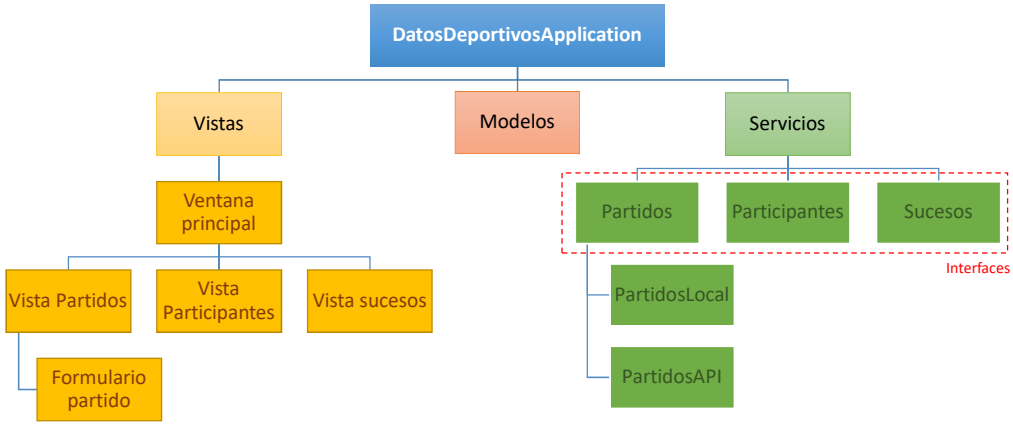
Práctica: Datos deportivos

- Arquitectura general



Práctica: Datos deportivos

- Gestión de dependencias (Vistas → Servicios):
  - Uso de Spring
  - Ventana principal → *@Component*
  - Servicios:
    - Declarados en clase de configuracion java
    - Las vistas secundarias obtienen los servicios a través del contexto de Spring (ApplicationContext)

Práctica: Datos deportivos

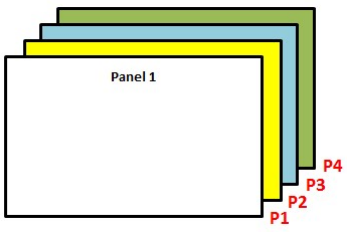
- Crear el proyecto con Spring Initializr
- Configurar arranque de la aplicación (modo no *headless*)
- Ventana principal
  - Menu
  - Cabecera
  - Barra de estado
- Implementar navegación
- Modelo de dominio
  - Clases Partido, Participante, Tarjeta y Gol

Ventana principal

CardLayout

Gestiona 2 o más componentes (normalmente paneles) que ocupan el mismo espacio. Solo se muestra uno al mismo tiempo. Se usan como alternativa a los paneles con pestañas, o para asistentes con varios pasos.

```
// Panel principal con layout tipo CardLayout
JPanel panelPrincipal = new JPanel(new CardLayout());

// Añadir un panel
// Ocupa todo el espacio
// Se identifica como "P1" dentro del CardLayout
panelPrincipal.add(new JPanel(), "P1");

// Añadir más paneles. Estos permanecen ocultos, bajo el panel 1
panelPrincipal.add(new JPanel(), "P2");
panelPrincipal.add(new JPanel(), "P3");
panelPrincipal.add(new JPanel(), "P4");

// Mostrar el panel 2, identificado como "P2"
// Ocultará el panel 1
CardLayout layout = (CardLayout) panelPrincipal.getLayout();
layout.show(panelPrincipal, "P2");
```



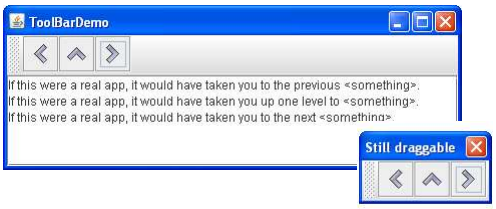
Práctica: Datos deportivos

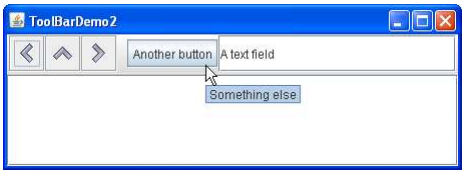
- Layout Partidos
  - Barra de herramientas con opciones "Crear", "Borrar" y "Editar"
  - Layout con SplitPane
  - Panel maestro:
    - Panel de filtro
    - Tabla de partidos

Layout Partidos

Barras de herramientas - JToolBar

- Contenedor de acciones relacionadas
- Botones u otros controles: checkbox, radiobuttons, combobox, cuadros de texto...
- Es "arrastrable" entre las zonas de un BorderLayout.





JToolBar

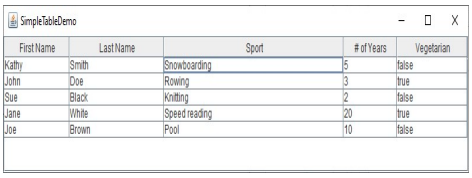
Contenedor orientado fundamentalmente a agrupar acciones mediante botones gráficos. También puede contener otro tipo de componentes, como combobox, checkbox, cuadros de texto...

```
// Constructor con título y orientación inicial
JToolBar barra =
    new JToolBar("Mi barra", HORIZONTAL); // o VERTICAL

// Añadir un botón gráfico y con "tooltip"
JButton boton = new JButton(new ImageIcon("imagen.png"));
boton.setToolTipText("Botón que hace algo");
boton.setActionCommand("hacer-algo");
boton.setActionListener(...);
barra.add(boton);

// Añadir un separador y un cuadro de texto
barra.addSeparator();
barra.add(new JTextField("Introduzca texto"));
```

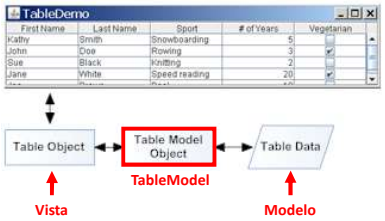
- Métodos
  - **add(Component)**: Añade componentes a la barra
  - **addSeparator()**: Añade un separador a la barra. El aspecto depende del L&F activo.
  - **setFloatable()**: Activa el arrastre de la barra fuera de la ventana.
- Listeners / Eventos:
  - No dispone de eventos o listeners específicos

Tablas (JTable)

- Muestra datos en formato tabular, con una cabecera.
- Cada columna permite un solo tipo de datos.
- Anchura y orden de las columnas modificables por el usuario.
- Dentro de un JSCrollPane, la cabecera se mantiene siempre visible.
- Permite selección simple, múltiple y no contigua.
- Permite la edición de las celdas. Se asigna un editor por defecto adecuado al tipo de dato básico.

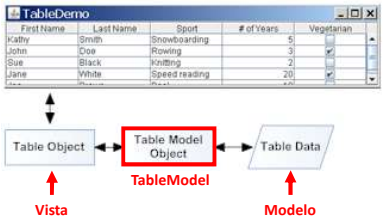


TableModel - Básico



- Un JTable no contiene datos propios, muestra datos de un modelo.
- Un **TableModel** asocia el modelo con la vista.
- Para tablas básicas (Vector o array bidimensional), se crea automáticamente un table model de tipo ***DefaultTableModel***.
- *DefaultTableModel* tiene métodos para agregar, modificar y eliminar datos en el array o Vector asociado.
- Tiene métodos para lanzar eventos al cambiar los datos del modelo. Los listeners son del tipo ***TableModelListener***.

TableModel - Avanzado



- Para usar modelos de datos más complejos, se debe crear un TableModel personalizado.
- Un TableModel debe implementar una interfaz muy simple:
  - *getRowCount()*
  - *getColCount()*
  - *getColumnName(numCol)*
  - *getColumnClass(numCol)*
  - *getValueAt(numFila, numCol)*
  - *setValueAt(numFila, numCol)*
  - *isCellEditable(numFila, numCol)*
- La clase *AbstractTableModel* provee una implementación básica que podemos extender y adaptar.
- Ejemplos: TableModel,s. que obtengan sus datos de una **base de datos** o de una **API REST**.
- Tiene métodos para lanzar eventos al cambiar los datos del modelo. Los listeners son del tipo ***TableModelListener***.

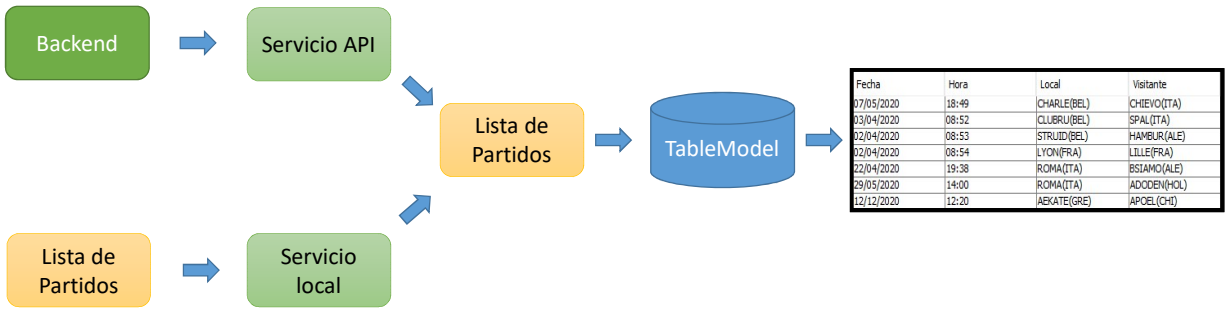
JTable / TableModel

JTable muestra y edita datos de un modelo en forma tabular. Admite como modelo arrays, vectores y cualquier objeto que implemente la interfaz *TableModel*.

```
// Crear una tabla a partir de un array bidimensional
private Object[][] datos ={
  {"Luis", "Perez", 23},
  {"Maria", "Lopez", 50},
  {"Carlos", "Muñoz", 34}
};
private String[] columnas = {"Nombre", "Apellidos", "Edad"};

JTable tabla = new JTable(datos, columnas);
```

- Métodos en JTable
  - **setSelectionMode()**: Modo de selección (simple, intervalo simple o intervalo multiple)
  - **setRowSelectionAllowed()**: Permite selecciona fila entera
  - **setColumnSelectionAllowed()**: Permite selecciona columna entera
  - **setCellSelectionEnabled()**: Permite selección por celdas o bloques de celdas.
  - **getSelectedRow() / getSelectedColumn()**: Fila o columna seleccionada.
- Eventos / Listeners
  - **TableModelListener** (es usado por TableModel):
    - **tableChanged()**: En el evento se indica el rango de filas y columnas modificadas, y la operación (inserción, actualización o borrado)
  - **ListSelectionListener** (es usado por ListSelectionModel):
    - **valueChanged()**: Recibe un evento ListSelectionEvent cada vez que cambia la selección en la tabla.

Práctica: Datos deportivos



Fecha	Hora	Local	Visitante
07/05/2020	18:49	CHARLE(BEL)	CHIEVO(ITA)
03/04/2020	08:52	CLUBRU(BEL)	SPAL(ITA)
02/04/2020	08:53	STRUID(BEL)	HAMBUR(ALE)
02/04/2020	08:54	LYON(FRA)	LILLE(FRA)
22/04/2020	19:38	ROMA(ITA)	BSIAMO(ALE)
29/05/2020	14:00	ROMA(ITA)	ADODEN(HOL)
12/12/2020	12:20	AEKATE(GRE)	APOEL(CHI)

Práctica: Datos deportivos

- Ajustes: Añadir atributo "id" a Partido
  - Necesario porque no se va a almacenar el atributo "_links" de la respuesta de la API.

Práctica: Datos deportivos

- Mostrar datos **locales** en tabla Partidos:
  - Interfaz PartidosService
  - Servicio Partidos (local)
  - Crear TableModule para consumo del servicio

Práctica: Datos deportivos

- **Inicializar el servicio en una vista con el bean adecuado:**
  1. Permitir obtener el contexto de la aplicación en una vista:
     - Crear clase que implemente *ApplicationContextAware*
     - Agregar miembro y get estáticos para el ApplicationContext
     - Sobreescribir "*setApplicationContext()*"
     - Declarar como @Component
  2. En las clases, usar el get para obtener el contexto.
     - A partir del contexto obtenemos los beans con "getBean(class)"

Práctica: Datos deportivos

- **Servicio Partidos (API Backend):**
  - Llamadas HTTP
  - Conversion de respuesta HTTP a objetos de negocio
  - Implementar getPartidos()

HttpClient (Apache HttpComponents)

- HttpClient 5 (*org.apache.httpcomponents.client5*): Librería para uso de HTTP del lado de cliente.
- API completa: **HttpClient** *(httpclient5)*
- ➢ API "reducida": **HttpClient Fluent** (*httpclient5-fluent*):

HttpClient (Apache HttpComponents)

- Clase **Request**:
  - Métodos *get, post, delete*, etc.
  - Ejecución por defecto es síncrona, pero dispone de ejecutor asíncrono.
  - Comportamiento básico:
    - Hace la llamada HTTP
    - Si hay error lanza *IOException* o *HttpException*
    - Si es correcto devuelve el contenido como texto, InputStream o lo guarda en un fichero
    - No convierte el contenido a objetos, hay que hacerlo manualmente.

Conversión de respuesta a objetos

- El contenido de la respuesta (método GET) de nuestra API es una cadena de texto en formato JSON (application/json). debemos transformarla en una Lista de Partidos.
- Manejo de formato JSON en Java:
  - Librería **Jackson** (*https://github.com/FasterXML/jackson*)
  - Clase **ObjectMapper**: métodos de utilidad para interpretar texto en formato JSON
  - *Serializar*: convierte objetos java en texto JSON
  - *Deserializar*: convierte texto JSON en objetos java

Conversión de respuesta a objetos

- Paso 1: Convertir el texto en una estructura de objetos JSON:
  - Método *ObjectMapper.**readTree***():
    - Crea una estructura en árbol a partir del texto en JSON
    - Cada nodo (incluido el raíz) es un objeto ***JsonNode***
- Paso 2: Localizar el nodo que contiene el array de partidos:
  - Cada nodo tiene un método ***get****(<atributo>)*, para leer el contenido de un atributo "hijo"
  - Nodo principal → atributo "_embedded" → atributo "partidos":

```
JsonNode nodoPartidos = nodoPrincipal.get("_embedded").get("partidos");
```

  - Alternativa: buscar directamente el nodo por su nombre, con ***findValue()***:

```
JsonNode nodoPartidos = nodoPrincipal.findValue("partidos");
```

Conversión de respuesta a objetos

- Paso 3: Recorrer los elementos del nodo "partidos":
  - Método "elements()" de JsonNode → Iterador de hijos de un nodo:

```
for (Iterator<JsonNode> iterator = nodoPartidos.elements(); iterator.hasNext();) { ... }
```

- Paso 4: Convertir cada nodo hijo en un objeto Partido:
  - Método ObjectMapper.readValue(texto, Tipo): Crea un objeto del tipo indicado:

```
Partido partido = objectMapper.readValue( nodoPartido.toString(), Partido.class);
```

  - Para evitar que dé error por las propiedades "desconocidas" (_links):
    - Configurar el objectMapper:

```
objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
```

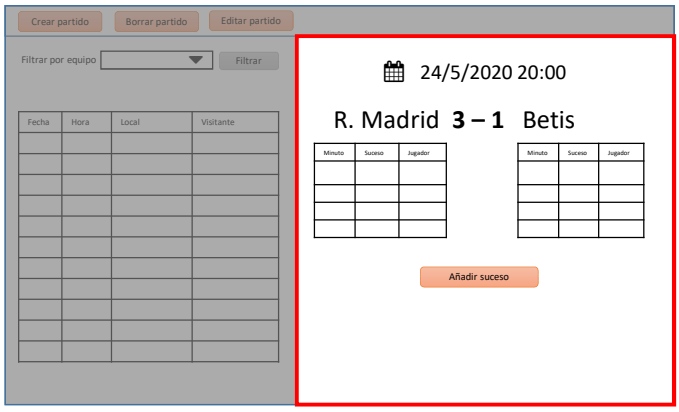
Varios

- Mostrar la ventana principal centrada en la pantalla
  - *setLocationRelativeTo(null);*
- Cargar una imagen directamente de una URL de internet:
  - *new ImageIcon (new URL("http://....."));*
- Centrar un componente en su contenedor (vertical y horizontal):
  - Usar GridBagLayout para el contenedor padre

Práctica: Datos deportivos

- **Vista Detalle de Partido**
  - Layout GridBagLayout
  - Servicio Partidos:
    - operación *getSucesosPorPartido()*
  - Table Model para tablas de sucesos

Layout Partidos



GridBagLayout

Dispone componentes en una rejilla. Permite combinar celdas, y que haya celdas no ocupadas.
Establece la posición, alineación, tamaño relativo, etc., de cada componente añadido a través de un objeto ***GridBagConstraints***.

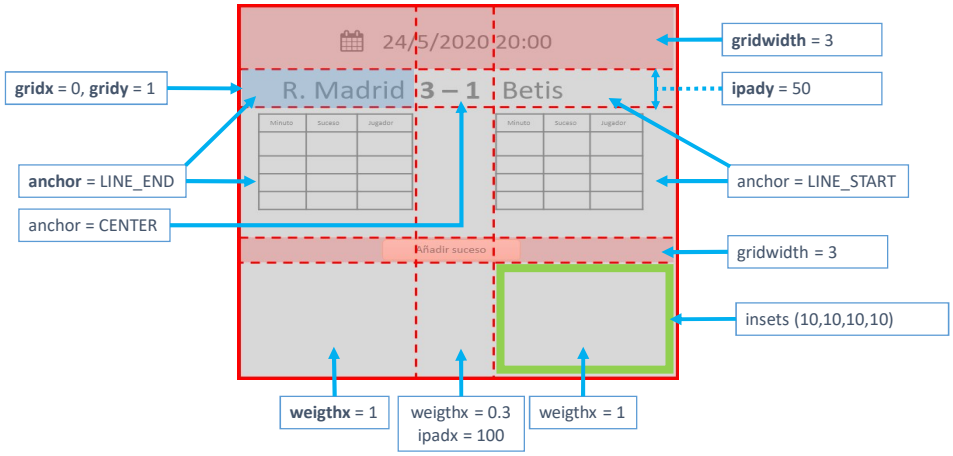
- Añade componentes al layout con **add**(*componente*, *gridBagContraints*)
- Objeto ***GridBagContraints***: define propiedades del componente en el layout:
  - **gridx, gridy**: fila y columna donde se añade
  - **gridwith, gridheight**: número de filas y columnas que ocupa (rowspan / colspan)
  - **ipadx, ipady**: ancho y alto mínimo que debe ocupar el componente (en puntos)
  - **weightx, weighty**: peso relativo del espacio del componente (de 0 a 1)
    - *Un componente con peso 0.5 ocupará la mitad que otro con peso 1*
  - **anchor**: Punto de anclaje del componente respecto al espacio ocupado
  - **insets**: márgenes internos (en puntos)

GridBagLayout

- Valor de "anchor" (Constantes de GridBagContraints)

FIRST_LINE_START	PAGE_START	FIRST_LINE_END
LINE_START	CENTER	LINE_END
LAST_LINE_START	PAGE_END	LAST_LINE_END

GridBagLayout



Práctica: Datos deportivos

- Mostrar detalle de partido al seleccionar un partido en la lista:
  - Cambiar el constructor del detalle para que inicialize el partido con el parámetro
  - Capturar el cambio de selección en JTable → ListSelectionListener
  - Obtener el índice de la fila seleccionada
  - Convertir al índice del modelo (puede ser diferente si se ha ordenado o filtrado)
  - Crear la VistaDetallePartido, pasando el objeto Partido asociado al índice en el modelo

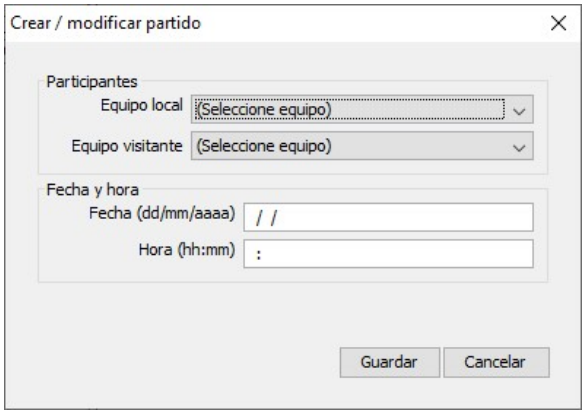
Práctica: Datos deportivos

- **Filtro por equipo participante:**
  - Operación *getPartidosPorParticipante()* en servicio Partidos
  - Servicio para Participantes
  - Cargar el combobox con los datos del servicio (operación *getParticipantes*)
  - Capturar el cambio de valor en el combobox
  - Recargar la tabla de partidos

Práctica: Datos deportivos

- Formulario de creación y modificación de partidos:
  - Layout
  - Inicializar combobox de participantes
  - Formatear texto para TextField de fecha y hora
  - Botón Cancelar: salir sin guardar
  - Botón Guardar:
    - Validar datos
    - Crear nuevo partido con servicio Partidos
    - Actualizar vista Partidos
  - Llamar a formulario desde VistaPartidos

Layout Formulario Partido



Layout Formulario Partido