

0101010
0100101
1101010

UD10.1 - ARQUITECTURA DE NIVELES

Programació – 1er DAW/DAM

0. CONTENIDOS

- . Introducción
- . Arquitectura de capas
 - Presentación
 - Lógica de negocio
 - Acceso a datos
 - Base de datos
- . Patrón MVC
 - Modelo
 - Vista
 - Controlador

1. Introducción

• Problema:

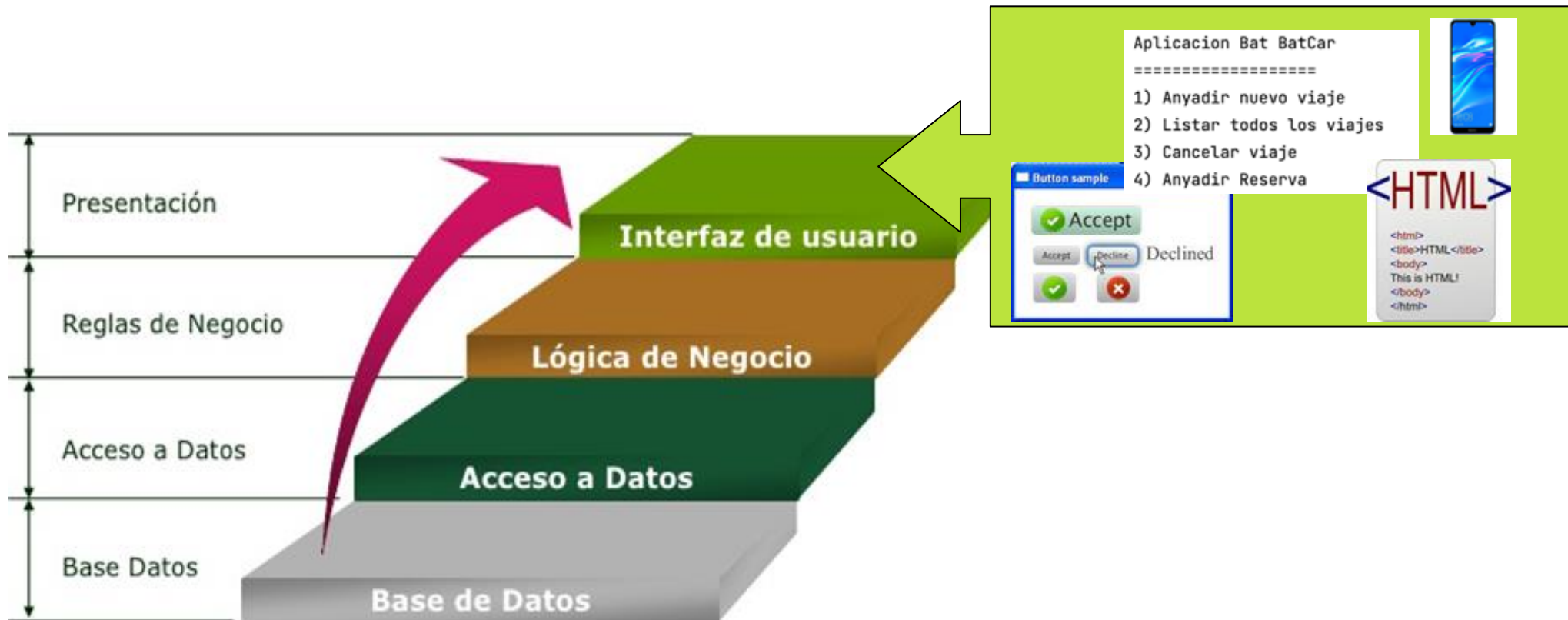
- Las aplicaciones actuales presentan un alto **grado de complejidad**.
- Se requiere de **patrones** o **arquitecturas** comunes, **bien probadas**, que representen **buenas prácticas** para la creación de software.
- Base de **conocimiento común** que favorece el entendimiento.



2. Arquitectura de capas

- La aplicación se arquitectura en **diferentes capas** cada una de las cuales tendrá asignada una **responsabilidad**
 - **Interfaz de usuario:** Representa la parte visible de la aplicación.
 - **Lógica de negocio:** Representa las diferentes reglas que deben regir el funcionamiento de la aplicación.
 - *¿Cómo conseguir puntos en un juego? ¿Qué debo hacer para pasar al nivel siguiente? ¿Cómo hacer una venta / devolución de un videojuego?*
 - **Acceso a datos:** *Constituye la parte del software que me permite hacer persistente los datos entre diferentes ejecuciones.*
 - *Almacenamiento de los datos en **ficheros locales**, en una **base de datos**,...*

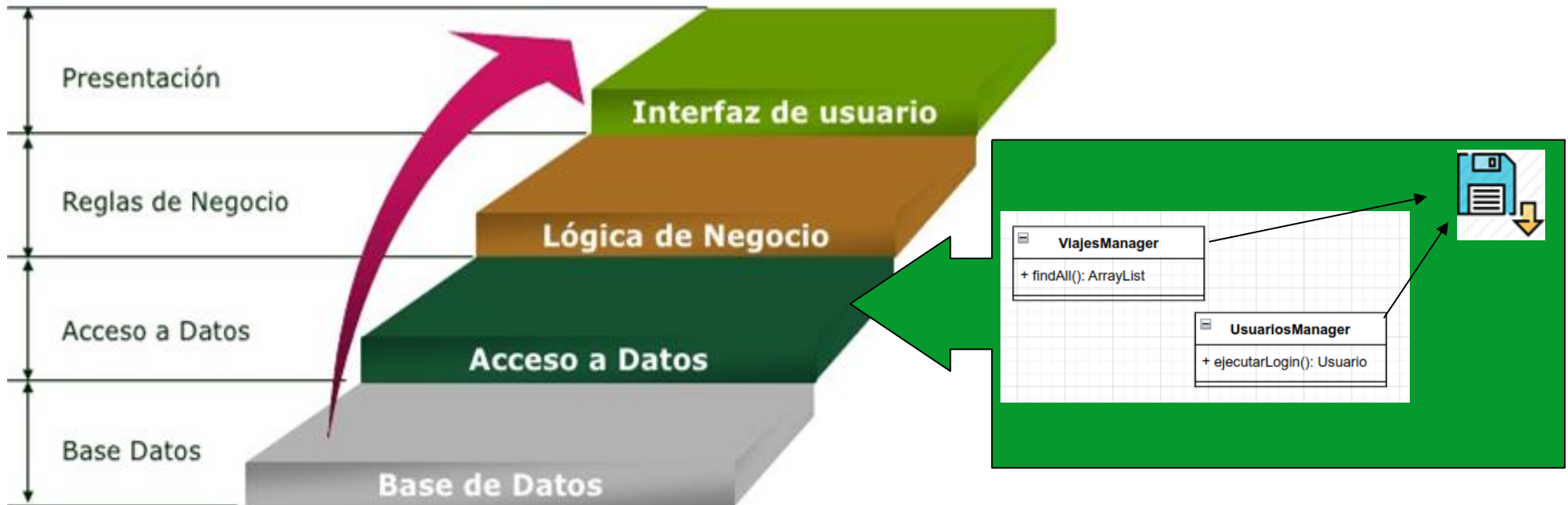
2.1. Capa de presentación



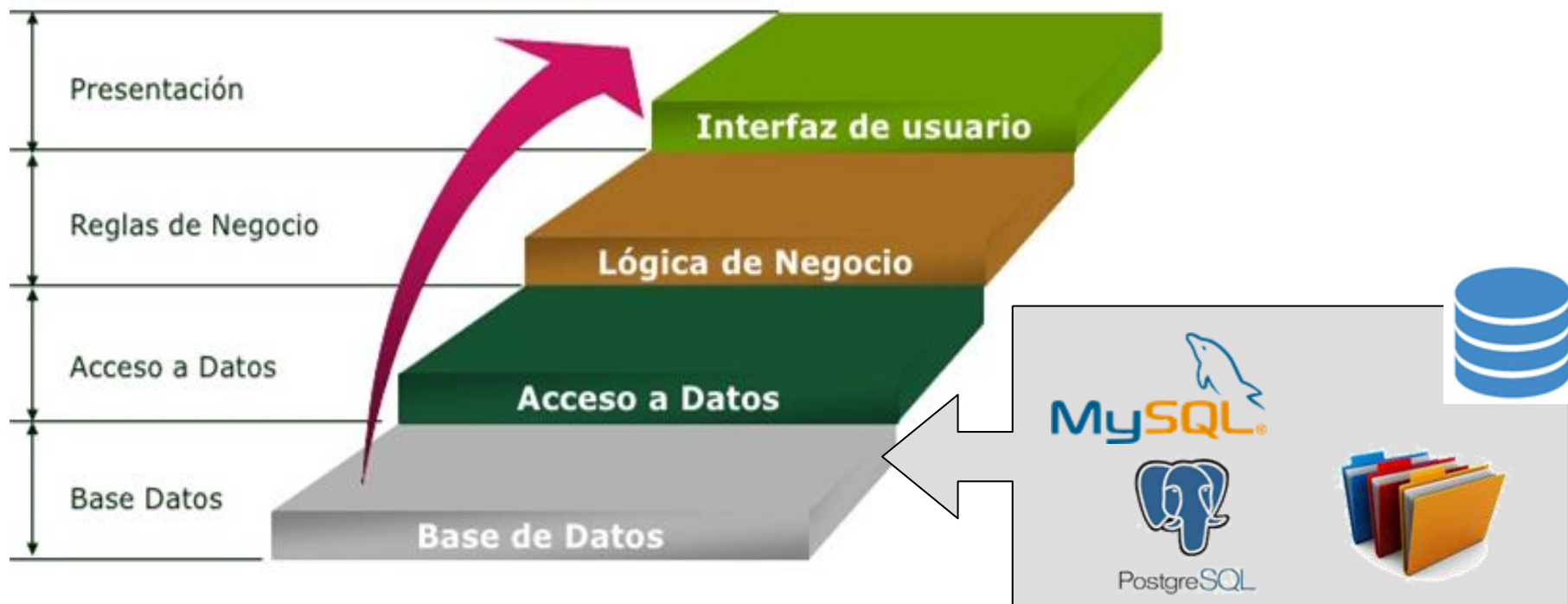
2.2. Capa Lógica de Negocio



2.3. Capa de Acceso a Datos



2.4. Capa de Base de Datos



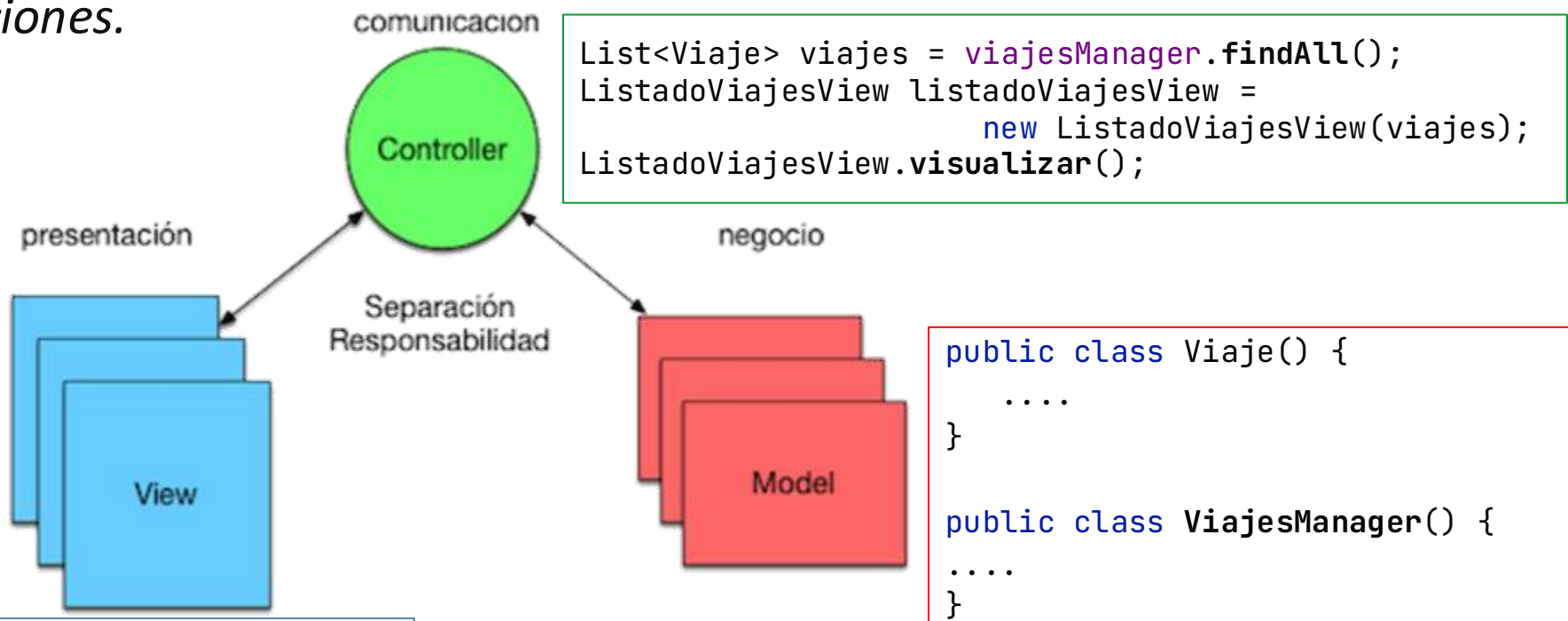
2.5 Ventajas

- Al crear una aplicación **mediante capas**, se aplica el principio de **"divide y vencerás"**
 - **Independencia** de la aplicación respecto a cómo el **usuario interactúa con ella. (Vista)**
 - **Independencia** respecto a cómo los datos son **almacenados. (Acceso a datos)**
 - Aumento de la **reusabilidad** de cada una de las capas creadas.
 - Permite **cambios** en cada una de las capas sin afectar al resto



3. Patrón MVC

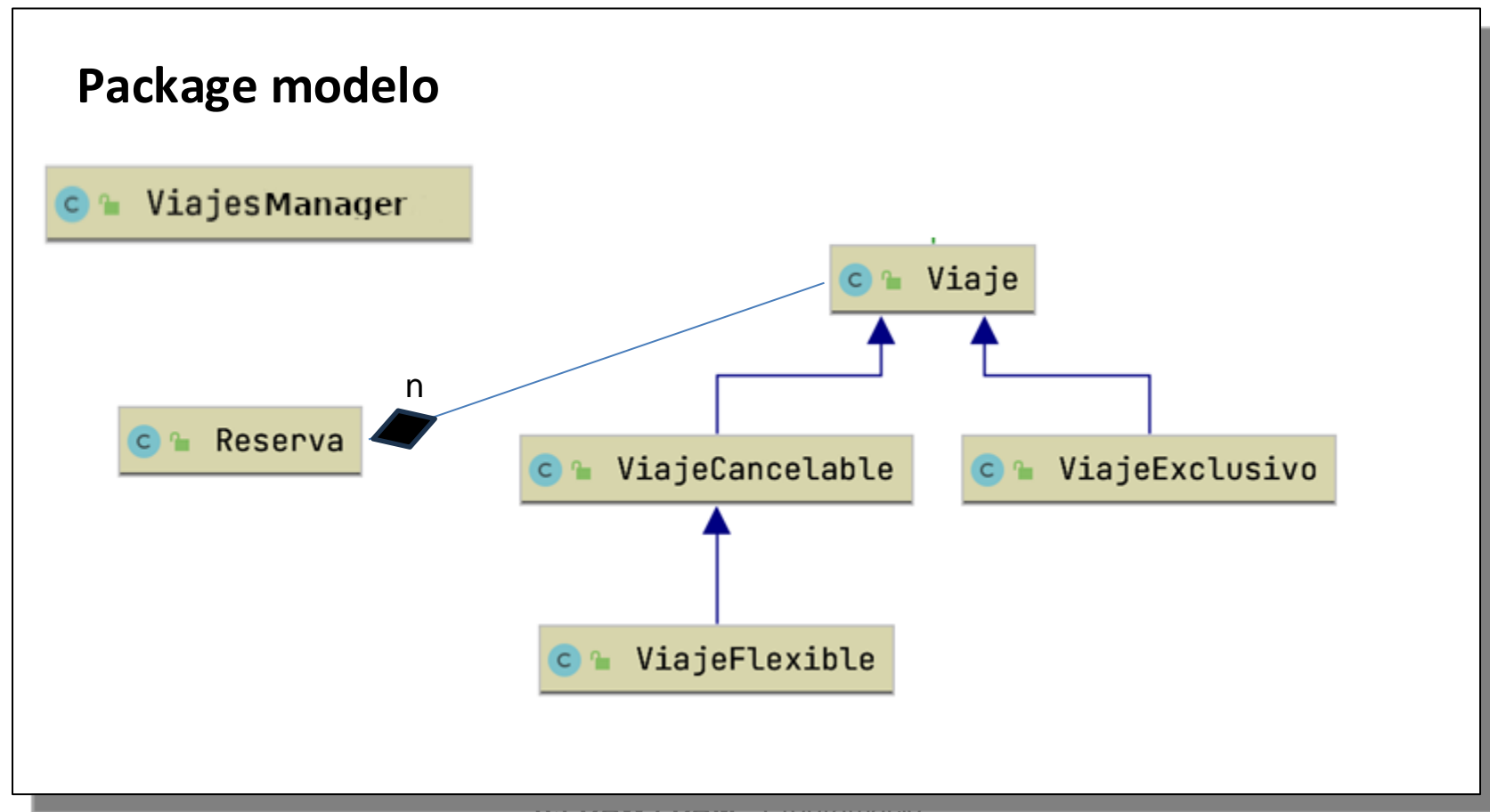
- Representa un **modelo** o **guía** para la implementación de aplicaciones siguiendo una **arquitectura de capas**.
- Su nombre **MVC** parte de las iniciales *Modelo Vista Controlador* que son los **grupos de componentes o capas** en las que organizaremos nuestras aplicaciones.



```
public class ListadoViajesView {
    ....
}
```

3.1. Modelo

- En la **capa modelo** encontramos una representación de los datos que maneja la aplicación.
 - Ejemplo.- En la aplicación **BatBatCar** tendremos las clases `Viaje`, `ViajeCancelable`, `Reserva`, ...



3.1. Modelo

```
public class Viaje {  
    private int codViaje;  
    private String ruta;  
    private LocalDateTime fechaSalida;  
    private long duracion;  
    private float precio;  
    protected int plazasOfertadas;  
    protected boolean cerrado;  
    private List<Reserva> reservas;  
    ....  
}
```

```
public class Reserva {  
    private String codigoReserva;  
    private String usuario;  
    private int plazasSolicitadas;  
    private LocalDateTime fechaRealizacion;  
    public Reserva(String codReserva) {  
        this.codigoReserva = codReserva;  
    }  
}
```

3.1. Modelo

- Será el **responsable** de que el **sistema se encuentre** en un **estado consistente**:

- **No** haya **más plazas reservadas** en un **viaje** que **asientos** existan.
- **No** exista ningún viaje **sin coche asociado**.
- **No** se pueda **reservar** un **viaje** que ya **ha salido**.

** Nota. Algunas de estas condiciones también podrían ser abordadas a nivel de la lógica de negocio.*

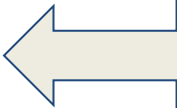
```
class Viaje {  
    ....  
    public void setPlazas(int plazas) {  
        if (plazasOfertadas > plazas) {  
            throw new PlazasCanNotBeReducedException();  
        }  
    }  
    ....  
}
```

}

3.1. Modelo

- Además, si la **aplicación** hace uso de una capa de **acceso a datos**, también encontraremos las clases que nos permiten **recuperar** y **guardar** los objetos.
 - En la aplicación `BatBatCar` utilizamos la clase `ViajesManager`

```
public class ViajesManager {  
    private List<Viaje> viajes;  
    public ViajesManager(){}  
  
    public void anyadir(Viaje viaje) {}  
  
    public Viaje getViaje(int codViaje) throws ViajeNotFoundException {}  
  
    public Viaje getViajeConReserva(String codReserva) throws ReservaNotFoundException {}  
  
    public List<Viaje> findAll(boolean disponibles) {}  
  
    public List<Viaje> findAll() { }  
  
}
```



Simulamos una **Base de datos en Memoria** haciendo uso de una colección.

3.2 Vista

- Es la responsable de generar la **interfaz de nuestra aplicación**, es decir, se encarga de **mostrar cualquier tipo de resultado visible e interactuar** con el usuario. Según el tipo de aplicación tenemos:
 - **Aplicación de consola:** La vista estará compuesta por los **flujos de entrada y salida** (Streams);

```
System.out.println("Hola Mundo");
```

```
int opcion = GestorIO.getInt(¿Quin tipus de viatge vol donar d'alta  
[1-normal, 2-cancel·lar, 3-exclusiu]?));
```

3.2 Vista

- **Aplicación web.** La vista estará compuesta por un conjunto de ficheros **html** y **css**: *tablas, párrafos, títulos, formularios,...*



A diagram consisting of two blue arrows. One arrow originates from the HTML code in the Notepad window and points to the rendered table. The other arrow originates from the same Notepad window and points to the form example below.

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

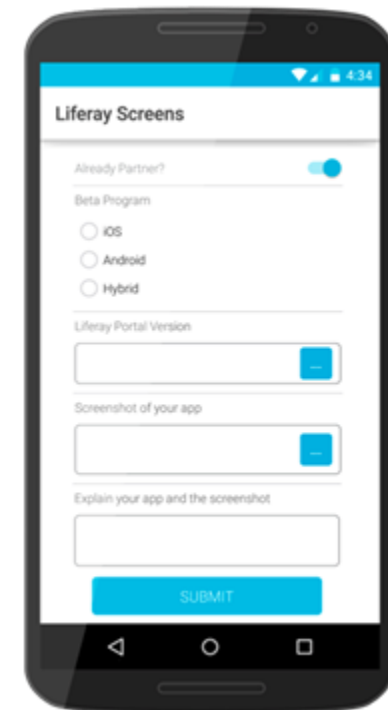
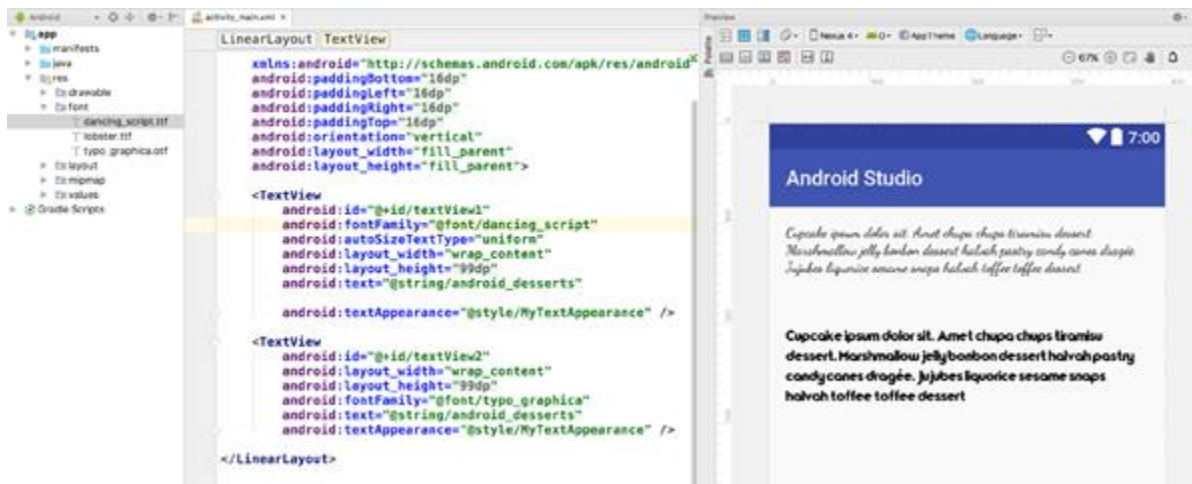
Example

First name:

Last name:

3.2 Vista

- **Aplicación de escritorio/móvil:** La vista estará compuesta por un conjunto de widgets o componentes visuales (listas, botones, labels) proporcionados por el framework de desarrollo y definidos, generalmente, con documentos declarativos como el XML.



3.2.1. Vista. Ejemplo

```
public class ListadoViajesView {

    private Set<Viaje> viajes;

    public ListadoViajesView(List<Viaje> viajes) {
        this.viajes = viajes;
    }

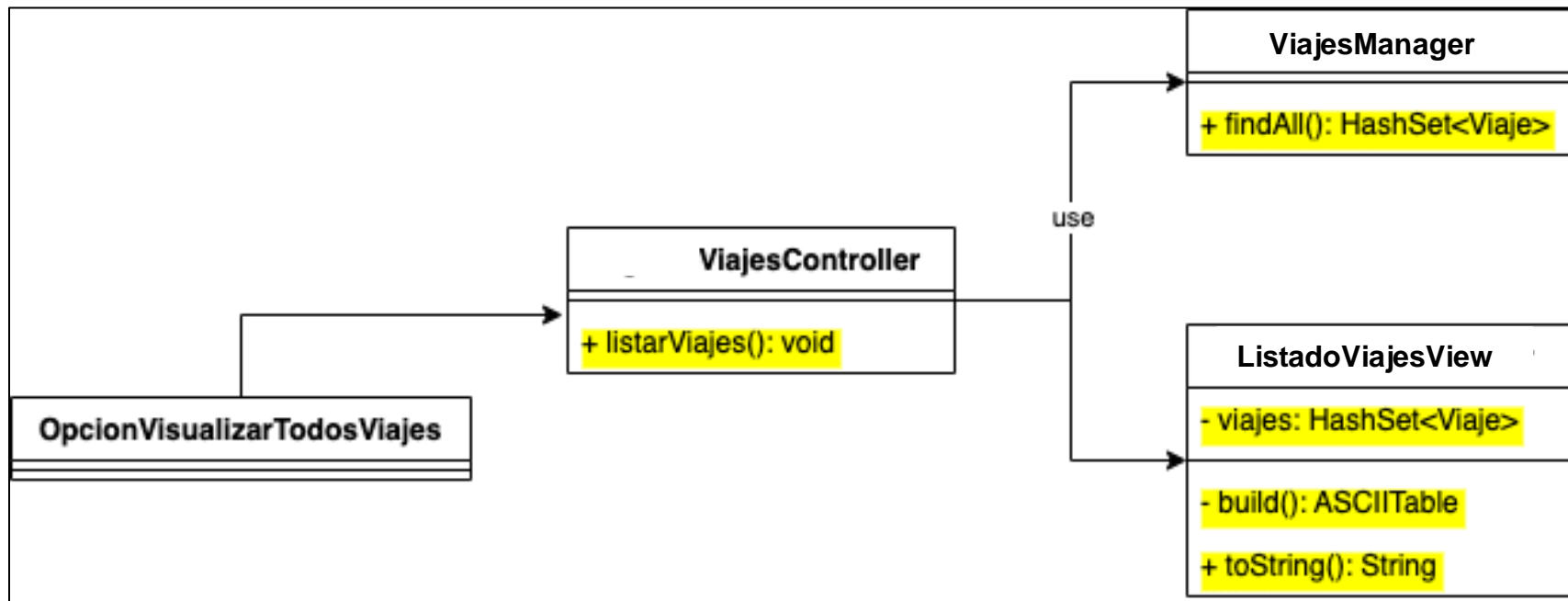
    private AsciiTable buildASCIITable() {
        AsciiTable view = new AsciiTable();
        view.setTextAlignment(TextAlignment.CENTER);
        view.addRule();
        AT_Row fila = view.addRow("*", "*", "*", "*", "*", "*", "*");
        fila.setTextAlignment(TextAlignment.CENTER);
        view.addRule();
        fila = view.addRow(null, null, null, null, null, null, "Listado Viajes");
        fila.setTextAlignment(TextAlignment.CENTER);
        view.addRule();
        fila = view.addRow("Cod. Viaje", null, "Ruta", "Precio", "Fecha Salida", "tipo", "Plazas disponibles");
        ...
        return view;
    }

    public void visualizar() {
        System.out.println(buildASCIITable().render(100));
    }
}
```

*	*	*	*	*	*	*
Listado Viajes						
Cod. Viaje	Ruta	Precio	Fecha Salida	tipo	Plazas disponibles	
1	Madrid-Murcia-Alicante	5,00	21-05-2022 a las 12:00	Viaje	1	
2	Alcoy-Ibi	5,00	20-01-2022 a las 11:00	Viaje	4	
3	Alicante-Valencia	5,00	25-02-2022 a las 05:00	Viaje Cancelable	4	

3.3 Controlador

- Su misión principal es actuar como **intermediario** entre el **usuario** y el **sistema**. Será el **responsable de coordinar la interacción** entre las clases que representan **el modelo y las vistas**.



3.3 Controlador. Ejemplo

```
public class ViajesController {

    private final ViajesManager viajesManager;

    public ViajesController() {
        this.viajesManager = new ViajesManager();
    }

    public void listarViajes() {
        Set<Viaje> viajes = viajesRepository.findAll();
        ListadoViajesView listadoViajesView = new ListadoViajesView(viajes);
        System.out.println(listadoViajesView);
    }

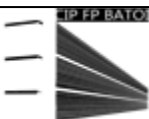
    ...
}
```

```
Aplicacion Bat BatCar
=====
1) Anyadir nuevo viaje
2) Listar todos los viajes
3) Cancelar viaje
4) Anyadir Reserva
5) Ver Reservas viaje
6) Cancelar reserva
7) Salir
Seleccione una opcion [1-7]
```

```
public class OpcionVisualizarTodosViajes extends Opcion {

    @Override
    public void ejecutar() {
        agenteViajesController.listarViajes();
    }

}
```



- Eso es todo... de momento :-)