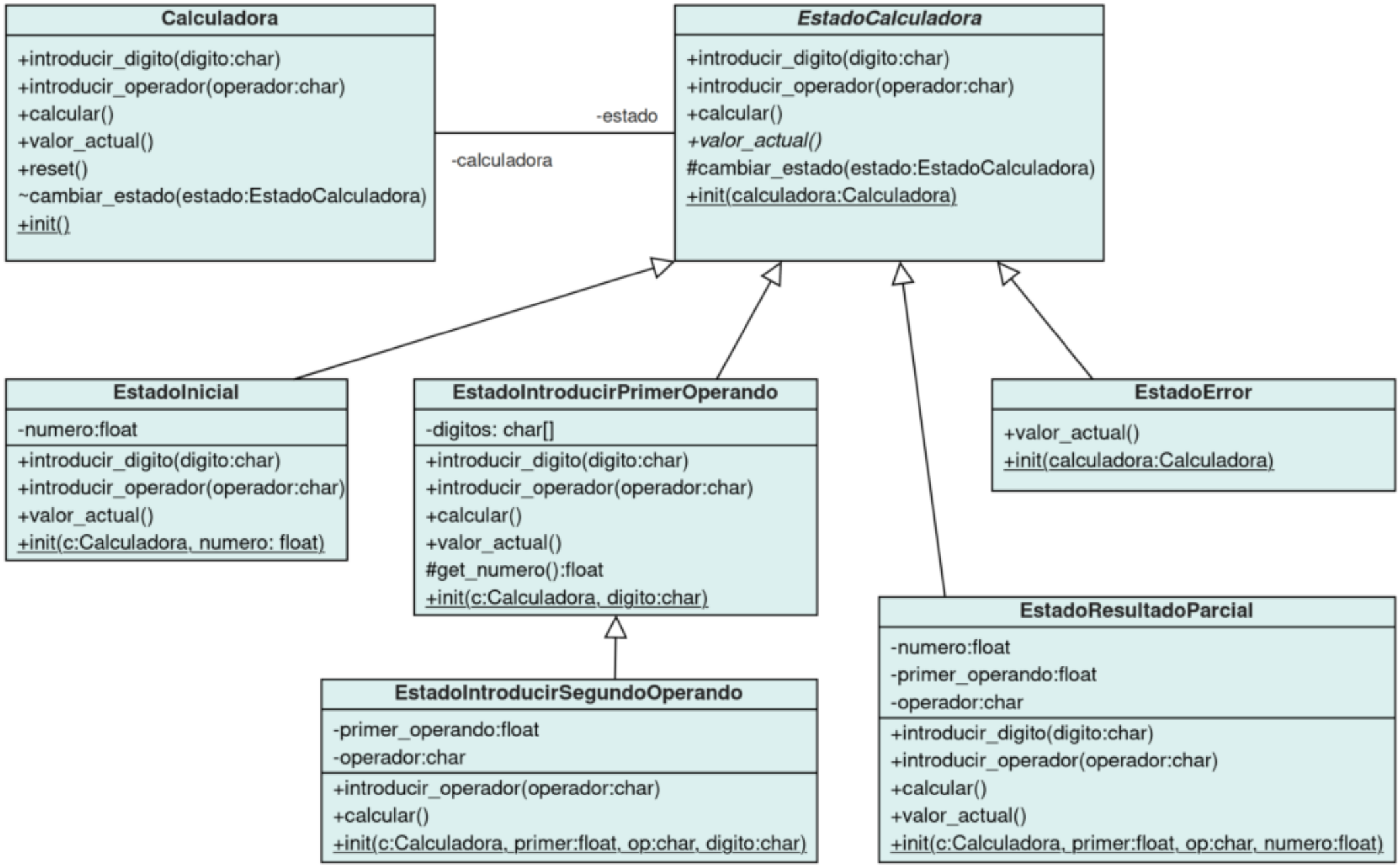


https://github.com/cienciadedatosuniagraria/programacion aplicada.git


introducir operador [raíz de número < 0]
introducir operador [es binario] /
almacenar operador y primer operando
introducir operador [es unario] /
calcular y actualizar número
introducir operador [es unario] /
calcular y actualizar número
introducir operador [es binario] /
actualizar operador
Inicial
calcular / almacenar número
Resultado
parcial
calcular [división por cero]
Error
Inicio
introducir operador [es binario] /
almacenar operador y primer operando
introducir operador
[raíz de número < 0]
introducir dígito /
almacenar dígito
introducir operador [es unario] /
calcular y almacenar número
introducir operador /
calcular y almacenar operador
y primer operando
introducir dígito / almacenar
primer operando, operador y dígito
Fin
Introduciendo
primer operando
Introduciendo
segundo operando
calcular [división por cero]
introducir operador [es binario] /
calcular y almacenar número
calcular / calcular
y almacenar número
introducir operador [raíz de número < 0]
introducir dígito /
almacenar dígito
introducir dígito /
almacenar dígito
introducir operador [raíz de número < 0]

Identificación y construcción de clases.




Calculadora
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
+reset()
~cambiar_estado(estado:EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
#cambiar_estado(estado:EstadoCalculadora)
+init(calculadora:Calculadora)
EstadoInicial
-numero:float
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+valor_actual()
+init(c:Calculadora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
#get_numero():float
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_operador(operador:char)
+calcular()
+init(c:Calculadora, primer:float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

calculadora_modelo.py.


```python
from math import sqrt
from abc import ABC, abstractmethod

suma = lambda x, y: x + y
resta = lambda x, y: x - y
multiplicacion = lambda x, y: x * y
division = lambda x, y: x / y
cambio_de_signo = lambda x: -x
raiz = lambda x: sqrt(x)

binarios = {
    '+': suma,
    '-': resta,
    '*': multiplicacion,
    '/': division
}

unarios = {
    's': cambio_de_signo,
    'r': raiz
}
```

Clase abstracta de la calculadora

Se define una clase abstracta que indica qué operaciones debe implementar la calculadora (binarias y unarias).

Las clases concretas usan diccionarios de operadores para ejecutar los cálculos.

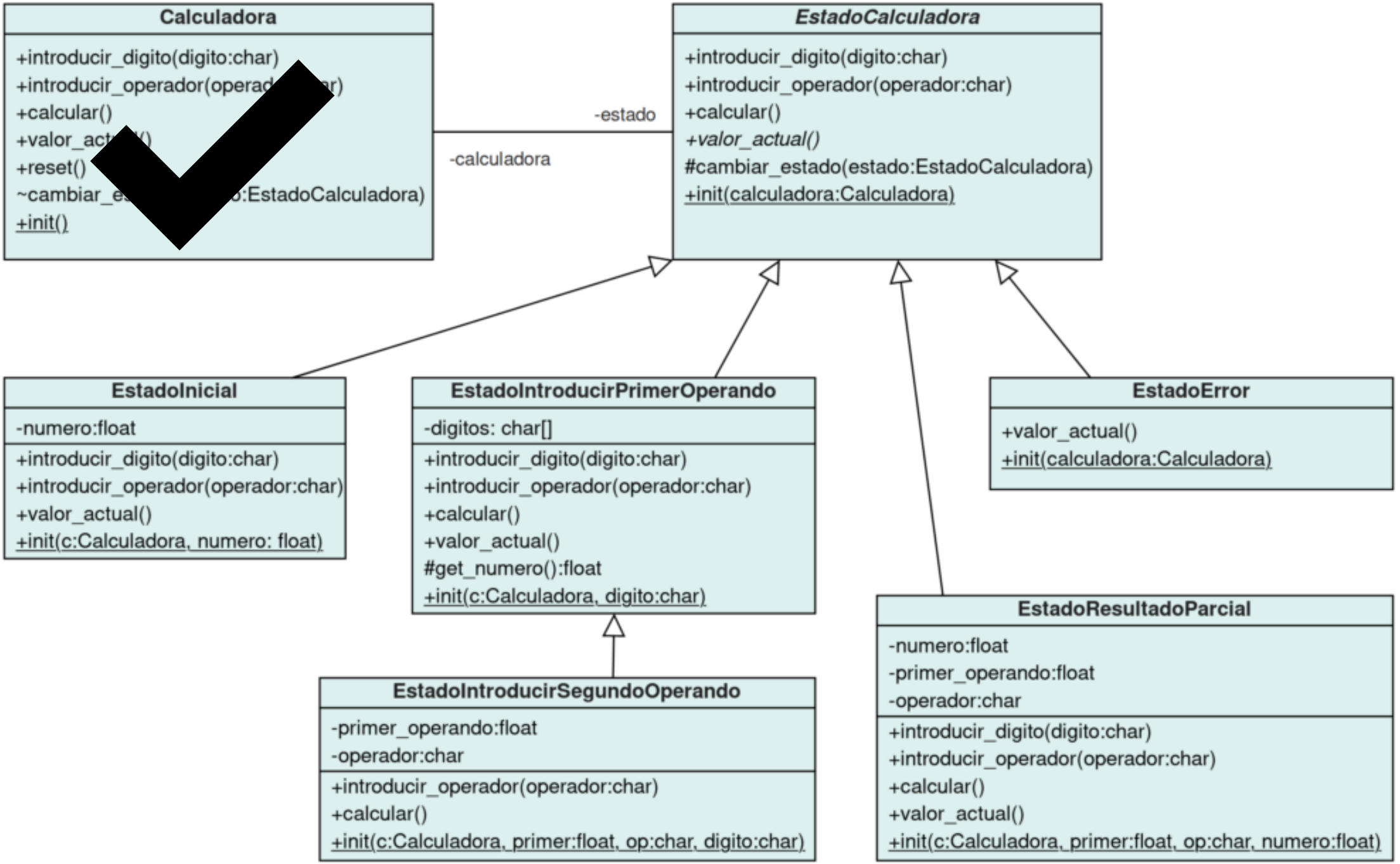
Definición de operaciones matemáticas

Se definen las operaciones básicas (suma, resta, multiplicación, división y raíz cuadrada).

Estas se organizan en dos diccionarios:

- **Operadores binarios**: usan dos valores (a + b).
- **Operadores unarios**: usan un solo valor (√a).

Cada símbolo se asocia a una función para facilitar su uso en la calculadora.


Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad r)
+calcular()
+valor_act ()
+reset()
~cambiar_e :EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
#cambiar_estado(estado:EstadoCalculadora)
+init(calculadora:Calculadora)
EstadoInicial
-numero:float
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+valor_actual()
+init(c:Calculadora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
#get_numero():float
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_operador(operador:char)
+calcular()
+init(c:Calculadora, primer:float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

clase Calculadora:


```python
class Calculadora:

    def __init__(self):
        self.reset()

    def reset(self):
        self._estado = EstadoInicial(self, 0)

    def introducir_digito(self, digito):
        self._estado.introducir_digito(digito)

    def introducir_operador(self, operador):
        try:
            self._estado.introducir_operador(operador)
        except:
            self.cambiar_estado(EstadoError(self))

    def calcular(self):
        try:
            self._estado.calcular()
        except:
            self.cambiar_estado(EstadoError(self))

    def valor_actual(self):
        return self._estado.valor_actual()

    def cambiar_estado(self, estado):
        self._estado = estado
```

Clase Calculadora y el Patrón Estado

La clase Calculadora parece vacía, porque delegamos los comportamientos en los estados.

Atributos:

•Solo tiene uno: _estado, que guarda la referencia al estado actual.

•Los demás datos se almacenan en los objetos estado.

Inicialización:

•Al crear un objeto Calculadora, se invoca reset(), que coloca la calculadora en el EstadoInicial.

Cambiar de estado:

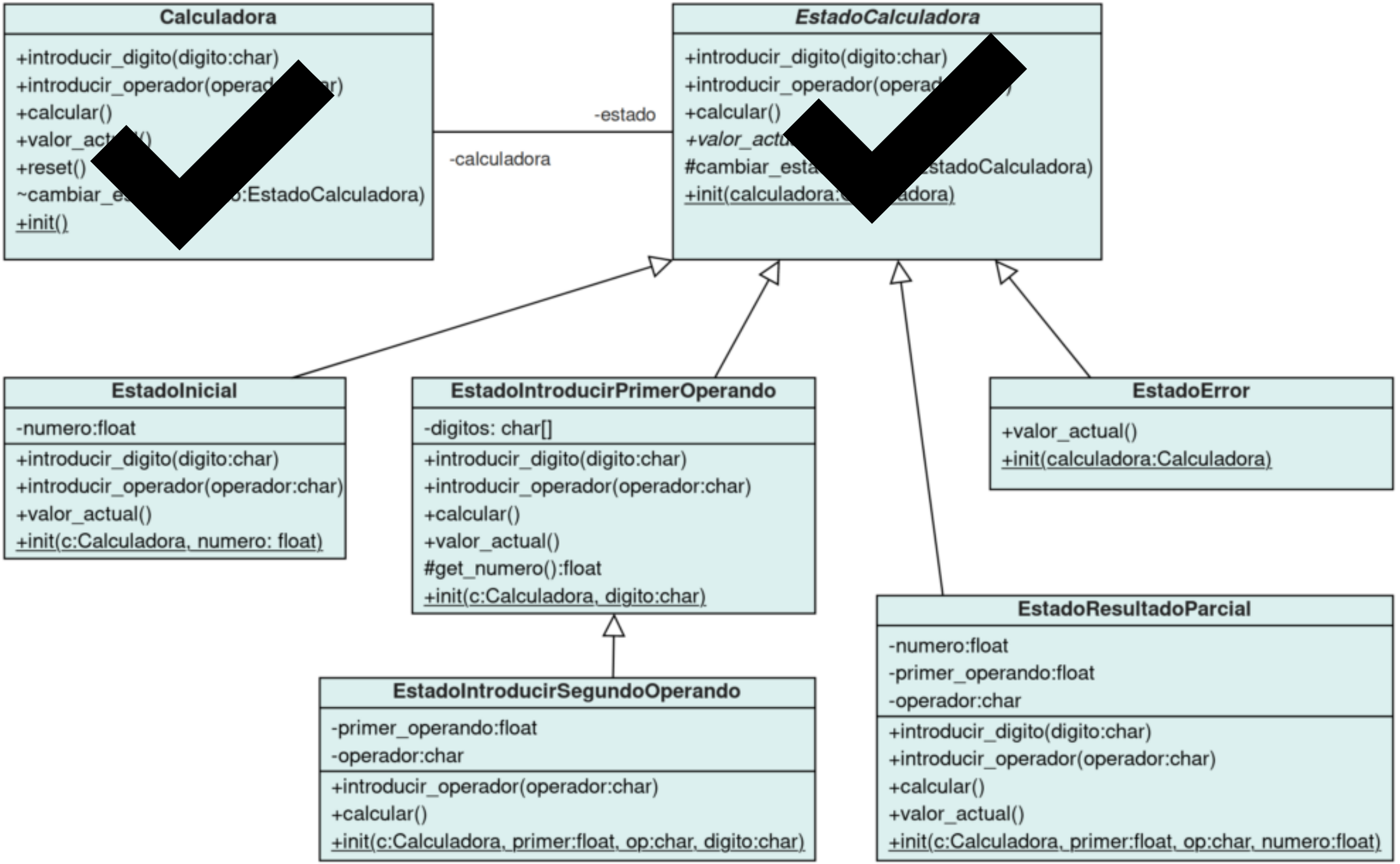
•El método cambiar_estado() permite que los propios estados realicen las transiciones cuando sea necesario.

Delegación de operaciones:

•Todas las operaciones se ejecutan llamando al método del estado actual.

•Algunas funciones usan try/except para capturar errores (operador inválido, división entre cero, raíz negativa, etc.).

•Ante un error, la calculadora pasa automáticamente al EstadoError.

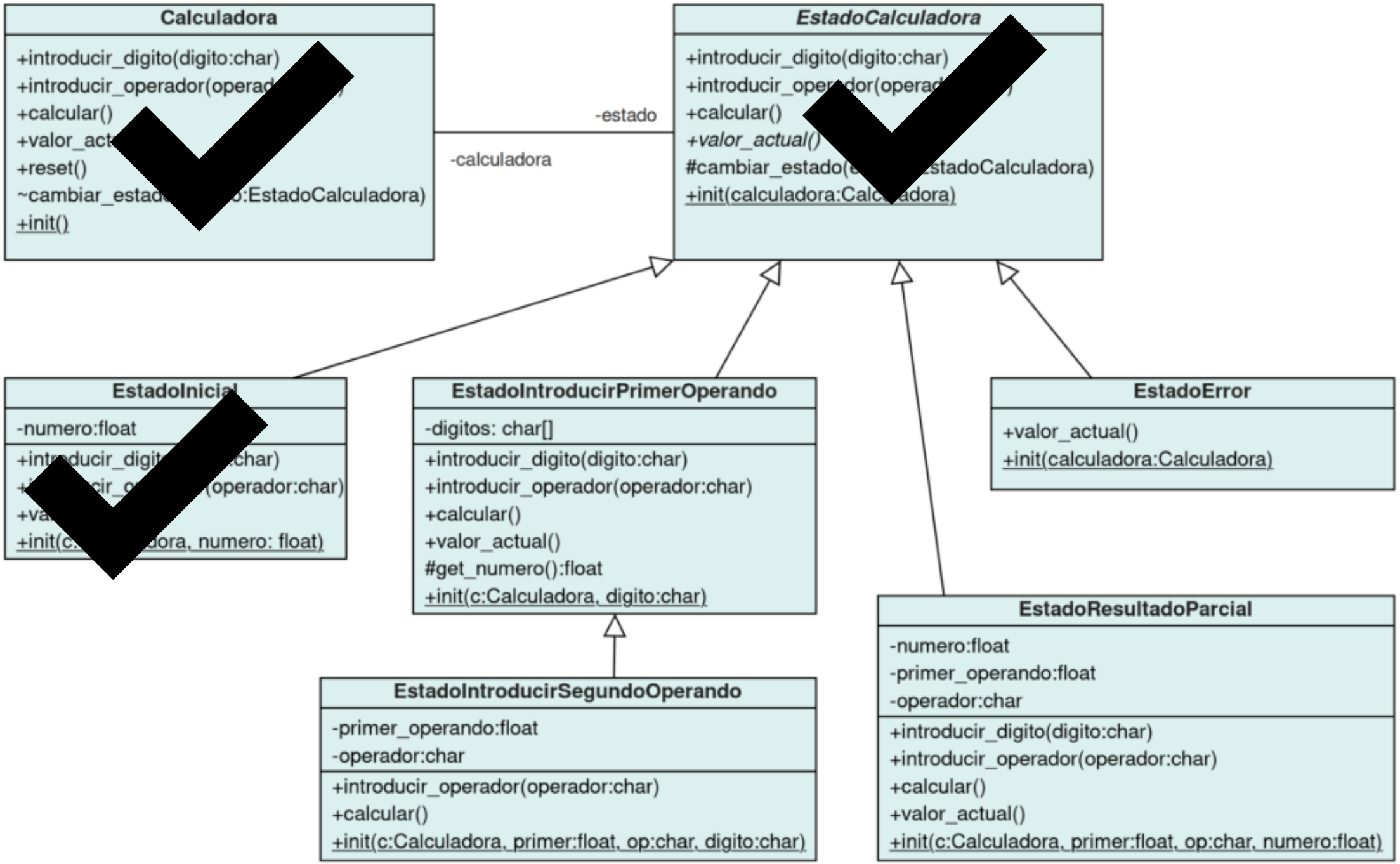

Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad r)
+calcular()
+valor_act ()
+reset()
~cambiar_e o:EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_operador(operad)
+calcular()
+valor_actu
#cambiar_esta stadoCalculadora)
+init(calculadora:C adora)
EstadoInicial
-numero:float
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+valor_actual()
+init(c:Calculadora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
#get_numero():float
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_operador(operador:char)
+calcular()
+init(c:Calculadora, primer:float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

clase padre abstracta EstadoCalculadora


```python
class EstadoCalculadora(ABC):

    def __init__(self, calculadora):
        self._calculadora = calculadora

    def introducir_digito(self, digito):
        pass

    def introducir_operador(self, operador):
        pass

    def calcular(self):
        pass

    @abstractmethod
    def valor_actual(self):
        pass

    def cambiar_estado(self, nuevo_estado):
        self._calculadora.cambiar_estado(nuevo_estado)
```

Clase EstadoCalculadora (Abstracta)

- **Herencia:**
  - Extiende de ABC (librería abc) → no se puede instanciar directamente.
- **Inicialización (__init__):**
  - Guarda en _calculadora la referencia al objeto Calculadora.
  - Esto permite que los estados **cambien el estado actual** de la calculadora.
- **Métodos:**
  - valor_actual → método **abstracto**, obliga a cada subclase a implementar cómo mostrar el valor en pantalla.
  - introducir_digito, introducir_operador, calcular → no hacen nada en esta clase.
    - Se redefinen en los **estados concretos**, ya que el comportamiento varía.
  - El resto de operaciones tienen comportamiento por defecto = "**no hacer nada**".
    - Ejemplo: en el **EstadoInicial**, si presionamos =, no ocurre nada.

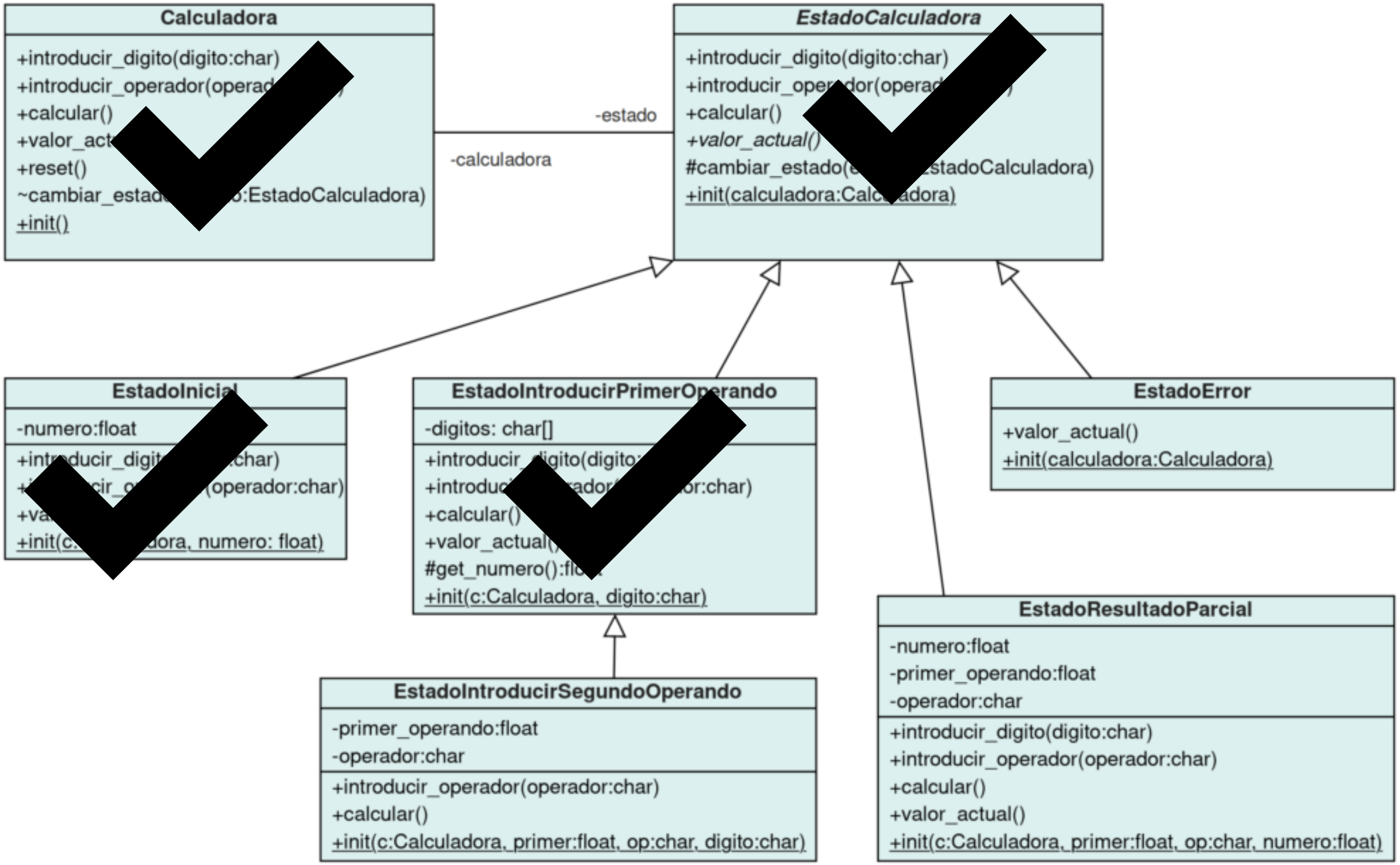

Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad
+calcular()
+valor_act
+reset()
~cambiar_estad
:EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_oper dor(operad
+calcular()
+valor_actual()
#cambiar_estado(e
stadoCalculadora)
+init(calculadora:Calc adora)
EstadoInicial
-numero:float
+introducir_digit
char)
+
cir_o
(operador:char)
+va
+init(c:
dora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
#get_numero():float
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_operador(operador:char)
+calcular()
+init(c:Calculadora, primer:float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

clase EstadoInicial.

```python
class EstadoInicial(EstadoCalculadora):

    def __init__(self, calculadora, numero):
        super().__init__(calculadora)
        self._numero = numero

    def introducir_digito(self, digito):
        self.cambiar_estado(EstadoIntroducirPrimerOperando(self._calculadora, digito))

    def introducir_operador(self, operador):
        if operador in unarios:
            self._numero = unarios[operador](self._numero)
        elif operador in binarios:
            self.cambiar_estado(EstadoResultadoParcial(self._calculadora, self._numero, self._numero, operador))
        else:
            raise ValueError(f'No existe el operador "{operador}".')

    def valor_actual(self):
        return self._numero
```

- Inicialización (__init__):
  - Recibe la calculadora y un número inicial (mostrado en pantalla).
  - Guarda el número en un atributo.
  - Llama al constructor de la clase padre con super().__init__(calculadora).

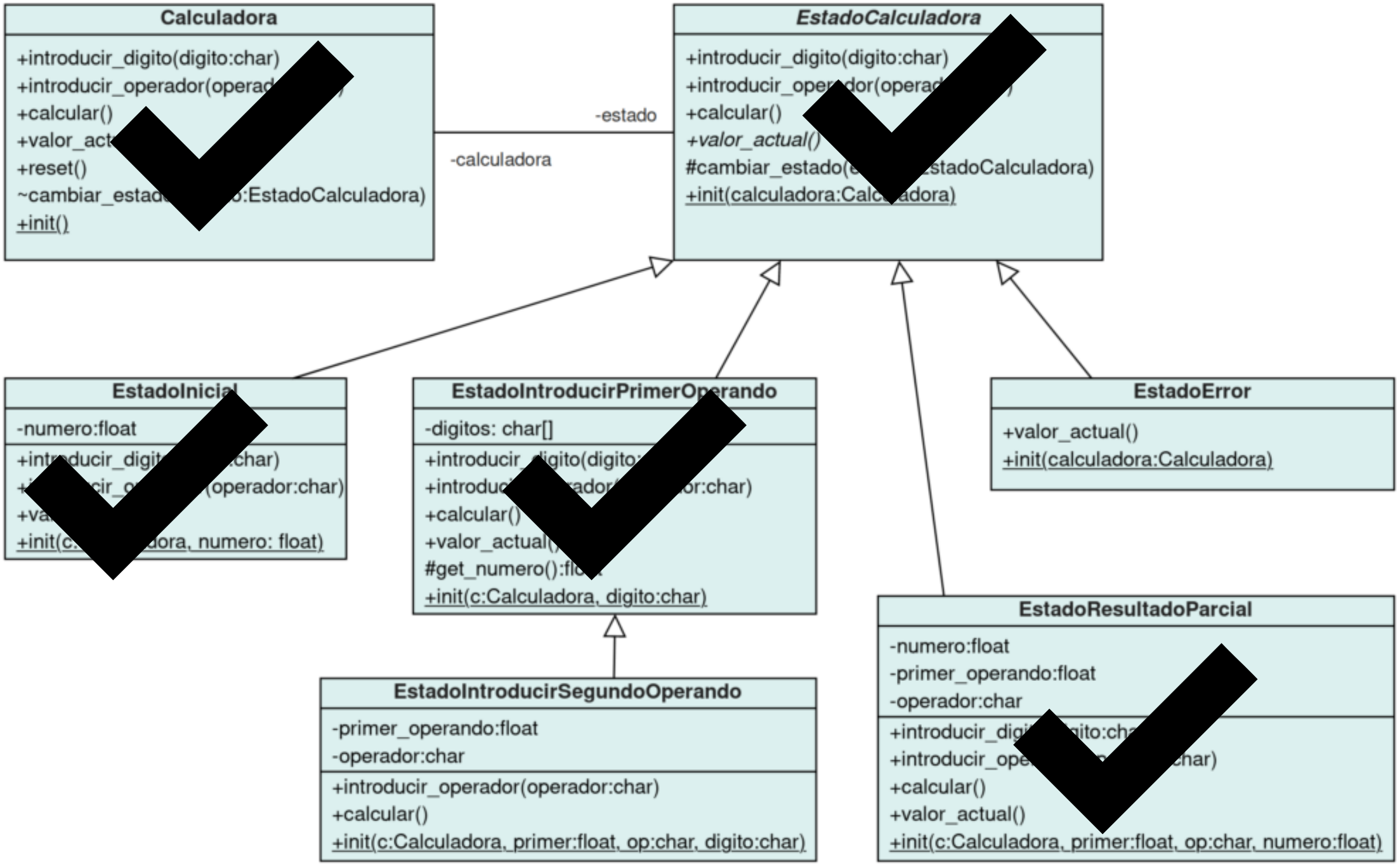
- Operaciones:
  - valor_actual → devuelve el número almacenado.
  - introducir_digito → cambia al estado EstadoIntroducirPrimerOperando (según el diagrama).
  - introducir_operador:
    - Operador unario → se calcula directamente usando el diccionario de operadores unarios. El estado no cambia (permanece en EstadoInicial).
    - Operador binario → se cambia al estado EstadoResultadoParcial, guardando:
      - primer operando,
      - número a mostrar,
      - operador.


Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad
+calcular()
+valor_act
+reset()
~cambiar_estad
:EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_oper dor(operad
+calcular()
+valor_actual()
#cambiar_estado(e stadoCalculadora)
+init(calculadora:Calc adora)
EstadoInicial
-numero:float
+introducir_digit char)
+ cir_o (operador:char)
+va
+init(c dora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:
+introduc rador or:char)
+calcular()
+valor_actual(
#get_numero():fl
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_operador(operador:char)
+calcular()
+init(c:Calculadora, primer:float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digito(digito:char)
+introducir_operador(operador:char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

```python
class EstadoIntroducirPrimerOperando(EstadoCalculadora):

    def __init__(self, calculadora, digito):
        super().__init__(calculadora)
        self._digitos = []
        self.introducir_digito(digito)

    def introducir_digito(self, digito):
        digito = str(digito)
        if digito == '.':
            if '.' not in self._digitos:
                if len(self._digitos) == 0:
                    self._digitos.append('0')
                self._digitos.append(digito)
        else:
            if len(self._digitos) == 1 and self._digitos[0] == '0':
                self._digitos.clear()
            self._digitos.append(digito)

    def introducir_operador(self, operador):
        if operador in unarios:
            numero = self._get_numero()
            numero = unarios[operador](numero)
            self.cambiar_estado(EstadoInicial(self._calculadora, numero))
        elif operador in binarios:
            numero = self._get_numero()
            self.cambiar_estado(EstadoResultadoParcial(self._calculadora, n
        else:
            raise ValueError(f'No existe el operador "{operador}".')

    def _get_numero(self):
        return float(''.join(self._digitos))

    def valor_actual(self):
        return ''.join(self._digitos)
```

EstadoIntroducirPrimerOperando:

- **Inicialización (__init__):**
  - Los dígitos introducidos por el usuario se almacenan en una **lista de caracteres** (ej. ['1', '2', '.', '3']).
  - Esto facilita validaciones y la construcción del número completo.
  - El método protegido _get_numero() convierte la lista de dígitos en un **número real** cuando se necesita.
- **Introducción de dígitos:**
  - Se controla que solo pueda existir **un punto decimal**.
  - Si hay un **cero inicial**, introducir un nuevo cero no lo duplica.
  - Si se introduce un punto con un cero inicial, se transforma en "0." en lugar de reemplazarlo.
- **Introducción de operadores:**
  - **Operador unario:**
    - Se construye el número a partir de la lista.
    - Se aplica la operación unaria.
    - Transición al **EstadoInicial**.
  - **Operador binario:**
    - Se construye el número a partir de la lista.
    - Se transita a **EstadoResultadoParcial**, guardando:
      - Primer operando.
      - Número en pantalla.
      - Operador.


Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad
+calcular()
+valor_act
+reset()
~cambiar_estad
:EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_oper dor(operad
+calcular()
+valor_actual()
#cambiar_estado(e stadoCalculadora)
+init(calculadora:Calc adora)
EstadoInicial
-numero:float
+introducir_digit char)
+ cir_o (operador:char)
+va
+init(c dora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:
+introduc rador or:char)
+calcular()
+valor_actual(
#get_numero():fl
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_operador(operador:char)
+calcular()
+init(c:Calculadora, primer:float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digi ito:cha
+introducir_ope char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

Estado de Resultado Parcial

Contexto:

•Representa el momento en el que se ha introducido un primer operando y un operador binario, pero aún no hay segundo operando.

•Operaciones destacadas:

•calcular:

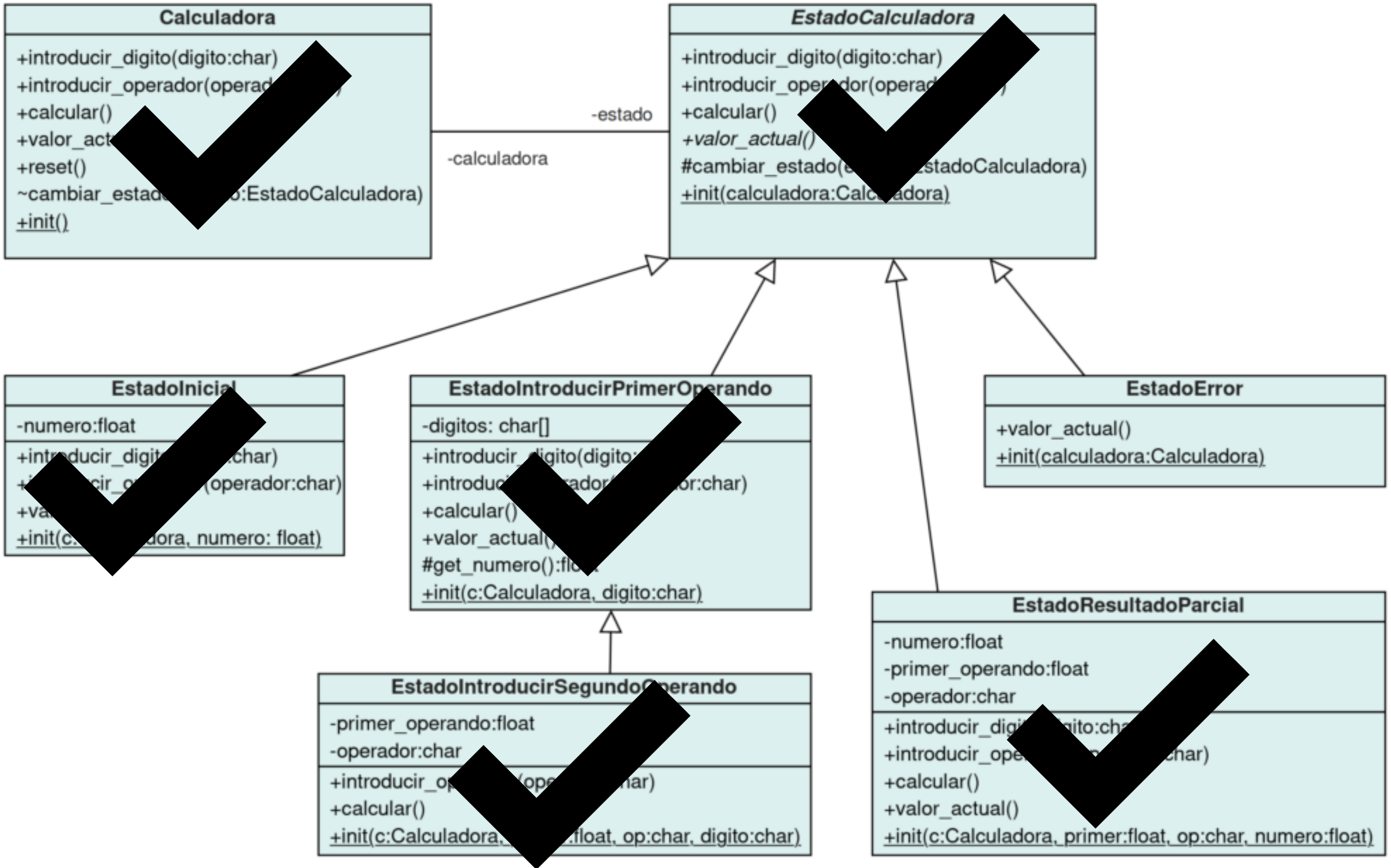
- •Realiza la operación binaria usando:
  - •El primer operando.
  - •El número actualmente mostrado en pantalla.
- •Devuelve el resultado.
- •Transita hacia el EstadoInicial, mostrando el resultado como nuevo valor.

```python
class EstadoResultadoParcial(EstadoCalculadora):

    def __init__(self, calculadora, primer_operando, numero, operador):
        super().__init__(calculadora)
        self._primer_operando = primer_operando
        self._numero = numero
        self._operador = operador

    def introducir_digito(self, digito):
        self.cambiar_estado(EstadoIntroducirSegundoOperando(self._calculadora, self._prim

    def introducir_operador(self, operador):
        if operador in unarios:
            self._numero = unarios[operador](self._numero)
        elif operador in binarios:
            self._operador = operador
        else:
            raise ValueError(f'No existe el operador "{operador}".')

    def calcular(self):
        resultado = binarios[self._operador](self._primer_operando, self._numero)
        self.cambiar_estado(EstadoInicial(self._calculadora, resultado))

    def valor_actual(self):
        return self._numero
```

•Otros aspectos:

•La lógica es similar a los estados previos.

•La diferencia clave es que aquí la calculadora ya tiene información previa (primer operando + operador) y decide qué hacer si se presiona = antes de introducir un segundo número.


Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad
+calcular()
+valor_act
+reset()
~cambiar_estad :EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_oper dor(operad
+calcular()
+valor_actual()
#cambiar_estado(stadoCalculadora)
+init(calculadora:Calc adora)
EstadoInicial
-numero:float
+introducir_digit char)
+ cir_o (operador:char)
+va
+init(c dora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:
+introduc rador or:char)
+calcular()
+valor_actual
#get_numero():fl
+init(c:Calculadora, digito:char)
EstadoError
+valor_actual()
+init(calculadora:Calculadora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_o (ope har)
+calcular()
+init(c:Calculadora, float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digi ito:cha
+introducir_ope char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

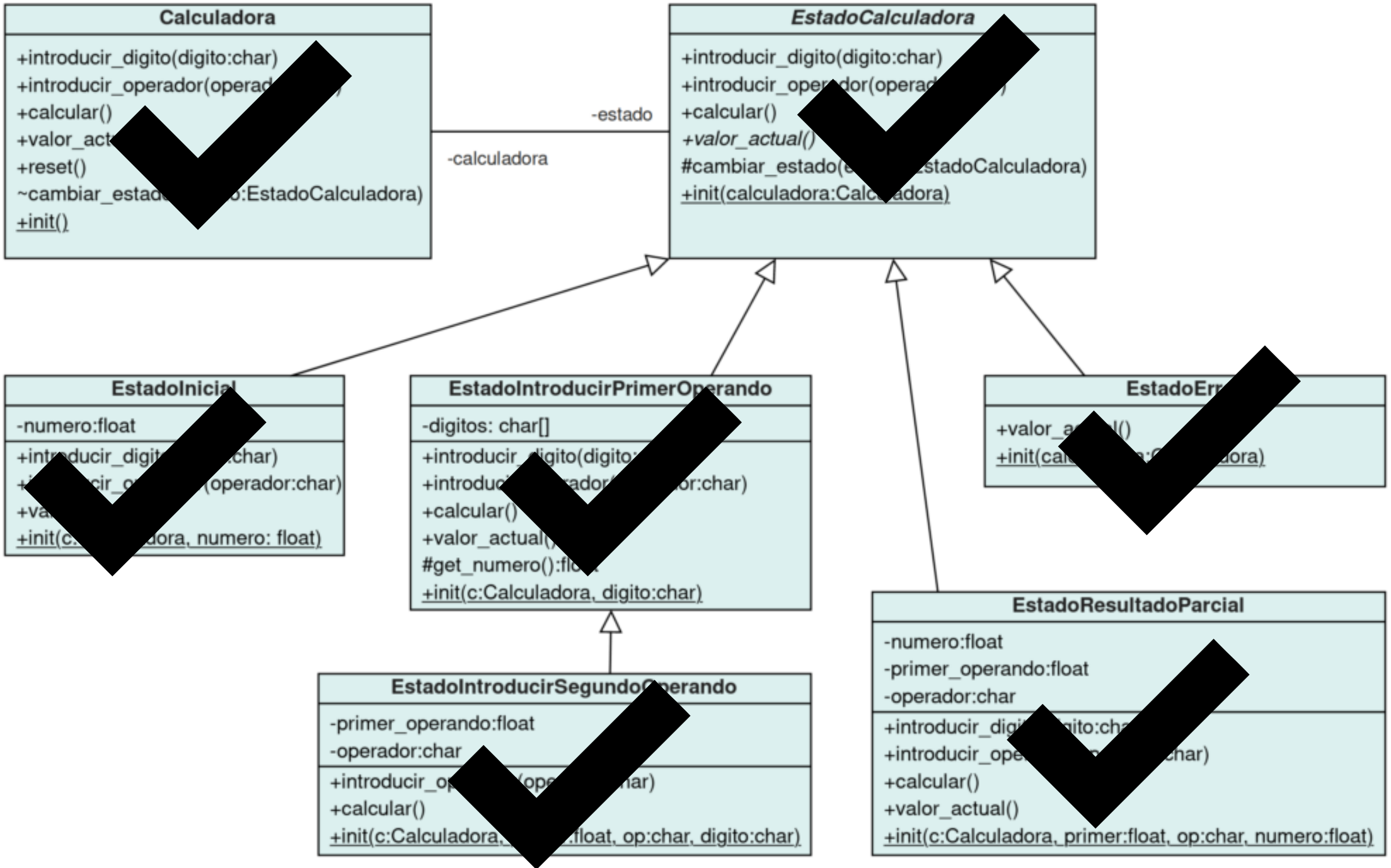
EstadoIntroduciendoSegundoOperando:

•**Herencia:**
•Extiende de EstadoIntroducirPrimerOperando.
•Comparte la lógica de gestión de dígitos y atributos.

•**Diferencias principales:**
•Solo redefine: introducir_operador y calcular
•Todo lo relacionado con la **entrada de dígitos** se reutiliza de la clase padre.

•**Comportamiento:**
•introducir_operador:
    •Permite encadenar operaciones (usando el número ya introducido como nuevo operando).
    •Puede transitar a EstadoResultadoParcial.
•calcular:
    •Ejecuta la operación binaria con:
        •Primer operando.
        •Segundo operando (obtenido de la lista de dígitos).
    •Devuelve el resultado y transita a EstadoInicial.

```python
class EstadoIntroducirSegundoOperando(EstadoIntroducirPrimerOperando):

    def __init__(self, calculadora, primer_operando, operador, digito):
        super().__init__(calculadora, digito)
        self._primer_operando = primer_operando
        self._operador = operador

    def introducir_operador(self, operador):
        if operador in unarios:
            resultado = unarios[operador](self._get_numero())
            self.cambiar_estado(EstadoResultadoParcial(self._calculadora, self._primer_ope
        elif operador in binarios:
            resultado = binarios[self._operador](self._primer_operando, self._get_numero()
            self.cambiar_estado(EstadoResultadoParcial(self._calculadora, resultado, resul
        else:
            raise ValueError(f'No existe el operador "{operador}".')

    def calcular(self):
        resultado = binarios[self._operador](self._primer_operando, self._get_numero())
        self.cambiar_estado(EstadoInicial(self._calculadora, resultado))
```


Calculadora
+introducir_digito(digito:char)
+introducir_operador(operad
+calcular()
+valor_act
+reset()
~cambiar_estad :EstadoCalculadora)
+init()
-estado
-calculadora
EstadoCalculadora
+introducir_digito(digito:char)
+introducir_oper dor(operad
+calcular()
+valor_actual()
#cambiar_estado(stadoCalculadora)
+init(calculadora:Calc adora)
EstadoInicial
-numero:float
+introducir_digit char)
+ cir_o (operador:char)
+va
+init(c dora, numero: float)
EstadoIntroducirPrimerOperando
-digitos: char[]
+introducir_digito(digito:
+introduc ador or:char)
+calcular()
+valor_actual()
#get_numero():flo
+init(c:Calculadora, digito:char)
EstadoErr
+valor_a ()
+init(cal dora)
EstadoIntroducirSegundoOperando
-primer_operando:float
-operador:char
+introducir_op (ope har)
+calcular()
+init(c:Calculadora, float, op:char, digito:char)
EstadoResultadoParcial
-numero:float
-primer_operando:float
-operador:char
+introducir_digi ito:cha
+introducir_ope char)
+calcular()
+valor_actual()
+init(c:Calculadora, primer:float, op:char, numero:float)

Estado de Error:

- **La más sencilla de todas**:
  - No redefine los métodos de la clase padre → hereda los comportamientos por defecto (no hacer nada).
  - Esto es intencional: en estado de error la calculadora **no debe responder** a nuevas entradas.
- **Obligación:**
  - Implementar el método valor_actual.
  - Devuelve un **mensaje de error** para mostrar en pantalla (ej. "ERROR").

```
class EstadoError(EstadoCalculadora):

    def __init__(self, calculadora):
        super().__init__(calculadora)

    def valor_actual(self):
        return '- Error -'
```

Ventajas del patrón Estado:

- Permite **repartir responsabilidades** entre varias clases.
- Evita un código lleno de **condicionales anidados**.
- Diseño **modular, extensible y mantenible**.

Test de la Calculadora <-> En terminal de Python

```
from calculadora import Calculadora

# Creamos una calculadora
c = Calculadora()

print("=== TEST 1: -34 + √4 ===")
c.introducir_digito("-")
c.introducir_digito("3")
c.introducir_digito("4")
print("Pantalla:", c.valor_actual())   # -34

c.introducir_operador("+")
print("Pantalla:", c.valor_actual())   # -34

c.introducir_operador("√")
c.introducir_digito("4")
print("Pantalla:", c.valor_actual())   # 2.0

c.calcular()
print("Resultado final:", c.valor_actual())

print("\n=== TEST 2: 5 + 7 ===")
c = Calculadora()
c.introducir_digito("5")
c.introducir_operador("+")
c.introducir_digito("7")
c.calcular()
print("Resultado final:", c.valor_actual())   # 12.0

print("\n=== TEST 3: 8 / 0 ===")
c = Calculadora()
c.introducir_digito("8")
c.introducir_operador("/")
c.introducir_digito("0")
c.calcular()
print("Resultado final:", c.valor_actual())
```

GRACIAS