



Universidad Centroccidental "Lisandro Alvarado"
Decanato de Ciencias y Tecnología
Análisis de Sistemas
Departamento de Sistemas
Programación



Arquitectura Modelo - Vista - Controlador



UNIDAD III. Metodología para el Desarrollo de Programas Orientado a Objetos

Objetivo General

Implementar soluciones con programación para problemas planteados aplicando la metodología para el desarrollo de programas orientados a objetos y la arquitectura MVC.

Objetivos Específicos

- Identificar los elementos que componen el patrón de diseño MVC (modelo, vista y controlador).
- Analizar el planteamiento de un problema y su diseño correspondiente con metodología MVC.
- Programar aplicaciones siguiendo un diseño con metodología MVC.
- Identificar en el diagrama de clases de un problema planteado los diversos componentes de la programación respectiva.
- Aplicar la metodología MVC usando estructuras de datos complejas.



Patrón de Diseño MVC



Contenido

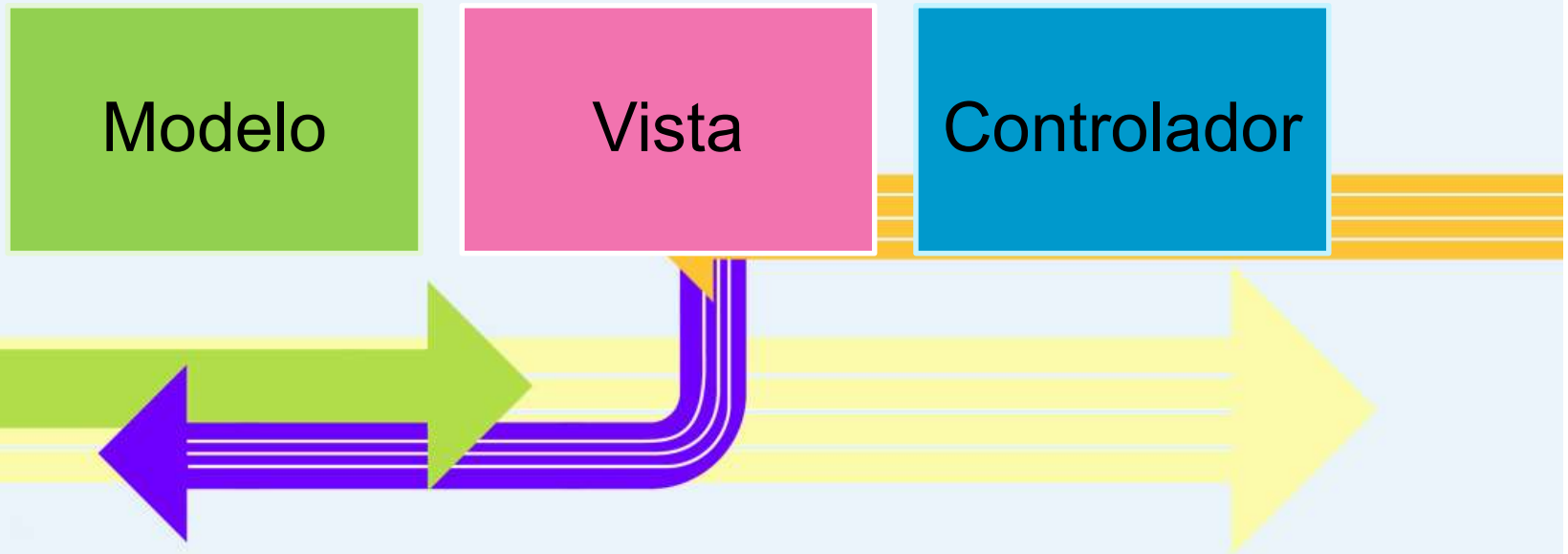
- Definición de Patrón MVC
- Componentes del Patrón MVC
- Ciclo de funcionamiento
- Análisis, Diseño e Implementación C++ usando el Patrón MVC
- Ventajas y Desventajas del Patrón MVC



Patrón de Diseño MVC

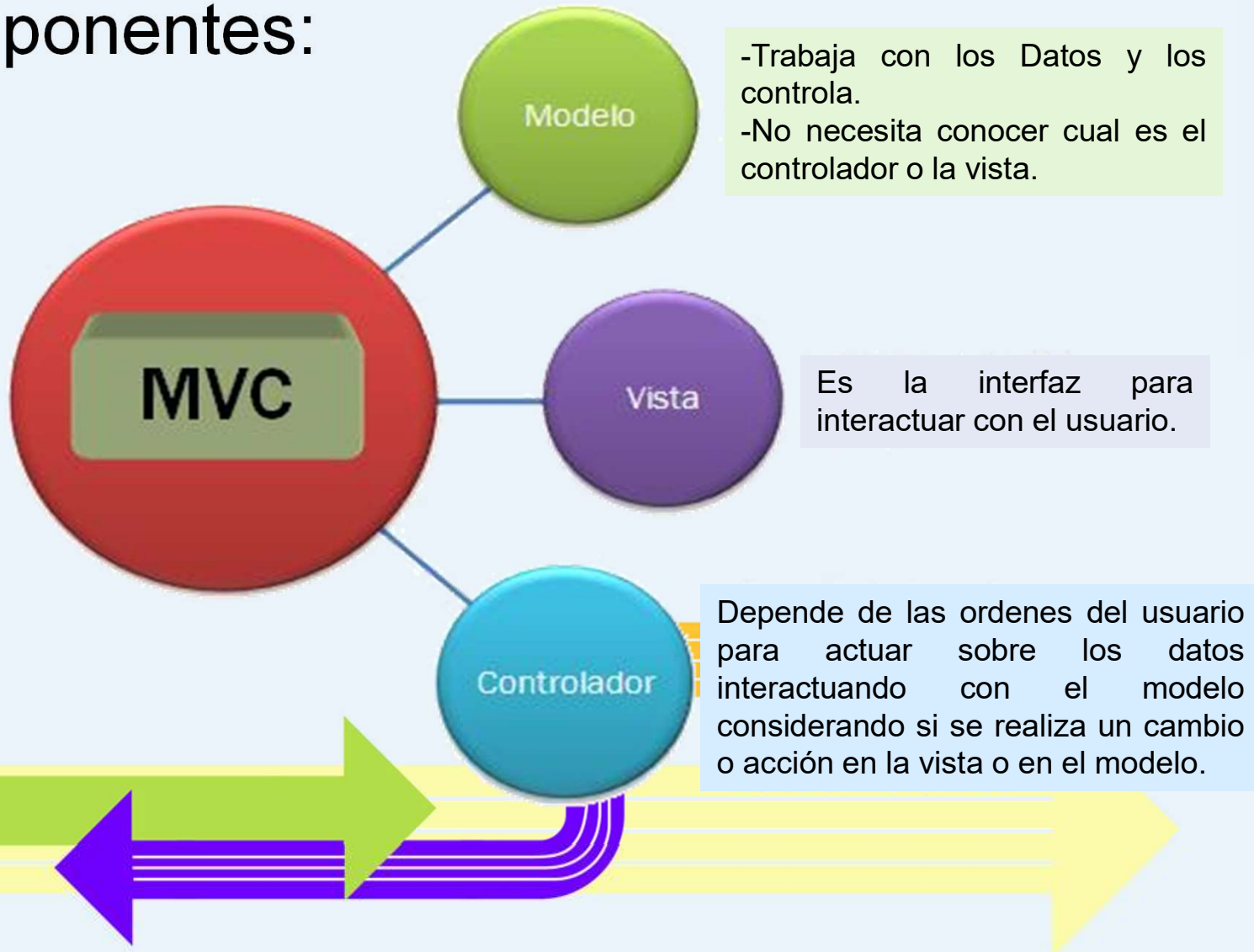
Definición

Es un patrón de arquitectura de software que separa los datos de una aplicación y la lógica de negocio, de la interfaz de usuario, implementándolos en componentes distintos.



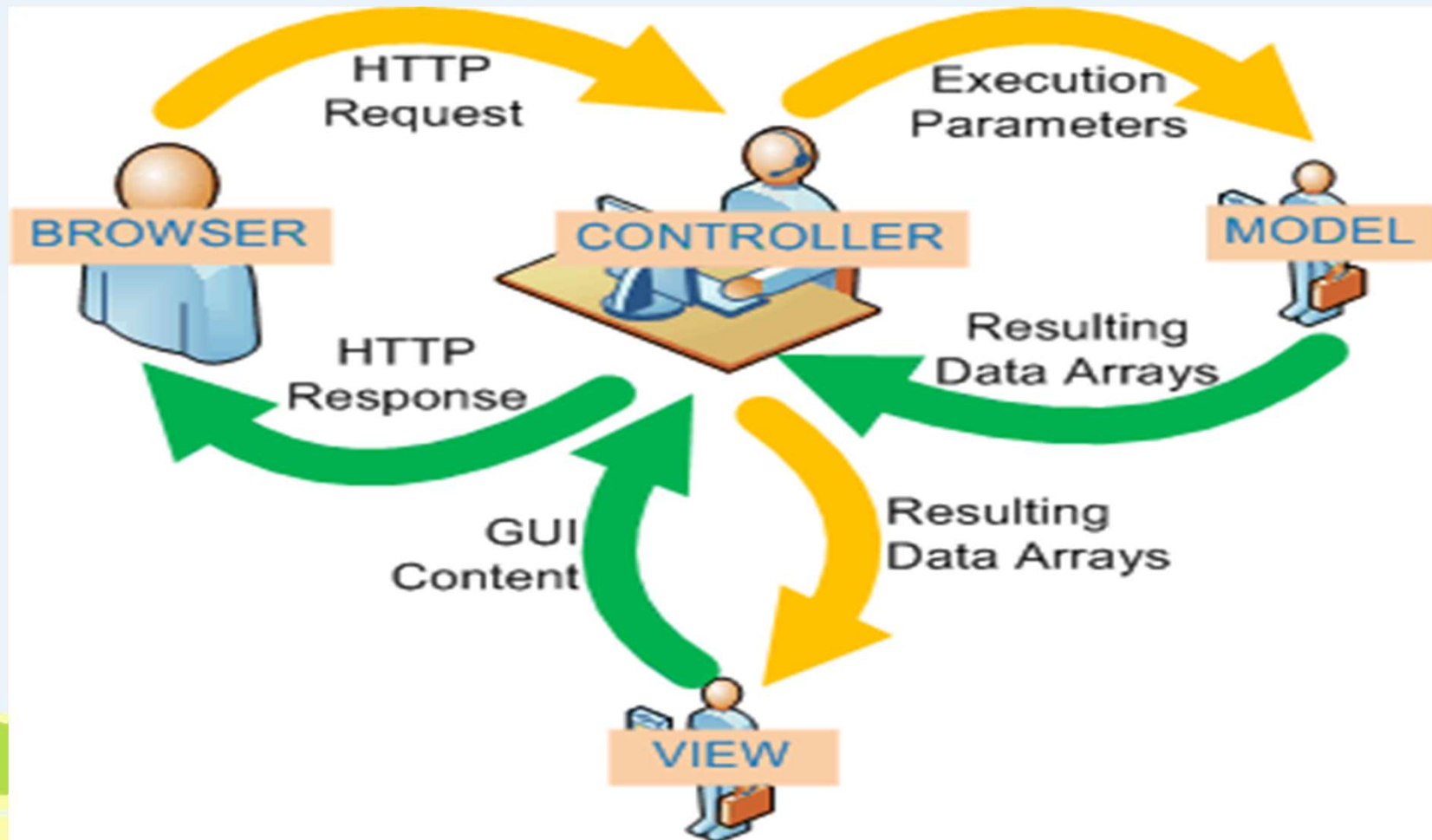
Patrón de Diseño MVC

Componentes:



Patrón de Diseño MVC

Ciclo de funcionamiento



Patrón MVC: Análisis



Enunciado: **Aerolínea SUPERVUELOS**

Una Aerolínea ofrece vuelos diariamente. Por cada uno de los vuelos se conoce: número de vuelo, tipo de vuelo (1.-Nacional, 2.-Internacional), costo del pasaje y número de pasajeros transportados. La línea aérea desea determinar el ingreso por vuelo, ingreso total del día, menor número de pasajeros transportados en un vuelo y porcentaje de vuelos internacionales.

Entradas: número de vuelo, tipo de vuelo, costo y número de pasajeros.

Salidas:

Por Vuelo:

1) Ingreso del vuelo

Por Aerolínea:

2) Ingreso Total del Día

3) Menor Número de Pasajeros Transportados en un Vuelo.

4) Porcentaje de Vuelos Internacionales.

Patrón MVC: Análisis

El análisis ya lo aprendimos en el 1er corte:
ES EXACTO IGUAL

numero	tipo	costo	pasajeros	1. ingreso()
1111	2	20	120	2400
2222	1	10	60	600
3333	2	30	100	3000
4444	2	15	80	1200

2. Ingreso Total del Día: 7200 Bs.

3. Menor Número de Pasajeros en un vuelo: 60 Pasajeros

4. Porcentaje de vuelos internacionales: 75 %

Abstracción

1. Costo * pasajeros
2. PROCESO UNIVERSAL DE ACUMULADOR.
3. PROCESO UNIVERSAL DE MENOR.
4. PROCESO UNIVERSAL DE PORCENTAJE:

$\text{cntVuelosInternacionales} / \text{cntVuelos}$

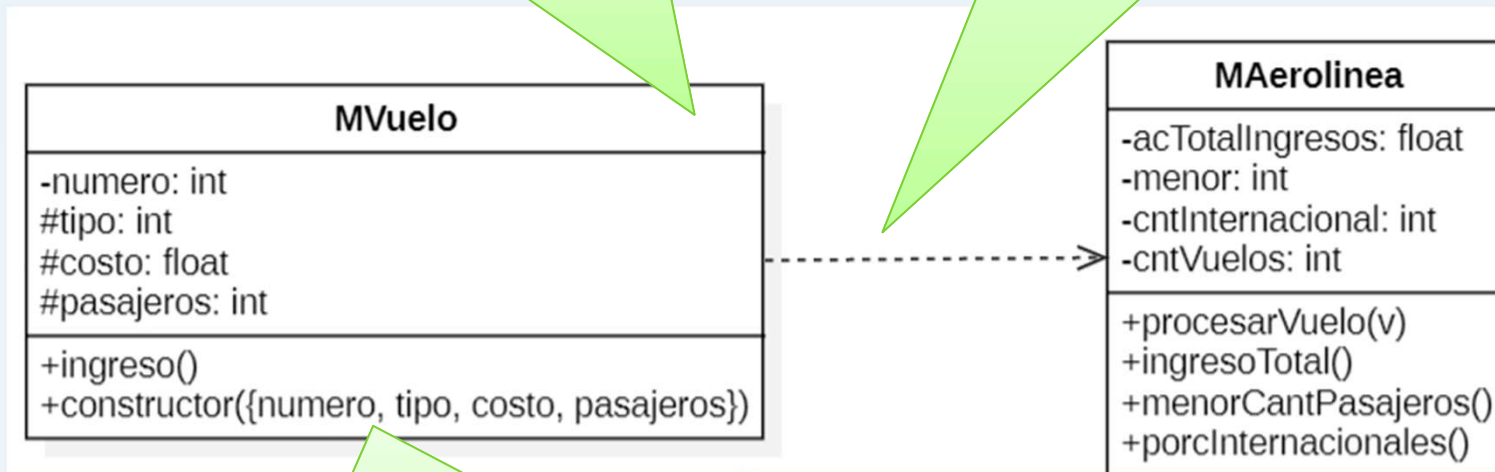
Tabla de Datos

Patrón MVC: Diseño

Diagrama de Clases: **Modelo**

Las clases que hemos programado hasta ahora son el modelo

Acá se denota una relación de tipo USO ya que el objeto MVuelo se para ya formado a MAerolinea



OBSERVA la nueva forma de inicializar un objeto: el constructor recibe otro objeto (datos entre llaves). Esto nos dará una serie de ventajas que iremos aprendiendo poco a poco.

Otro tipo de relación: **AGREGACIÓN**... Se pasan los datos para que el otro objeto cree (agregue) al menor

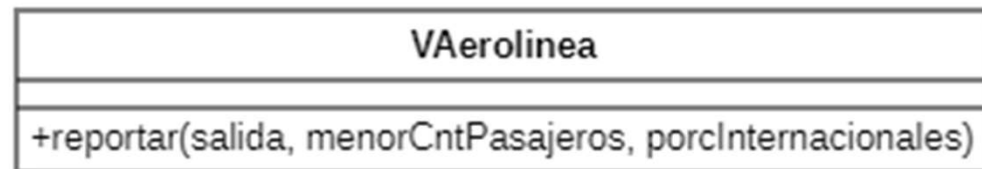
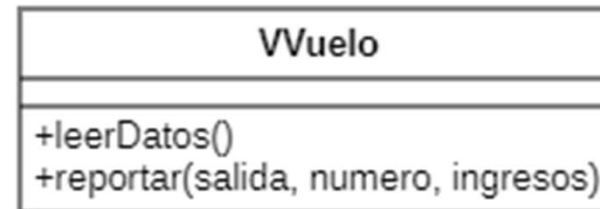
Patrón MVC: Diseño

Diagrama de Clases: **Vista**

La vista se encarga de “interactuar” con el usuario: leer datos y mostrar salidas

Ya que la clase menor tiene 1 requerimiento, entonces amerita el método reportar. Observa que los datos a reportar son lo que me piden

Similar la vista para la clase mayor: el reportar recibe los datos a mostrar.

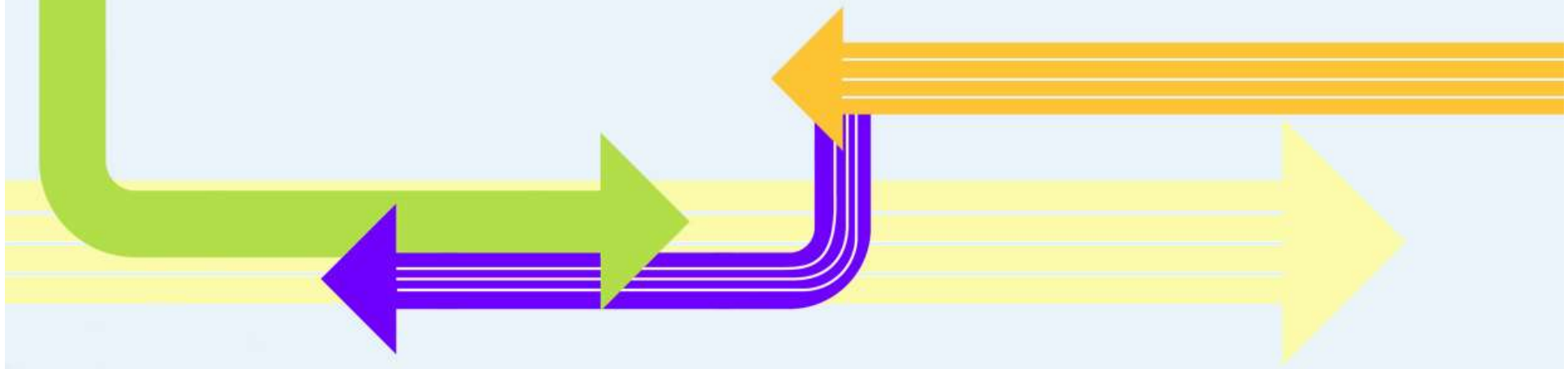
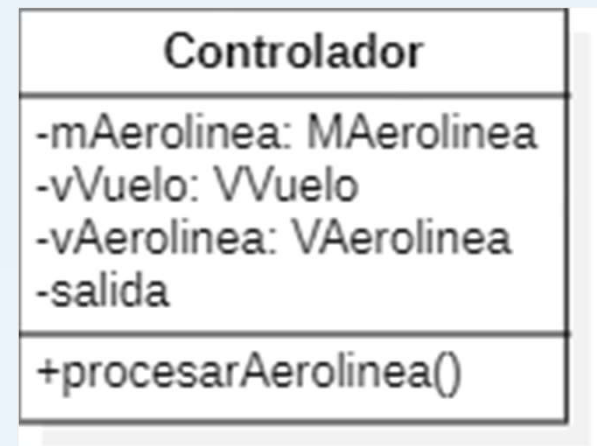


Patrón MVC: Diseño

Diagrama de Clases: **Controlador**

El controlador tiene como atributos las clases del modelo y de la vista (las necesarias).

El procesarAerolinea será el método que pone a funcionar tu programa: un ciclo hasta que se termine con la entrada de datos.



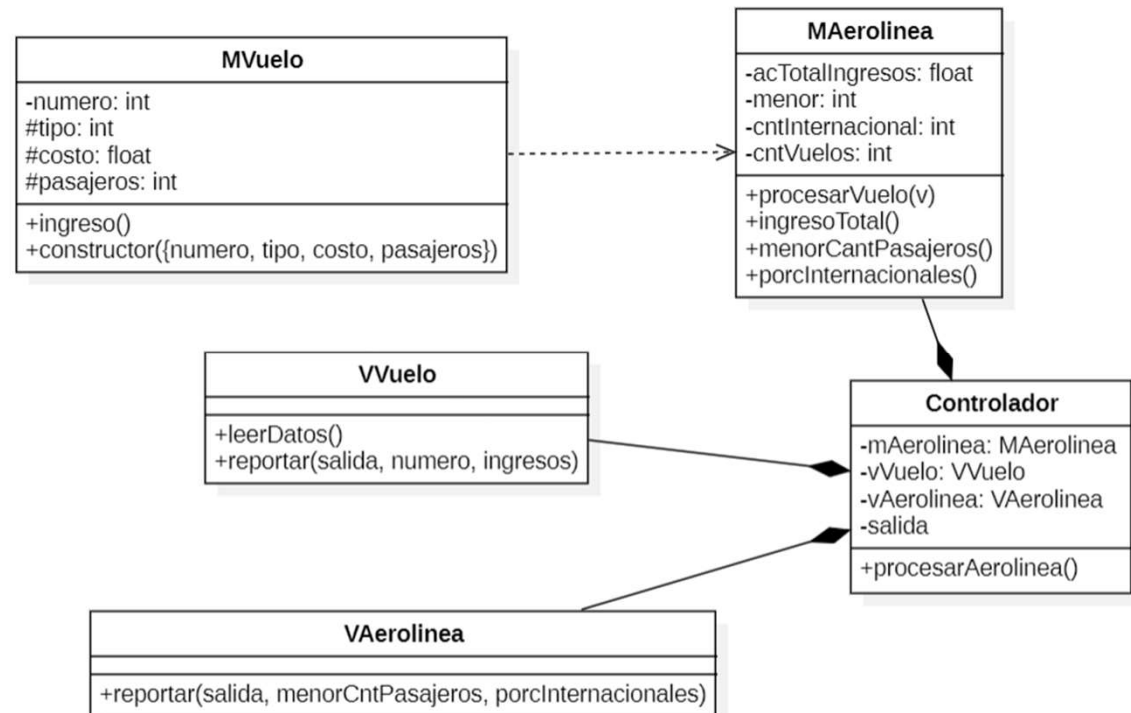
Solución 1: procesamiento básico

Patrón MVC: Diseño

Diagrama de Clases Completo con MVC

Todas las clases en el proyecto se relacionan. La clase MAerolinea **USA** la clase MVuelo, mientras que el controlador tienen como **COMPONENTES** las clases que ves como atributo

REGLA IMPORTANTE:
Diseña cuidadosamente tu diagrama de clases, será tu guía durante la implementación (es decir, durante **la programación**)



Solución 1: procesamiento básico

Patrón MVC: Diseño

El objeto MVuelo se construye enviando un objeto de datos (los datos entre comillas)

Acá la relación es de USO (se pasa el objeto ya creado)

Los prefijos ac y cnt son buena práctica para estos atributos

Los métodos se llaman TAL como lo dice el planteamiento, no importa si es largo el identificador

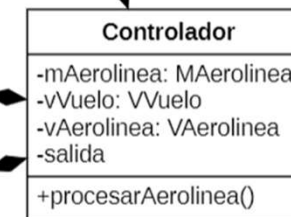
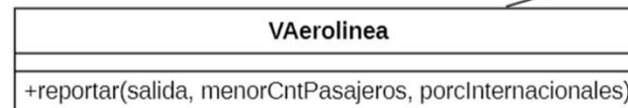
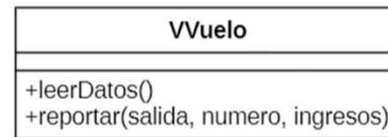
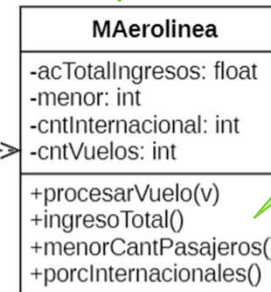
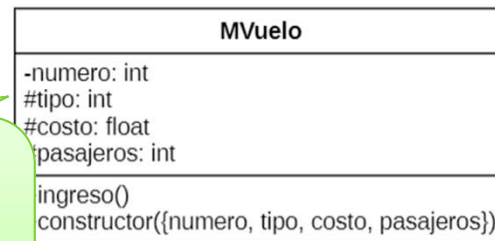
Elige los nombres de atributos y métodos los más acorde al planteamiento: te ayudará a trabajar más coherentemente

Ej.: en este caso, en cualquier parte del proyecto, los componentes de un vuelo se llaman: numero, tipo, costo y pasajeros

En los métodos de clases de la vista se debe colocar cada dato a reportar. NO PASAR objetos del modelo.

No tienes mayor material nuevo acá: el modelo ya lo habías visto en el 1er corte

El diagrama de clases es un instrumento que el programador debe tener a la mano para que la implementación sea más fluida.



En este caso la clase MVuelo no es componente ya que se crea en el procesar

Observa la línea de relación tipo COMPONENTE... Indica que es un atributo par la clase

Solución 1: procesamiento básico

Patrón MVC: Modelo en JavaScript

El MODELO fue lo
aprendimos en el
1er corte:
ES EXACTO IGUAL

```
export class Cl_mVuelo {  
  constructor({ numero, tipo, costo, pasajeros }) {  
    this.numero = numero;  
    this.tipo = tipo;  
    this.costo = costo;  
    this.pasajeros = pasajeros;  
  }  
  set tipo(t) {  
    this._tipo = +t;  
  }  
  get tipo() {  
    return this._tipo;  
  }  
  set pasajeros(p) {  
    this._pasajeros = +p;  
  }  
  get pasajeros() {  
    return this._pasajeros;  
  }  
  set costo(c) {  
    this._costo = +c;  
  }  
  get costo() {  
    return this._costo;  
  }  
  ingreso() {  
    return this.costo * this.pasajeros;  
  }  
}
```

Solución 1: procesamiento básico

Patrón MVC: Modelo en JavaScript

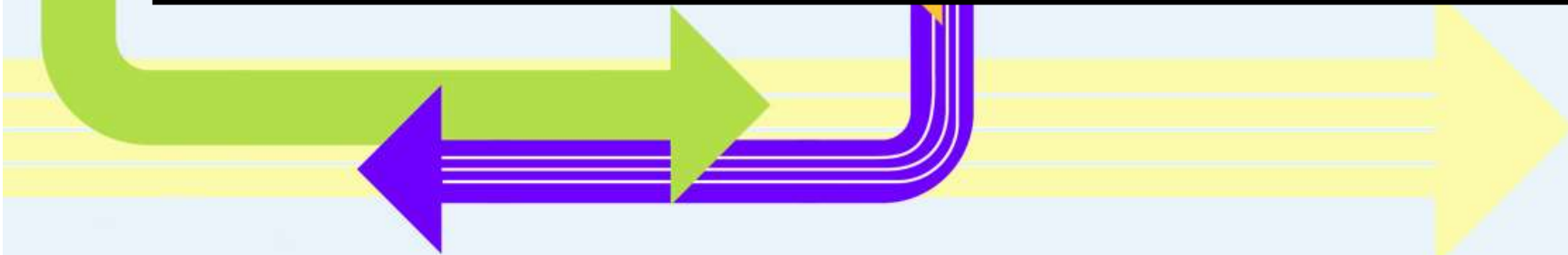
```
export class Cl_mAerolinea {  
  constructor() {  
    this.acTotalIngresos = 0;  
    this.menor = Infinity;  
    this.cntInternacional = 0;  
    this.cntVuelos = 0;  
  }  
  procesarVuelo(vuelo) {  
    this.acTotalIngresos += vuelo.ingreso();  
    if (vuelo.pasajeros < this.menor) this.menor = vuelo.pasajeros;  
    if (vuelo.tipo === 2) this.cntInternacional++;  
    this.cntVuelos++;  
  }  
  ingresoTotal() {  
    return this.acTotalIngresos;  
  }  
  menorCantPasajeros() {  
    return this.menor;  
  }  
  porcInternacionales() {  
    return (this.cntInternacional / this.cntVuelos) * 100;  
  }  
}
```

El MODELO fue lo
aprendimos en el
1er corte:
ES EXACTO IGUAL

Patrón MVC: Vista en JavaScript

```
export class Cl_vVuelo {  
  leerDatos() {  
    let numero = prompt("numero");  
    let tipo = prompt("tipo");  
    let costo = prompt("costo");  
    let pasajeros = prompt("pasajeros");  
    return { numero, tipo, costo, pasajeros };  
  }  
  reportar(salida, numero, ingreso) {  
    salida.innerHTML += `  
El vuelo ${numero} tiene  
un ingreso de ${ingreso}`;  
  }  
}
```

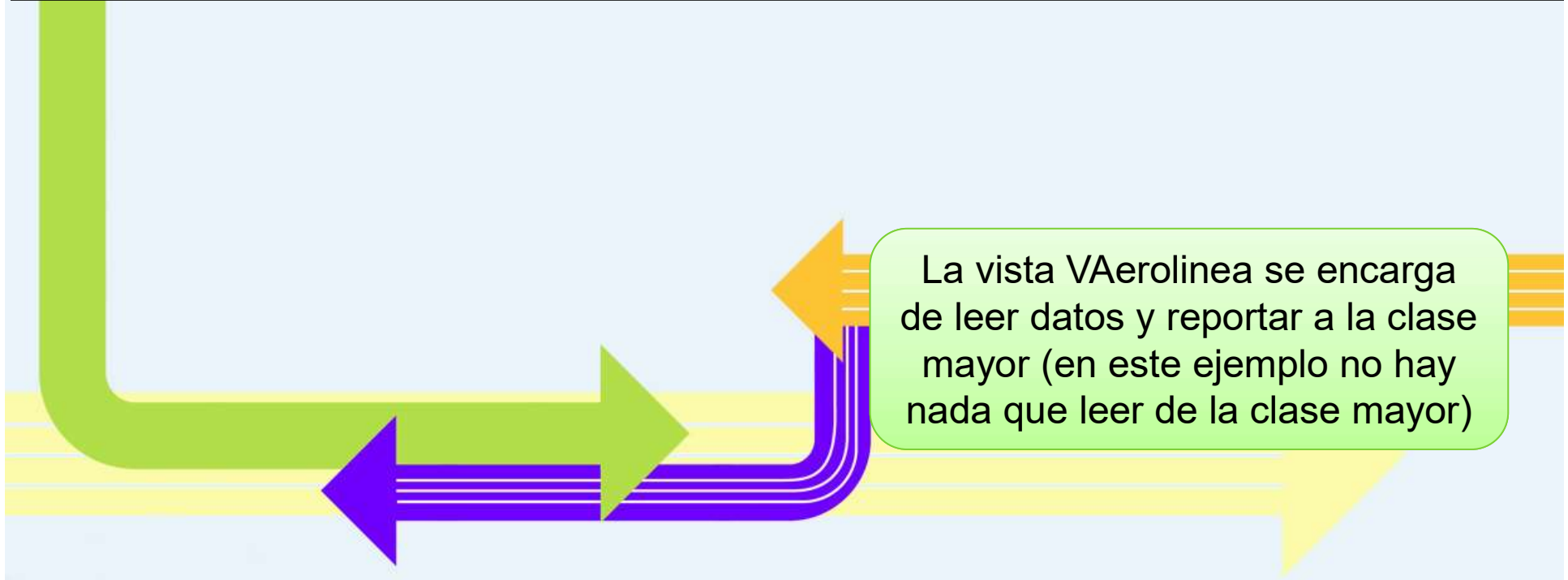
La vista VVuelo se encarga de leer datos y reportar a la clase menor



Solución 1: procesamiento básico

Patrón MVC: Vista en JavaScript

```
export class Cl_vAerolinea {  
  reportar(salida, ingTotal, menorCntPasajeros, porcInternacionales) {  
    salida.innerHTML += `<br>Total de ingresos: ${ingTotal}`;  
    salida.innerHTML += `<br>Menor cantidad de pasajeros: ${menorCntPasajeros}`;  
    salida.innerHTML += `<br>Porcentaje de vuelos internacionales: ${porcInternacionales}%`;  
  }  
}
```



La vista VAerolinea se encarga de leer datos y reportar a la clase mayor (en este ejemplo no hay nada que leer de la clase mayor)

Solución 1: procesamiento básico

Patrón MVC: Controlador en JavaScript

```
import { Cl_vAerolinea } from "./Cl_vAerolinea.js";
import { Cl_vVuelo } from "./Cl_vVuelo.js";
import { Cl_mVuelo } from "./Cl_mVuelo.js";
import { Cl_mAerolinea } from "./Cl_mAerolinea.js";
export class Controlador {
  constructor() {
    this.Cl_mAerolinea = new Cl_mAerolinea();
    this.Cl_vVuelo = new Cl_vVuelo();
    this.Cl_vAerolinea = new Cl_vAerolinea();
    this.salida = document.getElementById("salida");
  }
  procesarAerolinea() {
    do {
      let datos = this.Cl_vVuelo.leerDatos(),
          vuelo = new Cl_mVuelo(datos);
      this.Cl_mAerolinea.procesarVuelo(vuelo);
      this.Cl_vVuelo.reportar(this.salida, vuelo.numero, vuelo.ingreso());
    } while (confirm("¿Hay otro vuelo?"));
    this.Cl_vAerolinea.reportar(
      this.salida,
      this.Cl_mAerolinea.ingresoTotal(),
      this.Cl_mAerolinea.menorCantPasajeros(),
      this.Cl_mAerolinea.porcInternacionales()
    );
  }
}
```

El controlador crea atributos de las clases (en el constructor).

Luego hace un ciclo para procesar los objetos de la clase menor.

Cada vez que procesa un objeto, lo reporta (para cada objeto menor).

Al final reporta el objeto mayor

Solución 1: procesamiento básico

Patrón MVC: Programa Principal

```
import { Controlador } from "../Cl_controlador.js";
export class Cl_main {
  constructor() {
    let control = new Controlador();
    control.procesarAerolinea();
  }
}
```

El programa principal
“dispara” el procesamiento
por parte del controlador.

Principal



Solución 1: procesamiento básico

Archivo HTML

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>Proyecto Aerolínea</title>
  </head>
  <body>
    <div id="salida"></div>
    <script type="module">
      import { Cl_main } from "../js/Cl_main.js";
      new Cl_main();
    </script>
  </body>
</html>
```

El index.html, en este caso,
es tal como lo conoces.

Solución 1: procesamiento básico

Salida

El vuelo 111 tiene un ingreso de 2400
El vuelo 222 tiene un ingreso de 600
El vuelo 333 tiene un ingreso de 3000
El vuelo 444 tiene un ingreso de 1200
Total de ingresos: 7200
Menor cantidad de pasajeros: 60
Porcentaje de vuelos internacionales: 75%

La salida para la solución 1: **RUDIMENTARIA**

Solución 2: Integración con HTML y CSS

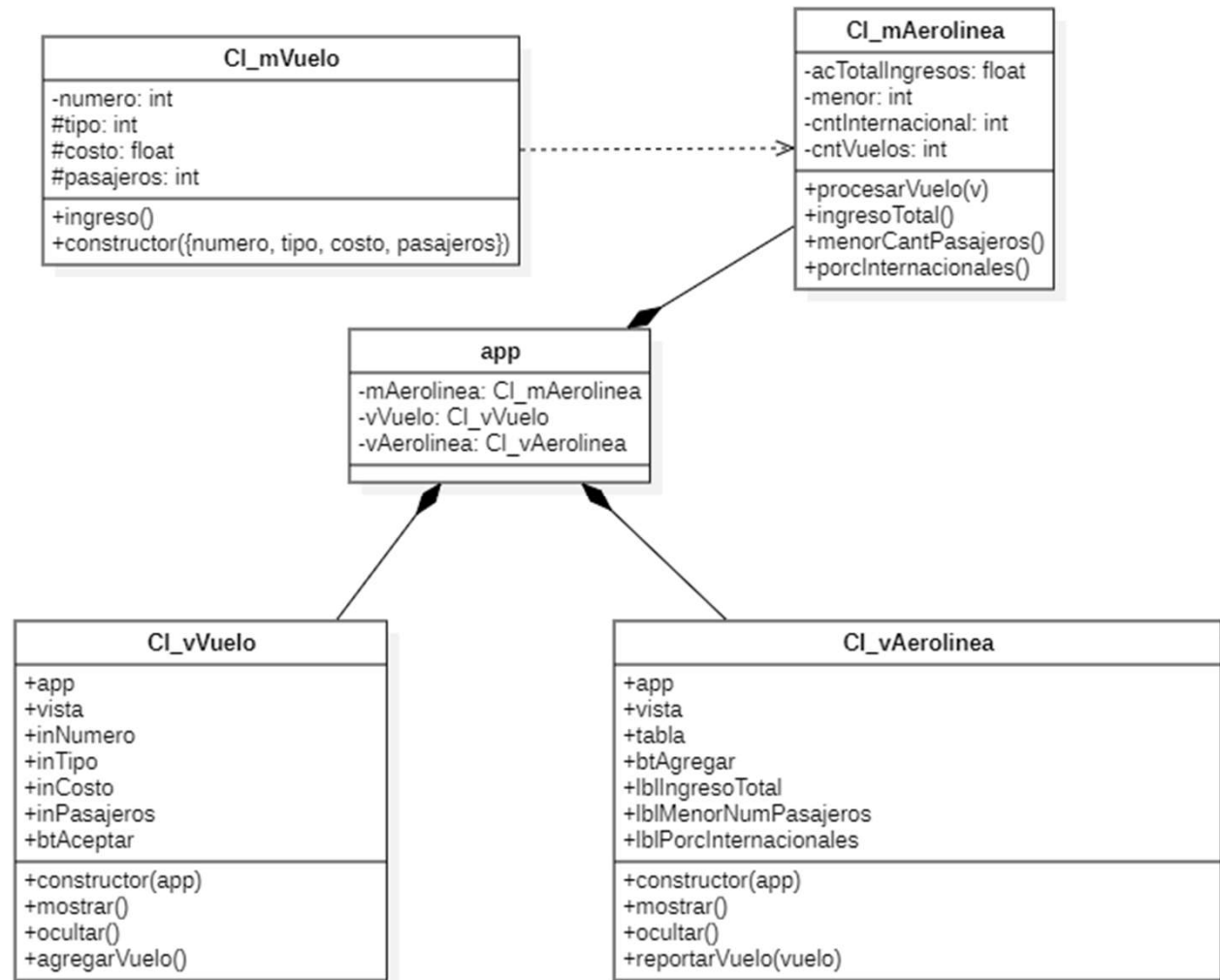
Patrón MVC: Diseño

Diagrama de Clases Completo con MVC

Haremos ahora una vista más parecida a la realidad... Usando HTML y sólo un poco de CSS

Ten en cuenta que hay cientos de formas de hacer la vista con HTML

Hemos procurado una manera elemental de hacerlo



Solución 2: Integración con HTML y CSS

Patrón MVC: Modelo en JavaScript

```
export class Cl_mVuelo {
  constructor({ numero, tipo, costo, pasajeros }) {
    this.numero = numero;
    this.tipo = tipo;
    this.costo = costo;
    this.pasajeros = pasajeros;
  }
  set tipo(t) {
    this._tipo = +t;
  }
  get tipo() {
    return this._tipo;
  }
  set pasajeros(p) {
    this._pasajeros = +p;
  }
  get pasajeros() {
    return this._pasajeros;
  }
  set costo(c) {
    this._costo = +c;
  }
  get costo() {
    return this._costo;
  }
  ingreso() {
    return this.costo * this.pasajeros;
  }
}
```

**El mismo modelo
que en Solución 1**

```
export class Cl_mAerolinea {
  constructor() {
    this.acTotalIngresos = 0;
    this.menor = Infinity;
    this.cntInternacional = 0;
    this.cntVuelos = 0;
  }
  procesarVuelo(vuelo) {
    this.acTotalIngresos += vuelo.ingreso();
    if (vuelo.pasajeros < this.menor) this.menor = vuelo.pasajeros;
    if (vuelo.tipo === 2) this.cntInternacional++;
    this.cntVuelos++;
  }
  ingresoTotal() {
    return this.acTotalIngresos;
  }
  menorCantPasajeros() {
    return this.menor;
  }
  porcInternacionales() {
    return (this.cntInternacional / this.cntVuelos) * 100;
  }
}
```

Nuestra vista tendrá 2 secciones: una tabla mostrando los elementos procesados, con el resumen de la salida en la parte de abajo. También hay un botón para agregar un nuevo objeto menor

Habrà otra sección para leer los datos del objeto menor.

Las secciones se ocultan / muestran convenientemente para que la aplicación funcione de forma correcta.

Lo lograremos con un poco de HTML: **usando tablas e inputs** (los inputs son elementos para entradas de datos)

También usaremos EVENTOS, tales como HACER CLICK y ejecutar un método de la vista

Aprenderemos un poco de Programación Orientada a Eventos, donde un evento es: construir un objeto, presionar un botón, etc.

Cuando se dispara un evento, invocamos métodos de los objetos.

Aerolínea SUPERVUELOS

Número	Tipo	Costo	Pasajeros	Ingreso
--------	------	-------	-----------	---------

Ingreso Total del día: Bs.?

Menor Número de Pasajeros en un vuelo: ? pasajeros

Porcentaje de vuelos internacionales: ?%

Agregar vuelo

Datos del vuelo

Número

Tipo

Costo

Pasajeros

Aceptar

Solución 2: Integración con HTML y CSS

Patrón MVC: Archivo HTML

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Aerolínea SUPERVUELOS</title>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <section id="mainForm">
      <h3>Aerolínea SUPERVUELOS</h3>
      <table id="mainForm_tabla">
        <tr>
          <th>Número</th>
          <th>Tipo</th>
          <th>Costo</th>
          <th>Pasajeros</th>
          <th>Ingreso</th>
        </tr>
      </table>
      <h5>Ingreso Total del día: Bs.<span id="mainForm_lblIngresoTotal">?</span></h5>
      <h5>Menor Número de Pasajeros en un vuelo: <span id="mainForm_lblMenorNumPasajeros">?</span> pasajeros</h5>
      <h5>Porcentaje de vuelos internacionales: <span id="mainForm_lblPorcInternacionales">?</span>%</h5>
      <button id="mainForm_btAgregar">Agregar vuelo</button>
    </section>
```

El HTML se usa para DIBUJAR cómo se verá la vista. Todo usando elementos HTML.

- *1 Sección Form principal
 - ✓ Título
 - ✓ Tabla
 - ✓ 3 salidas (requerimientos)
 - ✓ Botón agregar vuelo

Observa los distintos tipos de elementos HTML... Observa que se aplican para lograr el diseño que vimos antes.

Ver bibliografía para más detalles, al final de esta presentación

Solución 2: Integración con HTML y CSS

Patrón MVC: Archivo HTML

*2

```
<section id="vueloForm">
  <h3>Datos del vuelo</h3>
  <input type="text" id="vueloForm_inNumero" placeholder="Número"></input>
  <input type="text" id="vueloForm_inTipo" placeholder="Tipo"></input>
  <input type="text" id="vueloForm_inCosto" placeholder="Costo"></input>
  <input type="text" id="vueloForm_inPasajeros" placeholder="Pasajeros"></input>
  <button id="vueloFormBtAceptar">Aceptar</button>
</section>
```

*3

```
<script type="module">
  import {Cl_main} from "../js/Cl_main.js";
  new Cl_main()
</script>
</html>
```

Los elementos de tipo **section** se usan para poder ocultar / mostrar las formas (recuerda Google Form, es algo similar).

La clase **VVuelo** trabaja con los elementos HTML en la **section vueloForm**

La clase **VAerolinea** trabaja con los elementos HTML en la **section mainForm**

*2 Sección Form vuelo

- ✓ Título
- ✓ Inputs
- ✓ Botón Aceptar

*3 Inclusión de JavaScript (tipo módulo)

Solución 2: Integración con HTML y CSS

Patrón MVC: Vista en JavaScript

```
import { MVuelo } from "../Cl_mVuelo.js";
```

```
export class VVuelo {
```

```
  constructor(app) {
```

```
    this.app = app;
```

```
    this.vista = document.getElementById("vueloForm");
```

```
    this.vista.hidden = true;
```

```
    this.inNumero = document.getElementById("vueloForm_inNumero");
```

```
    this.inTipo = document.getElementById("vueloForm_inTipo");
```

```
    this.inCosto = document.getElementById("vueloForm_inCosto");
```

```
    this.inPasajeros = document.getElementById("vueloForm_inPasajeros");
```

```
    this.btAceptar = document.getElementById("vueloForm_btAceptar");
```

```
    this.btAceptar.onclick = () => this.agregarVuelo();
```

```
  }
```

```
  mostrar() {
```

```
    this.vista.hidden = false;
```

```
  }
```

```
  ocultar() {
```

```
    this.vista.hidden = true;
```

```
  }
```

```
  agregarVuelo() {
```

```
    let vuelo = new MVuelo({
```

```
      numero: this.inNumero.value,
```

```
      tipo: this.inTipo.value,
```

```
      costo: this.inCosto.value,
```

```
      pasajeros: this.inPasajeros.value,
```

```
    });
```

```
    this.app.mAerolinea.procesarVuelo(vuelo);
```

```
    this.app.vAerolinea.reportarVuelo(vuelo);
```

```
    this.ocultar();
```

```
    this.app.vAerolinea.mostrar();
```

```
  }
```

```
}
```

Acceso a la APP

Elemento vueloForm

Elementos de entrada

Evento OnClick BtAceptar

Ocultar / Mostrar vueloForm

Método agregarVuelo

Instanciar desde elementos de entrada

Procesar en Modelo
Reportar en Vista

Ocultar vueloForm
Mostrar mainForm (acceso a APP)

Solución 2: Integración con HTML y CSS

Patrón MVC: Vista en JavaScript

Elementos de salida

Evento
OnClick
BtAgregar

```
export class VAerolinea {  
  constructor(app) {  
    this.app = app;  
    this.vista = document.getElementById("mainForm");  
    this.tabla = document.getElementById("mainFormTabla");  
    this.btAgregar = document.getElementById("mainFormBtAgregar");  
    this.lblIngresoTotal  
      = document.getElementById("mainFormLblIngresoTotal");  
    this.lblMenorNumPasajeros  
      = document.getElementById("mainFormLblMenorNumPasajeros");  
    this.lblPorcInternacionales  
      = document.getElementById("mainFormLblPorcInternacionales");  
    this.btAgregar.onclick = () => {  
      this.ocultar();  
      this.app.vVuelo.mostrar();  
    };  
  }  
}
```

Constructor

Ocultar / Mostrar mainForm

Agregar fila a la tabla

Reporte de requerimientos

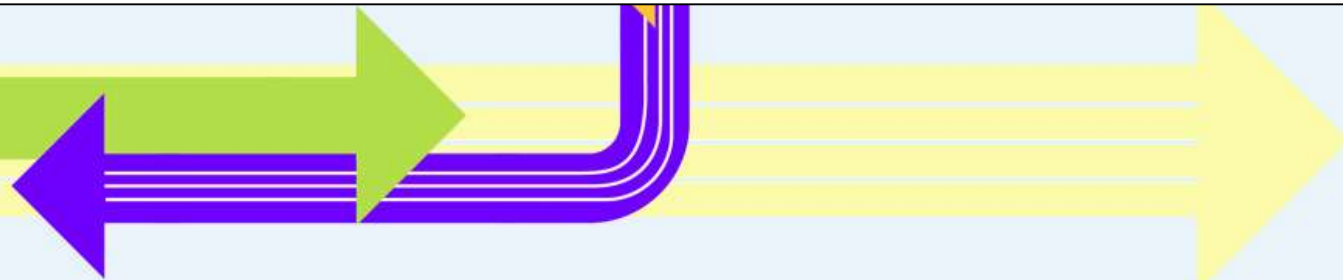
```
  mostrar() {  
    this.vista.hidden = false;  
  }  
  ocultar() {  
    this.vista.hidden = true;  
  }  
  reportarVuelo(vuelo) {  
    this.tabla.innerHTML += `  
    <tr>  
      <td>${vuelo.numero}</td>  
      <td>${vuelo.tipo}</td>  
      <td>${vuelo.costos}</td>  
      <td>${vuelo.pasajeros}</td>  
      <td>${vuelo.ingreso()}</td>  
    </tr>`;  
    this.lblIngresoTotal.innerHTML  
      = this.app.mAerolinea.ingresoTotal();  
    this.lblMenorNumPasajeros.innerHTML  
      = this.app.mAerolinea.menorCantPasajeros();  
    this.lblPorcInternacionales.innerHTML  
      = this.app.mAerolinea.porcInternacionales();  
  }  
}
```

Método
reportarVuelo

Solución 2: Integración con HTML y CSS

Patrón MVC: Programa Principal - APP

```
import { VAerolinea } from "./Cl_vAerolinea.js";
import { VVuelo } from "./Cl_vVuelo.js";
import { MAerolinea } from "./Cl_mAerolinea.js";
class Cl_app {
  constructor() {
    this.mAerolinea = new MAerolinea();
    this.vVuelo = new VVuelo(this);
    this.vAerolinea = new VAerolinea(this);
  }
}
export default Cl_app
```



Solución 2: Integración con HTML y CSS

Salida

Aerolínea SUPERVUELOS

Número	Tipo	Costo	Pasajeros	Ingreso
111	2	20	120	2400
222	1	10	60	600
333	2	30	100	3000
444	2	15	80	1200

Ingreso Total del día: Bs.7200

Menor Número de Pasajeros en un vuelo: 60 pasajeros

Porcentaje de vuelos internacionales: 75%

Agregar vuelo

Datos del vuelo

444

2

15

80

Aceptar

Bibliografía

Tablas

- https://www.w3schools.com/html/html_tables.asp

Formas

- https://www.w3schools.com/html/html_forms.asp

Módulos en JavaScript

- https://www.w3schools.com/js/js_modules.asp

Esta Presentación

- <https://github.com/g-torrealba-ucla/L24.1.p.c2.00-Contenido.git>

GitHub de Aerolínea v.1

- <https://github.com/g-torrealba-ucla/L24.1.p.c2.01-Aerolinea.git>

GitHub de Aerolínea v.2

- <https://github.com/g-torrealba-ucla/L24.1.p.c2.02-Aerolinea.git>

Patrón de Diseño MVC

Ventajas

- La separación del Modelo de la Vista
- Es mucho más sencillo agregar múltiples representaciones de los mismos datos o información.
- Crea independencia de funcionamiento.
- Facilita el mantenimiento en caso de errores.
- Facilita agregar nuevos tipos de datos

Desventajas

- La separación de conceptos en capas agrega complejidad al sistema.
- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.

