

1. Inicia el recorrido del código
2. Llamada a la Función **crearCuentaBancaria**: Llama a la función **crearCuentaBancaria** con el argumento **1000**, que es el saldo inicial de la cuenta bancaria.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

The screenshot shows the Python Tutor interface for JavaScript (ES6). The code defines a function `crearCuentaBancaria(saldoInicial)` that returns an object with methods `consultarSaldo`, `realizarDeposito`, and `realizarRetiro`. It also includes a test example that creates a bank account with an initial balance of 1000, deposits 500, and withdraws 200. The execution is at Step 1 of 34, where the variable `miCuenta` is assigned the return value of `crearCuentaBancaria(1000)`. The Frames panel shows the Global frame with `crearCuentaBancaria` and `miCuenta`. The Objects panel shows the function object for `crearCuentaBancaria` with its internal state and methods.

3. Declaración de la Variable **saldo**:

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

The screenshot shows the Python Tutor interface for JavaScript (ES6) at Step 2 of 34. The code is at line 4, where the variable `saldo` is declared and initialized with the value of `saldoInicial` (1000). The Frames panel shows the Global frame with `crearCuentaBancaria` and `miCuenta`. The Objects panel shows the function object for `crearCuentaBancaria` with its internal state, including the `saldo` variable. The console output shows the log message: "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."

4. La línea 24 inicia el retorno de un objeto literal que contiene métodos públicos para interactuar con la cuenta bancaria. Este objeto permite a los usuarios consultar el saldo, realizar depósitos y retiros, mientras mantiene encapsulados los métodos privados **depositar** y **retirar** y la variable **saldo**.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

The image shows the Python Tutor interface with JavaScript (ES6) code. The code defines a function `crearCuentaBancaria(saldoInicial)` that returns an object with public methods `consultarSaldo`, `realizarDeposito`, and `realizarRetiro`. The object literal on line 24 is highlighted with a red arrow. The Python Tutor interface shows the execution state, with the `Global frame` and `Objects` panels displaying the structure of the returned object.

JavaScript (ES6) Code:

```
14 function retirar(cantidad) {
15   if (cantidad > 0 && cantidad <= saldo) {
16     saldo -= cantidad;
17   } else {
18     console.log(
19       "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
20     );
21   }
22 }
23 // Retornamos un objeto con métodos públicos
24 return {
25   consultarSaldo: function () {
26     return saldo;
27   },
28   realizarDeposito: function (cantidad) {
29     depositar(cantidad);
30   },
31   realizarRetiro: function (cantidad) {
32     retirar(cantidad);
33   }
34 };
```

Frames:

Frame	Variable	Value
Global frame	crearCuentaBancaria	function
Global frame	miCuenta	undefined
Global frame	crearCuentaBancaria	function
Global frame	saldoInicial	1000
Global frame	saldo	1000
Global frame	depositar	function
Global frame	retirar	function

Objects:

```
function crearCuentaBancaria(saldoInicial) {
  // Propiedad privada
  var saldo = saldoInicial;
  // Método privado para depositar dinero
  function depositar(cantidad) {
    if (cantidad > 0) {
      saldo += cantidad;
    } else {
      console.log("La cantidad a depositar debe ser mayor a cero.");
    }
  }
  // Método privado para retirar dinero
  function retirar(cantidad) {
    if (cantidad > 0 && cantidad <= saldo) {
      saldo -= cantidad;
    } else {
      console.log(
        "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
      );
    }
  }
  // Retornamos un objeto con métodos públicos
  return {
    consultarSaldo: function () {
      return saldo;
    },
    realizarDeposito: function (cantidad) {
      depositar(cantidad);
    },
    realizarRetiro: function (cantidad) {
      retirar(cantidad);
    }
  };
}
```

5. Ejecuta función `crearCuentabancaria`

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6)

```

29     depositar(cantidad);
30 },
31 realizarRetiro: function (cantidad) {
32     retirar(cantidad);
33 },
34 };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo());
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo());
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo());
44 // Intento de acceder a métodos privados (no funciona)
45 // Como manejar excepciones en JavaScript utilizando try
46 try {
47     // El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
48     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
49 } catch (e) {
50     // El parámetro e es una referencia al objeto de excepción que fue lanzado
51     console.log(e.message); // message es la propiedad del objeto e, contiene una string describiendo el error
52 }
53
54 try {

```

Print output (drag lower right corner to resize)

Frames

Global frame

crearCuentaBancaria

miCuenta

Objects

```

function crearCuentaBancaria(saldoInicial) {
    // Propiedad privada
    var saldo = saldoInicial;
    // Método privado para depositar dinero
    function depositar(cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        } else {
            console.log("La cantidad a depositar debe ser mayor a cero.");
        }
    }
    // Método privado para retirar dinero
    function retirar(cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
        } else {
            console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
        }
    }
    // Retornamos un objeto con métodos públicos
    return {
        consultarSaldo: function () {
            return saldo;
        },
        realizarDeposito: function (cantidad) {
            depositar(cantidad);
        },
        realizarRetiro: function (cantidad) {
            retirar(cantidad);
        },
    };
}

```

object

consultarSaldo	function () { return saldo; }
realizarDeposito	function (cantidad) { depositar(cantidad); }

Step 5 of 34

Sponsor: interested in a [free Python tip every week?](#)

Get AI Help

[Move and hide objects](#)

6. & 9 La línea 39 imprime el saldo inicial de la cuenta bancaria en la consola. Utiliza el método **consultarSaldo** para obtener el saldo actual y luego imprime el resultado con un mensaje descriptivo. Esto confirma que la cuenta bancaria se ha creado correctamente con el saldo inicial especificado.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6)

```

34     };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45 // Como manejar excepciones en JavaScript utilizando try catch
46 try {
47     // El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
48     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
49 } catch (e) {
50     // El parámetro e es una referencia al objeto de excepción que fue lanzado
51     console.log(e.message); // message es la propiedad del objeto e, contiene una string describiendo el error
52 }
53
54 try {

```

Print output (drag lower right corner to resize)

Frames

Global frame

crearCuentaBancaria

miCuenta

Objects

```

function crearCuentaBancaria(saldoInicial) {
    // Propiedad privada
    var saldo = saldoInicial;
    // Método privado para depositar dinero
    function depositar(cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        } else {
            console.log("La cantidad a depositar debe ser mayor a cero.");
        }
    }
    // Método privado para retirar dinero
    function retirar(cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
        } else {
            console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
        }
    }
    // Retornamos un objeto con métodos públicos
    return {
        consultarSaldo: function () {
            return saldo;
        },
        realizarDeposito: function (cantidad) {
            depositar(cantidad);
        },
        realizarRetiro: function (cantidad) {
            retirar(cantidad);
        },
    };
}

```

object

consultarSaldo	function () { return saldo; }
realizarDeposito	function (cantidad) { depositar(cantidad); }

Step 6 of 34

7. & 8 La línea 26 devuelve el valor actual del saldo de la cuenta bancaria cuando se llama al método **consultarSaldo**. Este valor es utilizado por cualquier llamada a **miCuenta.consultarSaldo()** para obtener el saldo actual de la cuenta bancaria.

```

16     saldo -= cantidad;
17 } else {
18     console.log(
19 "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
20 );
21 }
22 }
23 // Retornamos un objeto con métodos públicos
24 return {
25     consultarSaldo: function () {
26         return saldo;
27     },
28     realizarDeposito: function (cantidad) {
29         depositar(cantidad);
30     },
31     realizarRetiro: function (cantidad) {
32         retirar(cantidad);
33     },
34 };
35 }

```

[Edit this code](#)

Print output (drag lower right corner to resize)

Global frame

```

creaCuentaBancaria
miCuenta
    this
    parent:saldo 1000
    parent:depositar
    parent:retirar
    Return value 1000

```

```

function creaCuentaBancaria(saldoInicial) {
    // Propiedad privada
    var saldo = saldoInicial;
    // Método privado para depositar dinero
    function depositar(cantidad) {
        if (cantidad > 0) {
            saldo += cantidad;
        } else {
            console.log("La cantidad a depositar debe ser mayor a cero.");
        }
    }
    // Método privado para retirar dinero
    function retirar(cantidad) {
        if (cantidad > 0 && cantidad <= saldo) {
            saldo -= cantidad;
        } else {
            console.log(
                "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
            );
        }
    }
    // Retornamos un objeto con métodos públicos
    return {
        consultarSaldo: function () {
            return saldo;
        },
        realizarDeposito: function (cantidad) {
            depositar(cantidad);
        },
        realizarRetiro: function (cantidad) {
            retirar(cantidad);
        },
    };
}

```

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

The screenshot shows a JavaScript IDE with the following components:

- Code Editor:** Contains a JavaScript script with a `crearCuentaBancaria` function and a `miCuenta` object. The code includes comments in Spanish and a `try-catch` block. Line 40 is highlighted in green, indicating it was just executed.
- Console:** Shows the output of the code execution, including the initial balance (1000), the deposit (500), and the withdrawal (200). The console also shows the state of the `miCuenta` object after each operation.
- Object Frames:** Displays the state of the `miCuenta` object at different points in the execution. The first frame shows the initial state with `parent:saldo` at 1000 and `parent:retirar` at 500. The second frame shows the state after the deposit, with `parent:saldo` at 1500. The third frame shows the state after the withdrawal, with `parent:saldo` at 1300.
- Navigation:** At the bottom, there are navigation buttons (`<< First`, `< Prev`, `Next >`, `Last >>`) and a step indicator (Step 10 of 34).

11. La línea 29 dentro del método `realizarDeposito` llama a la función privada `depositar` con la cantidad especificada, lo que permite agregar dinero al saldo de la cuenta bancaria. Esta llamada ejecuta la lógica de la función `depositar` para verificar y actualizar el saldo según la cantidad proporcionada.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (E6)
known limitations

```
20     );  
21 }  
22 }  
23 // Retornamos un objeto con métodos públicos  
24 return {  
25   consultarSaldo: function () {  
26     return saldo;  
27   },  
28   realizarDeposito: function (cantidad) {  
29     depositar(cantidad);  
30   },  
31   realizarRetiro: function (cantidad) {  
32     retirar(cantidad);  
33   },  
34 };  
35 }  
36  
37 // Ejemplo de uso  
38 var miCuenta = crearCuentaBancaria(1000);  
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000  
40 miCuenta.realizarDeposito(500);
```

Print output (drag lower right corner to resize)
Saldo inicial: 1000

Frames

- Global frame
 - crearCuentaBancaria
 - miCuenta
 - this
 - parent:saldo 1000
 - parent:depositar
 - parent:retirar
 - cantidad 500
 - depositar
 - parent:saldo 1000
 - parent:depositar
 - parent:retirar
 - cantidad 500

Objects

- function crearCuentaBancaria(saldoInicial) {
 // Propiedad privada
 var saldo = saldoInicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 && cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 realizarDeposito: function (cantidad) {
 depositar(cantidad);
 },
 realizarRetiro: function (cantidad) {
 retirar(cantidad);
 },
 };
}
object
consultarSaldo function () {
 return saldo;
}
depositar function (cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
}

Step 11 of 34

Sponsor: Interested in a [free Python tip every week?](#)

[Get AI Help](#)

[Move and hide objects](#)

12. La línea 7 verifica si la cantidad a depositar es mayor que 0. Esta verificación asegura que solo se puedan realizar depósitos positivos en la cuenta bancaria. Si la cantidad es mayor que 0, se incrementa el saldo. Si no, se muestra un mensaje de error indicando que la cantidad a depositar debe ser mayor a cero.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (E6)
known limitations

```
1 // Funcion fabrica para crear una cuenta bancaria  
2 function crearCuentaBancaria(saldoInicial) {  
3   // Propiedad privada  
4   var saldo = saldoInicial;  
5   // Método privado para depositar dinero  
6   function depositar(cantidad) {  
7     if (cantidad > 0) {  
8       saldo += cantidad;  
9     } else {  
10      console.log("La cantidad a depositar debe ser mayor a cero.");  
11    }  
12  }  
13  // Método privado para retirar dinero  
14  function retirar(cantidad) {  
15    if (cantidad > 0 && cantidad <= saldo) {  
16      saldo -= cantidad;  
17    } else {  
18      console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");  
19    }  
20  }  
21  // Retornamos un objeto con métodos públicos  
22  return {  
23    consultarSaldo: function () {  
24      return saldo;  
25    },  
26    realizarDeposito: function (cantidad) {  
27      depositar(cantidad);  
28    },  
29    realizarRetiro: function (cantidad) {  
30      retirar(cantidad);  
31    },  
32  };  
33 }
```

Print output (drag lower right corner to resize)
Saldo inicial: 1000

Frames

- Global frame
 - crearCuentaBancaria
 - miCuenta
 - this
 - parent:saldo 1000
 - parent:depositar
 - parent:retirar
 - cantidad 500
 - depositar
 - parent:saldo 1000
 - parent:depositar
 - parent:retirar
 - cantidad 500

Objects

- function crearCuentaBancaria(saldoInicial) {
 // Propiedad privada
 var saldo = saldoInicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 && cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 realizarDeposito: function (cantidad) {
 depositar(cantidad);
 },
 realizarRetiro: function (cantidad) {
 retirar(cantidad);
 },
 };
}
object
consultarSaldo function () {
 return saldo;
}
depositar function (cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
}

Step 11 of 34

Sponsor: Interested in a [free Python tip every week?](#)

[Get AI Help](#)

[Move and hide objects](#)

13. Realiza confirmacion si cantidad es mayor a 0

14. Al confirmar que la cantidad es mayor a 0 agrega los 500

15. La línea 40 llama al método **realizarDeposito** del objeto **miCuenta** para depositar **500** unidades de dinero en la cuenta bancaria. Esta llamada actualiza el saldo de la cuenta sumando **500** al saldo actual, pasando de **1000** a **1500**.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES5) [known limitations](#)

```

31 //
32 realizarRetiro: function (cantidad) {
33     retirar(cantidad);
34 },
35 };
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //El parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {

```

Print output (drag lower right corner to resize)

Saldo inicial: 1000

Frames

global frame

crearCuentaBancaria

miCuenta

Objects

function crearCuentaBancaria(saldoinicial) {
 // Propiedades privadas
 var saldo = saldoinicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 && cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 realizarDeposito: function (cantidad) {
 depositar(cantidad);
 },
 realizarRetiro: function (cantidad) {
 retirar(cantidad);
 },
 };
}

Step 15 of 34

16. La línea 41 imprime el saldo actual de la cuenta bancaria en la consola después de realizar un depósito. Utiliza el método `consultarSaldo` para obtener el saldo actualizado y luego imprime el resultado con un mensaje descriptivo. Esto confirma que el depósito realizado en la línea 40 se ha aplicado correctamente y que el saldo ha aumentado de **1000** a **1500**.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES5) [known limitations](#)

```

34 //
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //El parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {

```

Print output (drag lower right corner to resize)

Saldo inicial: 1000

Frames

Global frame

crearCuentaBancaria

miCuenta

parent:saldo 1500

parent:depositar

parent:retirar

Objects

function crearCuentaBancaria(saldoinicial) {
 // Propiedades privadas
 var saldo = saldoinicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 && cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disp");
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 realizarDeposito: function (cantidad) {
 depositar(cantidad);
 },
 realizarRetiro: function (cantidad) {
 retirar(cantidad);
 },
 };
}

object

consultarSaldo function () {
 return saldo;
}

Step 16 of 34

Sponsor: interested in a [free Python tip every week?](#)

[Get AI Help](#)

17. Ingresar a `consultarSaldo` para acceder al 'saldo'

18. Retornar el valor de 'consultarSaldo' y el valor de 'saldo'

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6) [known limitations](#)

```
15 // Ejemplo de uso de la función crearCuentaBancaria
16 saldo -= cantidad;
17 } else {
18   console.log(
19     "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
20   );
21 }
22 }
23 // Retornamos un objeto con métodos públicos
24 return {
25   consultarSaldo: function () {
26     return saldo;
27   },
28   realizarDeposito: function (cantidad) {
29     depositar(cantidad);
30   },
31   realizarRetiro: function (cantidad) {
32     retirar(cantidad);
33   },
34 };
35 }
36
```

Print output (drag lower right corner to resize)

Saldo inicial: 1000

Frames

Global frame

crearCuentaBancaria

miCuenta

parent:depositar

parent:retirar

Return value

Objects

function crearCuentaBancaria(saldoInicial) {
 // Propiedad privada
 var saldo = saldoInicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 & & cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 realizarDeposito: function (cantidad) {
 depositar(cantidad);
 },
 realizarRetiro: function (cantidad) {
 retirar(cantidad);
 },
 };
}

object

consultarSaldo

function () {
 return saldo;
}

Sponsor: interested in a [free Python tip every week?](#)

19. Imprime en consola el saldo

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6) [known limitations](#)

```
32   retirar(cantidad);
33 },
34 };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 // Como manejar excepciones en JavaScript utilizando try catch
47 try {
48   // El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49   miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51   // El parámetro e es una referencia al objeto de excepción que fue lanzado
52   console.log(e.message); // message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54
```

Print output (drag lower right corner to resize)

Saldo inicial: 1000
Saldo después del depósito: 1500

Frames

Global frame

crearCuentaBancaria

miCuenta

Objects

function crearCuentaBancaria(saldoInicial) {
 // Propiedad privada
 var saldo = saldoInicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 & & cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible.");
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 realizarDeposito: function (cantidad) {
 depositar(cantidad);
 },
 realizarRetiro: function (cantidad) {
 retirar(cantidad);
 },
 };
}

object

20. La línea 42 llama al método **realizarRetiro** del objeto **miCuenta** para retirar **200** unidades de dinero de la cuenta bancaria. Esta llamada actualiza el saldo de la cuenta restando **200** del saldo actual, pasando de **1500** a **1300**.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6)
[known limitations](#)

```

32  retirar(cantidad);
33  },
34  };
35  }
36
37  // Ejemplo de uso
38  var miCuenta = crearCuentaBancaria(1000);
39  console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40  miCuenta.realizarDeposito(500);
41  console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42  miCuenta.realizarRetiro(200);
43  console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44  // Intento de acceder a métodos privados (no funcionará)
45
46  // Como manejar excepciones en JavaScript utilizando try catch
47  try {
48    // El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49    miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50  } catch (e) {
51    // El parámetro e es una referencia al objeto de excepción que fue lanzado
52    console.log(e.message); // Error: miCuenta.depositar is not a function
53  }

```

⇒ line that just executed
→ next line to execute

Step 20 of 34

Sponsor: interested in a [free Python tip every week?](#)

Print output (drag lower right corner to resize)

```

Saldo inicial: 1000
Saldo después del depósito: 1500

```

Frames

Frame	Object
Global frame	function crearCuentaBancaria(saldoInicial) { // Propiedad privada var saldo = saldoInicial; // Método privado para depositar dinero function depositar(cantidad) { if (cantidad > 0) { saldo += cantidad; } else { console.log("La cantidad a depositar debe ser mayor a cero."); } } // Método privado para retirar dinero function retirar(cantidad) { if (cantidad > 0 && cantidad <= saldo) { saldo -= cantidad; } else { console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."); } } // Retornamos un objeto con métodos públicos return { consultarSaldo: function () { return saldo; }, realizarDeposito: function (cantidad) { depositar(cantidad); }, realizarRetiro: function (cantidad) { retirar(cantidad); }, }; }
crearCuentaBancaria	miCuenta
miCuenta	parent: saldo 1500 parent: depositar parent: retirar cantidad 200
retirar	parent: saldo 1500 parent: depositar parent: retirar cantidad 200
object	consultarSaldo function () { return saldo; }

21. La línea 32 dentro del método **realizarRetiro** llama a la función privada **retirar** con la cantidad especificada, lo que permite restar dinero del saldo de la cuenta bancaria. Esta llamada ejecuta la lógica de la función **retirar** para verificar y actualizar el saldo según la cantidad proporcionada.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6)
[known limitations](#)

```

23  // Retornamos un objeto con métodos públicos
24  return {
25    consultarSaldo: function () {
26      return saldo;
27    },
28    realizarDeposito: function (cantidad) {
29      depositar(cantidad);
30    },
31    realizarRetiro: function (cantidad) {
32      retirar(cantidad);
33    },
34  };
35  }
36
37  // Ejemplo de uso
38  var miCuenta = crearCuentaBancaria(1000);
39  console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40  miCuenta.realizarDeposito(500);
41  console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42  miCuenta.realizarRetiro(200);
43  console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300

```

⇒ line that just executed
→ next line to execute

Step 21 of 34

Sponsor: interested in a [free Python tip every week?](#)

Print output (drag lower right corner to resize)

```

Saldo inicial: 1000
Saldo después del depósito: 1500

```

Frames

Frame	Object
Global frame	function crearCuentaBancaria(saldoInicial) { // Propiedad privada var saldo = saldoInicial; // Método privado para depositar dinero function depositar(cantidad) { if (cantidad > 0) { saldo += cantidad; } else { console.log("La cantidad a depositar debe ser mayor a cero"); } } // Método privado para retirar dinero function retirar(cantidad) { if (cantidad > 0 && cantidad <= saldo) { saldo -= cantidad; } else { console.log("La cantidad a retirar debe ser mayor a cero y no exceder el saldo"); } } // Retornamos un objeto con métodos públicos return { consultarSaldo: function () { return saldo; }, realizarDeposito: function (cantidad) { depositar(cantidad); }, realizarRetiro: function (cantidad) { retirar(cantidad); }, }; }
crearCuentaBancaria	miCuenta
miCuenta	parent: saldo 1500 parent: depositar parent: retirar cantidad 200
retirar	parent: saldo 1500 parent: depositar parent: retirar cantidad 200
object	function () { return saldo; }

22. La línea 15 verifica si la cantidad a retirar es mayor que 0 y no excede el saldo disponible en la cuenta bancaria. Si ambas condiciones se cumplen, permite la retirada del saldo restando la cantidad especificada. Si alguna condición no se cumple, imprime un mensaje de error. Esta verificación asegura que solo se puedan realizar retiros válidos y que no se pueda retirar más dinero del que hay en la cuenta.

Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

JavaScript (ES6)
[known limitations](#)

```
1 // Retornamos un objeto con métodos públicos
2 function depositar(cantidad) {
3   if (cantidad > 0) {
4     saldo += cantidad;
5   } else {
6     console.log("La cantidad a depositar debe ser mayor a cero.");
7   }
8 }
9 // Método privado para retirar dinero
10 function retirar(cantidad) {
11   if (cantidad > 0 && cantidad <= saldo) {
12     saldo -= cantidad;
13   } else {
14     console.log(
15       "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
16     );
17   }
18 }
19 // Retornamos un objeto con métodos públicos
20 return {
21   consultarSaldo: function () {
22     return saldo;
23   },
24   depositar: function (cantidad) {
25     depositar(cantidad);
26   },
27   retirar: function (cantidad) {
28     retirar(cantidad);
29   },
30 };
31
```

⇒ line that just executed

➔ next line to execute

Step 22 of 34

<< First

< Prev

Next >

Last >>

Sponsor: interested in a [free Python tip every week?](#)

Get AI Help

Move and hide objects

Print output (drag lower right corner to resize)

Saldo inicial: 1000
Saldo después del depósito: 1500

Frames

Objects

Global frame

crearCuentaBancaria

miCuenta

this

parent:saldo 1500

parent:depositar

parent:retirar

cantidad 200

retirar

parent:saldo 1500

parent:depositar

parent:retirar

cantidad 200

function crearCuentaBancaria(saldoInicial) {
 // Propiedades privadas
 var saldo = saldoInicial;
 // Método privado para depositar dinero
 function depositar(cantidad) {
 if (cantidad > 0) {
 saldo += cantidad;
 } else {
 console.log("La cantidad a depositar debe ser mayor a cero.");
 }
 }
 // Método privado para retirar dinero
 function retirar(cantidad) {
 if (cantidad > 0 && cantidad <= saldo) {
 saldo -= cantidad;
 } else {
 console.log(
 "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
);
 }
 }
 // Retornamos un objeto con métodos públicos
 return {
 consultarSaldo: function () {
 return saldo;
 },
 depositar: function (cantidad) {
 depositar(cantidad);
 },
 retirar: function (cantidad) {
 retirar(cantidad);
 },
 };
}

object

consultarSaldo

function () {
 return saldo;
}

function (cantidad) {
 retirar(cantidad);
}

23. La línea 16 resta la cantidad especificada del saldo actual de la cuenta bancaria, actualizando el saldo después de una operación de retiro. Esta operación solo se realiza si la cantidad a retirar es válida y no excede el saldo disponible, según lo verificado en la línea 15.



24. El decremento se realiza y se imprime

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
5 // Método privado para depositar dinero
6 function depositar(cantidad) {
7   if (cantidad > 0) {
8     saldo += cantidad;
9   } else {
10     console.log("La cantidad a depositar debe ser mayor a cero.");
11   }
12 }
13 // Método privado para retirar dinero
14 function retirar(cantidad) {
15   if (cantidad > 0 && cantidad <= saldo) {
16     saldo -= cantidad;
17   } else {
18     console.log(
19 "La cantidad a retirar debe ser mayor a cero y no exceder el saldo disponible."
20 );
21 }
22 }
23 // Retornamos un objeto con métodos públicos
24 return {
25   consultarSaldo: function () {
26     return saldo;
27   }
28 }
```

[Edit this code](#)

 line that just executed
 next line to execute

<< First < Prev Next > Last >>

Step 23 of 34

24. La línea 32 dentro del método **realizarRetiro** llama a la función privada **retirar** con la cantidad especificada, lo que permite restar dinero del saldo de la cuenta bancaria. Esta llamada ejecuta la lógica de la función **retirar** para verificar y actualizar el saldo según la cantidad proporcionada.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
23 // Retornamos un objeto con métodos públicos
24 return {
25   consultarSaldo: function () {
26     return saldo;
27   },
28   realizarDeposito: function (cantidad) {
29     depositar(cantidad);
30   },
31   realizarRetiro: function (cantidad) {
32     retirar(cantidad);
33   },
34 };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
```

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Step 24 of 34

25. La línea 42 imprime el saldo actual de la cuenta bancaria en la consola después de realizar un retiro. Utiliza el método `consultarSaldo` para obtener el saldo actualizado y luego imprime el resultado con un mensaje descriptivo. Esto confirma que el retiro realizado en la línea 42 se ha aplicado correctamente y que el saldo ha disminuido de **1500** a **1300**.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
33 },
34 };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //el parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
```

→ line that just executed
→ next line to execute

[Edit this code](#)

<< First < Prev Next > Last >>

26. La línea 43 es un comentario que indica que los intentos de acceder a métodos privados (**depositar** y **retirar**) desde fuera del objeto **miCuenta** no funcionarán. El comentario sirve como explicación para el bloque de código que sigue, el cual demuestra que estos métodos no son accesibles directamente y cómo manejar los errores resultantes.

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
34     };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //el parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {
```

[Edit this code](#)

→ line that just executed
→ next line to execute

27. Accede a 'consultarSaldo' para verificar 'saldo'

28. Retorna el valor actual de 'saldo'

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
25 consultarSaldo: function () {
26     return saldo;
27 },
28 realizarDeposito: function (cantidad) {
29     depositar(cantidad);
30 },
31 realizarRetiro: function (cantidad) {
32     retirar(cantidad);
33 },
34 };
35 }
36
37 // Ejemplo de uso
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
→ 43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
```

Print output
Saldo in
Saldo de

Global f
crearCue

→ line that just executed
→ next line to execute

Edit this code

<< First < Prev Next > Last >>

Step 28 of 34

Sponsor: interested in a [free Python tip every week](#)?

29. Devuelve el saldo en consola luego del retiro

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
30 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
→ 43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
→ 49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //el parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {
55     miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
56 } catch (e) {
57     console.log(e.message);
58 }
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Step 29 of 34

30 & 31. Evalua Try y Catch para verificar si hubo algún error que se tenga que exceptuar de la recorrido del código

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
38 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //el parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {
55     miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
56 } catch (e) {
57     console.log(e.message);
58 }
```

[Edit this code](#)

→ line that just executed

→ next line to execute

<< First

< Prev

Next >

Last >>

Step 30 of 34

32. La línea 52 imprime el mensaje de error asociado con el objeto de error **e** en la consola. Este mensaje proporciona una descripción del error que ocurrió, ayudando a entender por qué falló la llamada dentro del bloque **try**. Esta línea es crucial para el manejo de errores y depuración en JavaScript.
33. Hace la captura e imprime

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

JavaScript (ES6)
[known limitations](#)

```
30 var miCuenta = crearCuentaBancaria(1000);
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //el parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {
55     miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
56 } catch (e) {
57     console.log(e.message);
58 }
```

[Edit this code](#)

→ line that just executed
→ next line to execute

34. La línea 55 intenta llamar al método **retirar** del objeto **miCuenta** directamente. Dado que **retirar** es un método privado y no está expuesto como un método público del objeto **miCuenta**, esta llamada genera un error. Este error es capturado por el bloque **catch**, que luego imprime el mensaje de error correspondiente. Este ejemplo ilustra cómo los métodos privados no pueden ser accedidos directamente desde fuera de la función **crearCuentaBancaria** y cómo manejar tales errores en JavaScript.

```
39 console.log("Saldo inicial: " + miCuenta.consultarSaldo()); // Saldo inicial: 1000
40 miCuenta.realizarDeposito(500);
41 console.log("Saldo después del depósito: " + miCuenta.consultarSaldo()); // Saldo después del depósito: 1500
42 miCuenta.realizarRetiro(200);
43 console.log("Saldo después del retiro: " + miCuenta.consultarSaldo()); // Saldo después del retiro: 1300
44 // Intento de acceder a métodos privados (no funcionará)
45
46 //Como manejar excepciones en JavaScript utilizando try catch
47 try {
48     //El código dentro de try se ejecuta. Si no hay errores, el bloque catch se omite.
49     miCuenta.depositar(100); // Error: miCuenta.depositar is not a function
50 } catch (e) {
51     //el parámetro e es una referencia al objeto de excepción que fue lanzado
52     console.log(e.message); //message es la propiedad del objeto e, contiene una string describiendo el error
53 }
54 try {
55     miCuenta.retirar(100); // Error: miCuenta.retirar is not a function
56 } catch (e) {
57     console.log(e.message);
58 }
```

[Edit this code](#)

→ line that just executed

→ next line to execute

<< First < Prev Next > Last >>

Step 33 of 34

`TypeError: miCuenta.retirar is not a function`