INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	Introduccion al desarrollo Web				
TÍTULO DE LA PRÁCTICA:	*LABORATORIO 14 – JAVASCRIPT: ORIENTACIÓN A OBJETOS I*				
NÚMERO DE PRÁCTICA:	*14*	**AÑO LECTIVO:**	*2025*	**NRO. SEMESTRE:**	*II*
FECHA DE PRESENTACIÓN	*12/11/25*	**HORA DE PRESENTACIÓN**			
INTEGRANTE (s): ● Paredes Chalco Emerson Jair				**NOTA:**	
DOCENTE(s): NORMAN PATRICK HARVEY ARCE					

SOLUCIÓN Y RESULTADOS
Trabajo personal:

```javascript
/*3. Redondeo de precios. Pide un número decimal que represente el precio de un producto y que muestre:
• Redondeo hacia abajo
• Redondeo hacia arriba
• Redondeo normal
Tip: prueba con el número 12.49 y 12.5 */
function redondeoPrecios() {
    let num = parseFloat(prompt("Ingrese su precio de producto (En decimales XX,XX)"));
    if (isNaN(num)) {
        alert("Ingrese un numero, saliendo del programa")
        return;
    }

    alert("Redondeo hacia abajo: " + Math.floor(num) +
        "\nRedondeo hacia arriba: " + Math.ceil(num) +
        "\nRedondeo hacia normal: " + Math.round(num))
}

/*4. Número aleatorio en un rango. Crear una función numeroAleatorio(min, max) que devuelva un número entero entre m
max (incluidos */
function numeroAleatorio(min, max) {
    min = Math.ceil(min);
    max = Math.floor(max);
    return Math.floor(Math.random() * (max - min + 1) + min)
}
```

```javascript
/*5. Lanzamiento de dados. Simula el lanzamiento de dos dados (valores del 1 al 6) y muestra su suma.
Tip: reutiliza la función del ejercicio anterior */
function lanzamientoDados() {
    dado1 = numeroAleatorio(1, 6);
    dado2 = numeroAleatorio(1, 6);
    return dado1 + dado2;
}
/*6. Potencias y raíces. Solicita un número y muestra su cuadrado, cubo y raíz cuadrada usando Math.pow() y Math.sqrt
*/
function potenciaRaiz() {
    const num = parseFloat(prompt("Ingrese su numero"));
    alert(num + " al cuadrado: " + Math.pow(num, 2) + "\n" +
        num + " al cubo: " + Math.pow(num, 3) + "\n" +
        num + " a la raiz cuadrada: " + Math.sqrt(num));
}
/*7. Conversión de grados a radianes y de radianes a grados. Crea una función gradosARadianes(grados) que convierta
ángulos
de grados a radianes y muestre el seno y coseno del ángulo
Tip: usa la fórmula radianes = grados * (π / 180).
Crea una función radianesAGrados(radianes) que convierta ángulos de radianes a grados
Tip: usa la fórmula grados=radianes * (180/ π) */
function gradosARadianes(grados) {
    const radianes = grados * (Math.PI / 180);
    console.log("Sen(" + grados + ") " + Math.sin(radianes) +
        "\n Cos(" + grados + ") " + Math.cos(radianes));
}
function radianesAGrados(radianes) {
    return radianes * (180 / Math.PI)
}
```

```javascript
/*8. Generar contraseñas numéricas. Crear una función que genere una contraseña aleatoria de 6 dígitos (sólo número
Tip: recorre un bucle 6 veces y genera un dígito del 0 al 9 en cada iteración*/
function contraseñaAleatoria() {
    let contraseña = "";
    for (let i = 0; i < 6; i++) {
        contraseña += numeroAleatorio(0, 9);
    }
    console.log(contraseña);
}
/*9. Calcular distancia entre dos puntos en el plano cartesiano. Dadas las coordenadas (x1, y1) y (x2, y2), calcula
distancia
entre los puntos y la suma de las distancias de cada punto al origen */
function calcularDistancia(x1, y1, x2, y2) {
    return Math.sqrt(Math.pow(y2 - y1, 2) + Math.pow(x2 - x1, 2))
}

/*10. Normalización de calificaciones. Dado un arreglo de calificaciones, normalizar todos los valores al rango 0-1
dividiendo cada
nota entre el máximo del arreglo
Tip: usa el operador de propagación: Math.max(...array). */
const calificaciones = [17, 13, 12, 2, 5, 18];
function normalizarCalificaciones(calificaciones) {
    const maximo = Math.max(...calificaciones);
    const normalizadas = calificaciones.map(nota => nota / maximo);
    return normalizadas
}
```

```javascript
/*11. Control de inventario con encapsulación. Crear una clase Producto con atributos privados nombre, precio, stoc
Implementa setters que validen que el precio y el stock sean mayores a 0
Agregar un método vender(cantidad) que disminuya el stock solo si hay unidades suficientes */
You, 3 minutes ago | 1 author (You)
class Producto {
    #nombre;
    #precio;
    #stock;
    constructor(nombre, precio, stock) {
        this.#nombre = nombre;
        this.setPrecio(precio);
        this.setStock(stock);
    }
    setPrecio(precio) {
        if (precio > 0 && !isNaN(precio))
            this.#precio = precio;
        else
            console.log("El precio tiene que ser mayor a 0");
        return;
    }
    setStock(stock) {
        if (stock > 0 && !isNaN(stock))
            this.#stock = stock;
        else {
            console.log("El stock tiene que ser mayor a 0");
            return;
        }
    }
```

```
    vender(cantidad) {
        if (isNaN(cantidad)) {
            console.log("La cantidad tiene que ser un numero");
            return;
        }
        if (cantidad < 0) {
            console.log("La cantidad tiene que ser mayor a 0");
            return;
        }
        if (cantidad <= this.#stock) {
            this.#stock -= cantidad;
            console.log("Venta realizada")
        }
        else {
            console.log("Stock insuficiente")
        }
    }
}
```

```
/*12. Modificar Producto para que el getter precio devuelva el valor con formato de moneda ($120.00) y que el sett
acepte
tanto número como cadena ("120.5")
Tip: puedes usar Number() y toFixed(2) */
You, 3 minutes ago | 1 author (You)
class Producto2 {
    #nombre;
    #precio;
    #stock;
    constructor(nombre, precio, stock) {
        this.#nombre = nombre;
        this.setPrecio(precio);
        this.setStock(stock);
    }
    setPrecio(precio) {
        const valor = Numer(precio);
        if (precio > 0 && !isNaN(precio))
            this.#precio = precio;
        else
            console.log("El precio tiene que ser mayor a 0");
        return;
    }
```

```
    setStock(stock) {
        if (stock > 0 && !isNaN(stock))
            this.#stock = stock;
        else {
            console.log("El stock tiene que ser mayor a 0");
            return;
        }
    }
    getPrecio() {
        return ("$" + this.#precio.toFixed(2));
    }
    vender(cantidad) {
        if (isNaN(cantidad)) {
            console.log("La cantidad tiene que ser un numero");
            return;
        }
        if (cantidad < 0) {
            console.log("La cantidad tiene que ser mayor a 0");
            return;
        }
        if (cantidad <= this.#stock) {
            this.#stock -= cantidad;
            console.log("Venta realizada")
        }
        else {
            console.log("Stock insuficiente")
        }
    }
}
```

```
/*13. Herencia. Crear una clase Figura. Debe tener un método area() y perimetro() que las subclases Cuadrado y Triangulo deben
sobrescribir.
Tip: llamar a constructor de la superclase */
You, 4 minutes ago | 1 author (You)
class Figura {
    constructor(nombre) {
        this.nombre = nombre;
    }
    area() {
        console.log("Area de figura");
    }
    perimetro() {
        console.log("Perimetro de figura");
    }
}
You, 4 minutes ago | 1 author (You)
class Cuadrado extends Figura {
    constructor(lado) {
        super("Cuadrado");
        this.lado = lado;
    }
    area() {
        return this.lado * this.lado;
    }
    perimetro() {
        return this.lado * 4;
    }
}
```

```
You, 4 minutes ago | 1 author (You)
class Triangulo extends Figura {
    constructor(lado1, lado2, lado3) {
        super("Triangulo");
        this.lado1 = lado1;
        this.lado2 = lado2;
        this.lado3 = lado3;
    }
    area() {
        let semiPerimetro = (this.lado1 + this.lado2 + this.lado3) / 2;
        return Math.sqrt(semiPerimetro * (semiPerimetro - this.lado1) * (semiPerimetro - this.lado2) * (semiPerimetro - this.lado3))
    }
    perimetro() {
        return this.lado1 + this.lado2 + this.lado3;
    }
}
```

```
/*14. Herencia. Crear una clase base Usuario con nombre y email. Que lo hereden Cliente y Administrador
  • Cliente tiene un nivel de fidelidad [1-5]
  • Administrador tiene permisos (crear, editar, eliminar)
Cada uno sobrescribe mostrarInfo() con diferente detalle
Tip: llama a super() para reutilizar atributos base*/
You, 4 minutes ago | 1 author (You)
class Usuario {
    constructor(nombre, email) {
        this.nombre = nombre;
        this.email = email;
    }
    mostrarInfo() {
        return ("Usuario: " + this.nombre + "| Email: " + this.email);
    }
}
You, 4 minutes ago | 1 author (You)
class Cliente extends Usuario {
    constructor(nombre, email, nivelFidelidad) {
        super(nombre, email);
        this.setNivelFidelidad(nivelFidelidad)
    }
    setNivelFidelidad(nivelFidelidad) {
        if (!isNaN(nivelFidelidad) && nivelFidelidad >= 1 && nivelFidelidad <= 5)
            this.nivelFidelidad = Number(nivelFidelidad);
        else
            console.log("Nivel de fidelidad debe estar entre 1 y 5");
    }
    mostrarInfo() {
        return ("Cliente:" + this.nombre + "| Email: " + this.email + "| Nivel de Fidelidad: " + this.nivelFidelidad)
    }
}
You, 4 minutes ago | 1 author (You)
class Administrador extends Usuario {
    constructor(nombre, email, permisos) {
        super(nombre, email);
        this.permisos = permisos;
    }
    mostrarInfo() {
        return ("Administrador:" + this.nombre + "| Email: " + this.email + "| Permisos: " + this.permisos);
    }
}
/*15. Polimorfismo. Crear una lista de usuarios (Cliente, Administrador) y recórrela mostrando la información con
mostrarInfo().
Tip: usa un forEach o for...of para recorrer el array */
const usuarios = [
    new Cliente("Ana López", "ana@mail.com", 4),
    new Administrador("Carlos Pérez", "carlos@admin.com", ["crear", "editar"]),
    new Cliente("Luis Ramos", "luis@mail.com", 2),
    new Administrador("María Torres", "maria@admin.com", ["eliminar", "editar", "crear"])
];
```

```
/*16. Herencia. Crear la jerarquía Empleado - Programador - ProgramadorSenior
   • Empleado tiene nombre y sueldoBase
   • Programador añade lenguaje y método calcularSueldo() con bono del 10%
   • ProgramadorSenior sobreescribe calcularSueldo() con un bono del 25%
   Tip: llama a super.calcularSueldo() desde la subclase */
You, 4 minutes ago | 1 author (You)
class Empleado {
    constructor(nombre, sueldoBase) {
        this.nombre = nombre;
        this.sueldoBase = sueldoBase;
    }
    calcularSueldo() {
        return this.sueldoBase;
    }
}
You, 4 minutes ago | 1 author (You)
class Programador extends Empleado {
    constructor(nombre, sueldoBase, lenguaje) {
        super(nombre, sueldoBase);
        this.lenguaje = lenguaje;
    }
    calcularSueldo() {
        return this.sueldoBase * 1.10;
    }
}
```

```
You, 4 minutes ago | 1 author (You)
class ProgramadorSenior extends Programador {
    constructor(nombre, sueldoBase, lenguaje) {
        super(nombre, sueldoBase, lenguaje);
    }
    calcularSueldo() {
        const sueldoBaseMasBono = super.calcularSueldo();
        return sueldoBaseMasBono * 1.15;
    }
}
```

```
/*17. Encapsulación y polimorfismo. Crear una clase Cuenta con atributo privado saldo y métodos depositar() y retirar
Luego crea subclases CuentaAhorro y CuentaCorriente que redefinan retirar() según sus reglas (por ejemplo, permitir
sobregiro en la cuenta corriente pero con un límite) y también la transferencia entre cuentas
Tip: implementa validaciones distintas en cada clase hija.*/
...
class Cuenta {
    #saldo;
    constructor(saldoInicial) {
        if (!isNaN(saldoInicial) && saldoInicial >= 0)
            this.#saldo = saldoInicial;
        else {
            console.log("El saldo inicial debe ser un número mayor o igual a 0");
            this.#saldo = 0;
        }
    }

    getSaldo() {
        return this.#saldo;
    }

    depositar(monto) {
        if (isNaN(monto) || monto <= 0) {
            console.log("El monto a depositar debe ser un número mayor que 0");
            return;
        }
        this.#saldo += monto;
        console.log("Depósito realizado. Nuevo saldo: $" + this.#saldo.toFixed(2));
    }

```

```
retirar(monto) {
    if (isNaN(monto) || monto <= 0) {
        console.log("El monto a retirar debe ser un número mayor que 0");
        return;
    }
    if (monto <= this.#saldo) {
        this.#saldo -= monto;
        console.log("Retiro realizado. Nuevo saldo: $" + this.#saldo.toFixed(2));
    } else {
        console.log("Fondos insuficientes");
    }
}

transferir(monto, cuentaDestino) {
    if (!(cuentaDestino instanceof Cuenta)) {
        console.log("Cuenta destino inválida");
        return;
    }
    if (isNaN(monto) || monto <= 0) {
        console.log("El monto a transferir debe ser un número mayor que 0");
        return;
    }
    if (monto > this.#saldo) {
        console.log("Fondos insuficientes para transferir");
        return;
    }
    this.#saldo -= monto;
    cuentaDestino.depositar(monto);
    console.log("Transferencia realizada con éxito");
}
```

```
class CuentaAhorro extends Cuenta {
    constructor(saldoInicial) {
        super(saldoInicial);
    }

    retirar(monto) {
        if (monto > this.getSaldo()) {
            console.log("No se puede retirar más del saldo disponible en cuenta de ahorro");
            return;
        }
        super.retirar(monto);
    }
}

You, 5 minutes ago | 1 author (You)
class CuentaCorriente extends Cuenta {
    constructor(saldoInicial, limiteSobregiro) {
        super(saldoInicial);
        this.limiteSobregiro = limiteSobregiro;
    }
```

```
    retirar(monto) {
        if (isNaN(monto) || monto <= 0) {
            console.log("El monto a retirar debe ser un número válido");
            return;
        }
        if (monto <= this.getSaldo() + this.limiteSobregiro) {
            const nuevoSaldo = this.getSaldo() - monto;

            super.depositar(nuevoSaldo - this.getSaldo());
            console.log("Retiro con sobregiro autorizado. Nuevo saldo: $" + this.getSaldo().toFixed(2));
        } else {
            console.log("Límite de sobregiro excedido");
        }
    }
}
```

```
/*18. Composición. Crear una clase Carrito que contenga una lista de objetos Producto. Agrega métodos agregarProducto(
calcularTotal() y mostrarResumen()
Tip: usa un array de objetos */
```

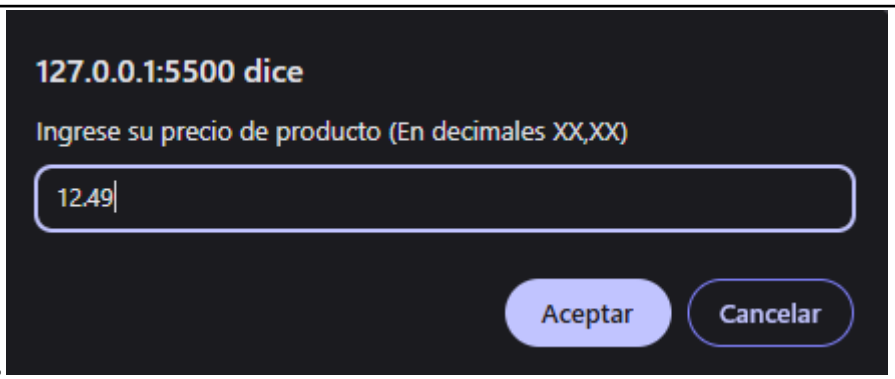
```
class Carrito {
    constructor() {
        this.productos = [];
    }

    agregarProducto(producto) {
        if (producto instanceof Producto2) {
            this.productos.push(producto);
            console.log("Producto agregado al carrito");
        } else {
            console.log("Solo se pueden agregar objetos de tipo Producto2");
        }
    }

    calcularTotal() {
        let total = 0;
        this.productos.forEach(p => {
            total += Number(p.getPrecio().replace("$", ""));
        });
        return total;
    }

    mostrarResumen() {
        console.log("Resumen del carrito:");
        this.productos.forEach((p, index) => {
            console.log((index + 1) + ". " + p.getPrecio());
        });
        console.log("Total a pagar: $" + this.calcularTotal().toFixed(2));
    }
```

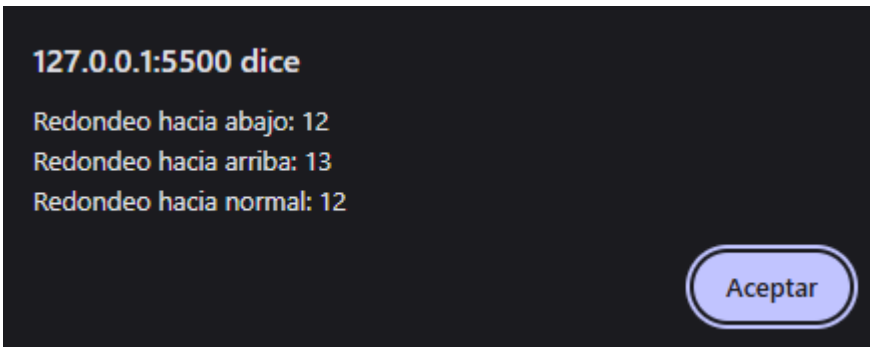
```
/*19. Polimorfismo. Crear una clase base Notificacion con un método enviar(). Implementa subclases Email, SMS y Push q
sobrescriban enviar() con mensajes distintos. Luego crea una función que reciba una lista de notificaciones y las
procese
dinámicamente
Tip: usa una estructura de datos adecuada que almacene objetos y que llame a notificacion.enviar() en un bucle*/
...
class Notificacion {
    constructor(destinatario, mensaje) {
        this.destinatario = destinatario;
        this.mensaje = mensaje;
    }

    enviar() {
        console.log("Enviando notificación genérica a " + this.destinatario);
    }
}

...
class Email extends Notificacion {
    enviar() {
        console.log("📧 Enviando correo a " + this.destinatario + ": " + this.mensaje);
    }
}

...
```

```
class SMS extends Notificacion {
    enviar() {
        console.log("📱 Enviando SMS a " + this.destinatario + ": " + this.mensaje);
    }
}

You, 1 minute ago | 1 author (You)
class Push extends Notificacion {
    enviar() {
        console.log("🔔 Enviando notificación push a " + this.destinatario + ": " + this.mensaje);
    }
}

function procesarNotificaciones(lista) {
    for (let notificacion of lista) {
        if (notificacion instanceof Notificacion)
            notificacion.enviar();
        else
            console.log("Elemento inválido en la lista");
    }
```

Prueba:

3.





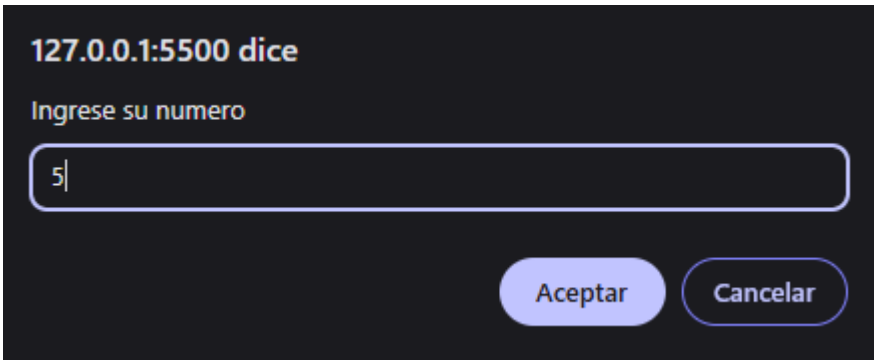
4.

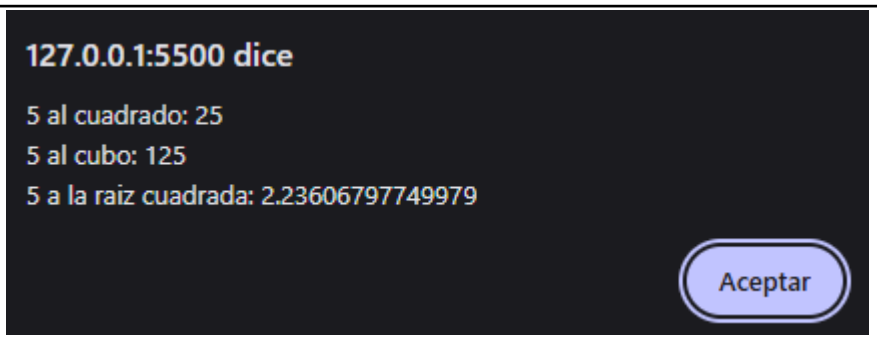
```
> numeroAleatorio(1,5)
< 5
```

5.

```
> lanzamientoDados()
< 6
```

6.





7.
```
> gradosARadianes(60)
Sen(60) 0.8660254037844386
 Cos(60) 0.5000000000000001
```

```
> radianesAGrados(2*Math.PI)
< 360
```

8.
```
> contraseñaAleatoria()
821463
```

9.
```
> calcularDistancia(1,2,3,4)
< 2.8284271247461903
```

10.
```
> normalizarCalificaciones(calificaciones);
< (6) [0.9444444444444444, 0.7222222222222222, 0.6666666666666666, 0.
1111111111111111, 0.2777777777777778, 1]
    0: 0.9444444444444444
    1: 0.7222222222222222
    2: 0.6666666666666666
    3: 0.1111111111111111
    4: 0.2777777777777778
    5: 1
    length: 6
```

11.
```
> const p1=new Producto("Mouse",10,2);
< undefined
> p1.vender(2);
Venta realizada
< undefined
> p1.vender(1);
Stock insuficiente
```

12.

```
> const p2=new Producto2("Teclado","120.5",5);
< undefined
> console.log(p2.getPrecio());
  $120.50
```

13.

```
> const cuadrado=new Cuadrado(4);
< undefined
> const triangulo=new Triangulo(3,4,5);
< undefined
> console.log(cuadrado.area());
  16
< undefined
> console.log(triangulo.area());
  6
< undefined
> console.log(cuadrado.perimetro());
  16
< undefined
> console.log(triangulo.perimetro());
  12
```

14 y 15.

```
usuarios.forEach(u => console.log(u.mostrarInfo()));
Cliente:Ana López| Email: ana@mail.com| Nivel de Fidelidad: 4
Administrador:Carlos Pérez| Email: carlos@admin.com|
Permisos: crear,editar
Cliente:Luis Ramos| Email: luis@mail.com| Nivel de Fidelidad:
2
Administrador:María Torres| Email: maria@admin.com| Permisos:
eliminar,editar,crear
```

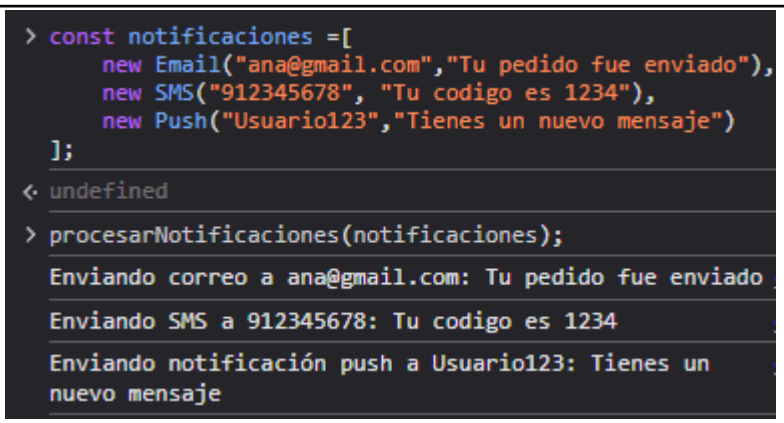
16.

```
> console.log(prog.calcularSueldo())
  2200
< undefined
> console.log(senior.calcularSueldo())
  3795.0000000000005
< undefined
```

17.
```
> const ahorro=new CuentaAhorro(500);
< undefined
> const corriente = new CuentaCorriente(200,300);
< undefined
> ahorro.retirar(100);
  Retiro realizado. Nuevo saldo: $400.00
< undefined
> corriente.retirar(400);
  Retiro con sobregiro autorizado. Nuevo saldo: $-200.00
< undefined
> corriente.retirar(600);
  Límite de sobregiro excedido
< undefined
> corriente.depositar(250);
  Depósito realizado. Nuevo saldo: $50.00
< undefined
> ahorro.transferir(100,corriente);
  Depósito realizado. Nuevo saldo: $150.00
  Transferencia realizada con éxito
```

18.
```
> const prod1=new Producto2("Pan",3.5,10);
< undefined
> const prod2=new Producto2("Leche", "4.2",5);
< undefined
> const carrito= new Carrito();
< undefined
> carrito.agregarProducto(prod1);
  Producto agregado al carrito
< undefined
> carrito.agregarProducto(prod2);
  Producto agregado al carrito
< undefined
> carrito.mostrarResumen();
  Resumen del carrito:
  1. $3.50
  2. $4.20
  Total a pagar: $7.70
```

19.



Link a github

https://github.com/TioSniperxD/Lab14IDWEB