

# Resolución de la etapa 4

## 1. Componentes de la aplicación

### Clase Router



Genera una clase `IngresoDineroComponent` en JavaScript con un constructor que reciba `parent` (elemento HTML contenedor), `model` (datos de la cuenta actual), y `router` (objeto Router). Incluye métodos `handleVolver` para regresar al componente de cuenta y `handleIngresarDinero` para realizar el ingreso de dinero, y un método `render` que muestre la interfaz para ingresar dinero, incluyendo campos para el monto y motivo, botones estilizados para confirmar el ingreso y volver, y la asignación de eventos de clic a cada botón.



Aquí tienes una implementación de la clase `IngresoDineroComponent` en JavaScript. Esta clase incluye un constructor que recibe los parámetros `parent`, `model`, y `router`. Además, contiene métodos para manejar la navegación de regreso a la vista de cuenta (`handleVolver`), realizar el ingreso de dinero (`handleIngresarDinero`), y renderizar la interfaz para ingresar dinero (`render`).

```
class Router {  
  constructor(parent, model){  
    this.parent = parent;  
    this.model = model;  
  }  
  
  navigateTo(Componente, model){  
    let componente = new Componente(this.parent, this.model, this);  
    componente.render();  
  }  
}
```

## Componente de *Login*

Genera una clase LoginComponent en JavaScript con un constructor que reciba parent (el elemento HTML contenedor), model (el objeto que maneja los datos y la lógica de autenticación), y router (el objeto Router). Esta clase debe tener los métodos handleRegistrarse para navegar a un RegistroComponent, handleIniciarSesion para manejar el inicio de sesión con validación de username y password, y render para construir y mostrar el formulario de inicio de sesión con eventos de clic para los botones de registro e inicio de sesión. Usar innerHTML



Para crear la clase `LoginComponent` en JavaScript con los requisitos que has mencionado, puedes seguir este ejemplo:

```
javascript
```

[Copiar código](#)

```
//Todo. Constructor
class LoginComponent {

    constructor(parent, model, router){
        this.parent = parent;
        this.model = model;
        this.router = router;
    }

    handleRegistrarse(){
        this.router.navigateTo(RegistroComponent)
    }
}
```

...

```
handleIniciarSesion(){
  try{

    const username = this.parent.querySelector('#login_txtUsername').value;
    const password = this.parent.querySelector('#login_txtPassword').value;
    this.model.login(username, password);
    this.router.navigateTo(CuentaComponent);
  } catch(excepcion){
    document.querySelector("#login_mstError").innerHTML = excepcion;
  }
}
```

...

...

```
render(){
  this.parent.innerHTML = `
    <style>
      /* styles.css */
    </style>
    <div class="login-body">
      <div class="login-container">
        <form class="login-form">
          <h2>Iniciar sesión</h2>
          <label for="username">Nombre de usuario</label>
          <input type="text" id="login_txtUsername" name="username" required>

          <label for="password">Contraseña</label>
          <input type="password" id="login_txtPassword" name="password" required>

          <button type="button" id="login_btnIniciarSesion">
            Iniciar sesión
          </button>
        </form>
      </div>
    </div>
  `
}
```

...

...

```
        <button type="button" id="login_btnRegistrarse" class="register-button">
            Registrarse
        </button>

        <div id="login_mstError">

        </div>
    </form>
</div>
</div>
`;

const btnRegistrarse = this.parent.querySelector('#login_btnRegistrarse');
btnRegistrarse.addEventListener('click', () => this.handleRegistrarse());

const btnLogin = this.parent.querySelector('#login_btnIniciarSesion');
btnLogin.addEventListener("click", ()=> { this.handleIniciarSesion()});
}
}
```

Resultado final:

### Proyecto integrador Introducción a Objetos

#### Iniciar sesión

Nombre de usuario

Contraseña

Iniciar sesión

Registrarse

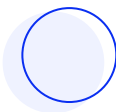


## Componente de *Cuenta de usuario*

Genera una clase `CuentaComponent` en JavaScript con un constructor que reciba `parent` (elemento HTML contenedor), `model` (datos de la cuenta actual), y `router` (objeto `Router`). Incluye métodos `handleSalir`, `handleIngresarDinero`, `handleVerMovimientos`, y `handleTransferir` para manejar la navegación a los componentes correspondientes, y un método `render` que muestre la interfaz de la cuenta con botones estilizados para estas acciones y asigne eventos de clic a cada botón. Utilizar `innerHTML` en el `render`.



Claro, aquí te dejo una implementación en JavaScript de la clase `CuentaComponent` que cumple con los requisitos mencionados. Esta clase utiliza el método `innerHTML` para renderizar la interfaz de usuario y asigna eventos de clic a los botones para manejar las diferentes acciones. Asegúrate de tener en cuenta que la estructura del HTML y la interacción con el `Router` podrían variar según el contexto específico en el que se utilice esta clase.



```
class CuentaComponent {  
  constructor(parent, model, router){  
    this.parent = parent;  
    this.model = model;  
    this.router = router;  
  }  
  
  handleSalir(){  
    this.router.navigateTo(LoginComponent);  
  }  
  
  handleIngresarDinero(){  
    this.router.navigateTo(IngresoDineroComponent);  
  }  
  
  handleVerMovimientos(){  
    this.router.navigateTo(MovimientosComponent);  
  }  
  
  handleTransferir(){  
    this.router.navigateTo(TransferenciaComponent);  
  }  
}
```

...

```
render(){
  this.parent.innerHTML = `
    <style>../style>
    <div class="card">
      <h2 class="username">
        Bienvenido ${this.model.cuentaActual.cliente.nombre}</span>
      </h2>

      <p id="cuenta_msgBalance" class="balance">$ ${this.model.cuentaActual.saldo}</p>

      <div class="btn-container">
        <button class="btn transfer" id="cuenta_btnTransferencia">Transferir dinero</button>
        <button class="btn deposit" id="cuenta_btnIngresarDinero">Ingresar dinero</button>
        <button class="btn transactions" id="cuenta_btnVerMovimientos">Ver movimientos</button>
        <button id="cuenta_btnSalir" class="btn transactions">Salir</button>
      </div>
    </div>
  `
}
```

...

...

```
    this.parent.querySelector("#cuenta_btnSalir").addEventListener("click",  
        ()=>{this.handleSalir()})  
    this.parent.querySelector("#cuenta_btnIngresarDinero").addEventListener("click",  
        ()=>{this.handleIngresarDinero()})  
    this.parent.querySelector("#cuenta_btnVerMovimientos").addEventListener("click",  
        ()=>{this.handleVerMovimientos()})  
    this.parent.querySelector("#cuenta_btnTransferencia").addEventListener("click",  
        ()=>{this.handleTransferir()})
```

}

}



Resultado final:



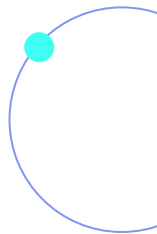
## Componente de transferencia



Genera una clase `TransferenciaComponent` en JavaScript con un constructor que reciba `parent` (elemento HTML contenedor), `model` (datos de la cuenta actual), y `router` (objeto Router). Incluye métodos `handleVolver` y `handleTransferir` para manejar la navegación de vuelta al componente de cuenta y realizar la transferencia de dinero, respectivamente, y un método `render` que muestre la interfaz para transferir dinero, incluyendo campos para correo del destinatario y monto, botones estilizados para confirmar la transferencia y volver, y la asignación de eventos de clic a cada botón.



Aquí tienes la clase `TransferenciaComponent` en JavaScript, diseñada para manejar la transferencia de dinero entre cuentas, con un constructor que acepta el contenedor HTML, los datos del modelo, y el router. La clase incluye métodos para manejar la navegación de vuelta y la transferencia de dinero, además de un método `render` que muestra la interfaz:



```
class TransferenciaComponent {  
  constructor(parent, model, router){  
    this.parent = parent;  
    this.model = model;  
    this.router = router;  
  }  
  
  handleVolver(){  
    this.router.navigateTo(CuentaComponent);  
  }  
  
  handleTransferir(){  
    try{  
      const mailDestino = this.parent.querySelector('#transferir_txtEmail').value;  
      const monto = this.parent.querySelector('#transferir_txtAmount').valueAsNumber;  
      this.model.transferir(this.model.cuentaActual.codigo, mailDestino, monto);  
      this.router.navigateTo(CuentaComponent);  
    }catch(exception){  
      this.parent.querySelector("#transferencia_msgError").innerHTML = exception  
    }  
  }  
}
```

...

```
render(){
  this.parent.innerHTML = `
    <div class="container mx-auto p-4">
      <div class="bg-white dark:bg-zinc-800 rounded-lg shadow-md p-4 w-full md:w-1/3 mx-auto">
        <h2 class="text-2xl font-bold mb-4">Transferencia</h2>
        <div class="mb-4">
          <label for="email" class="block text-sm font-medium text-zinc-700">Correo del destinatario</label>
          <input
            type="email"
            id="transferir_txtEmail"
            name="email"
            placeholder="ejemplo@correo.com"
            class="w-full px-3 py-2 border rounded-lg"
          />
        </div>
      </div>
    `
  }
```

...



...

```
<div class="mb-4">
  <label for="amount" class="block text-sm font-medium text-zinc-700">Monto</label>
  <input
    type="number"
    id="transferir_txtAmount"
    placeholder="Ingresa el monto"
    class="w-full px-3 py-2 border rounded-lg"
  />
</div>
<button id="transferir_btnTransferir" type="button" class="bg-blue-500 text-white p-2 rounded-lg
w-full" >
  Transferir
</button>
<button id="transferir_btnVolver" type="button" class="bg-red-500 text-white p-2 rounded-lg
w-full">
  Volver
</button>
```

...

■ ■ ■

```

        <div class="red-500" id="transferencia_msgError">

        </div>
    </div>
</div>
`;

this.parent.querySelector("#transferir_btnVolver").addEventListener("click",
    ()=>{ this.handleVolver()});
this.parent.querySelector("#transferir_btnTransferir").addEventListener("click",
    ()=>{ this.handleTransferir()});
}
}

```

Resultado final:

**Proyecto integrador**  
**Introducción a Objetos**

**Transferencia**  
Correo del destinatario  
  
Monto  
  

Transferir

Volver

## Componente para simular el ingreso del dinero

Genera una clase `IngresoDineroComponent` en JavaScript con un constructor que reciba `parent` (elemento HTML contenedor), `model` (datos de la cuenta actual), y `router` (objeto Router). Incluye métodos `handleVolver` para regresar al componente de cuenta y `handleIngresarDinero` para realizar el ingreso de dinero, y un método `render` que muestre la interfaz para ingresar dinero, incluyendo campos para el monto y motivo, botones estilizados para confirmar el ingreso y volver, y la asignación de eventos de clic a cada botón.



Aquí tienes una implementación de la clase ``IngresoDineroComponent`` en JavaScript. Esta clase incluye un constructor que recibe los parámetros ``parent``, ``model``, y ``router``. Además, contiene métodos para manejar la navegación de regreso a la vista de cuenta (``handleVolver``), realizar el ingreso de dinero (``handleIngresarDinero``), y renderizar la interfaz para ingresar dinero (``render``).



```
class IngresoDineroComponent {  
  
  constructor(parent, model, router){  
    this.parent = parent;  
    this.model = model;  
    this.router = router;  
  }  
  
  handleVolver(){  
    this.router.navigateTo(CuentaComponent);  
  }  
  
  handleIngresarDinero(){  
    const monto = this.parent.querySelector('#ingreso_txtMonto').valueAsNumber;  
    const motivo = this.parent.querySelector('#ingreso_txtMotivo').value;  
    this.model.cuentaActual.depositar(monto, motivo);  
    this.router.navigateTo(CuentaComponent);  
  }  
}
```

...

```
render(){
  this.parent.innerHTML = `
    <div class="bg-gray-100 p-6">
      <div class="max-w-md mx-auto bg-white shadow-md rounded-lg p-6">
        <h1 class="text-2xl font-bold mb-4 text-center">Simulación de ingreso de dinero</h1>
        <form id="simulacion-form" class="space-y-4">
          <div>
            <label for="monto" class="block text-lg font-medium text-gray-700">Monto</label>
            <input type="number" id="ingreso_txtMonto" name="monto" placeholder="Ingresa el monto"
class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none
focus:ring-blue-500 focus:border-blue-500" required>
          </div>
          <div>
            <label for="monto" class="block text-lg font-medium text-gray-700">Motivo</label>
            <input type="text" id="ingreso_txtMotivo" name="monto" placeholder="Ingresa el motivo"
class="mt-1 block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm focus:outline-none
focus:ring-blue-500 focus:border-blue-500" required>
          </div>
        </form>
      </div>
    </div>
  `
}
```

...

...

```
<button type="button" id="ingreso_btnIngresarDinero"
  class="w-full bg-blue-500 hover:bg-blue-600 text-white font-bold py-2 px-4 rounded-md">
  Simular ingreso
</button>
```

```
<button type="button" id="ingreso_btnVolver"
  class="w-full bg-blue-500 hover:bg-blue-600 text-white font-bold py-2 px-4 rounded-md">
  Volver
</button>
```

```
</form>
```

```
</div>
```

```
</div>
```

```
this.parent.querySelector("#ingreso_btnVolver").addEventListener("click", ()=>{ this.handleVolver() });
this.parent.querySelector("#ingreso_btnIngresarDinero").addEventListener("click",
  ()=>{ this.handleIngresarDinero() });
```

```
}
```

```
}
```

Resultado final:



## Proyecto integrador Introducción a Objetos

### Simulación de ingreso de dinero

Monto

Motivo

Simular ingreso

Volver