

## Confección de informes

Los informes obtenidos de una aplicación permiten **extraer y analizar diferentes** tipos de **información**:

- datos relativos al **uso** de esta,
- otro tipo de contenido que permiten analizar los datos tomados desde la aplicación en su **conexión a un origen de datos**.

Una de las herramientas para la creación de reportes más utilizadas en la actualidad es **JasperReports + iReports**. Esta herramienta pertenece a **Jaspersoft** y está disponible desde los principales IDE (**Eclipse y Netbeans**).

La creación de informes permite **adaptar** la presentación de los datos finales **a múltiples tipos de escenarios**, tanto en cuanto a las **plantillas** utilizadas como a la **forma de mostrar la información**. El **manejo** de estas herramientas vinculadas a entornos de desarrollo de programación es una **herramienta clave**.

### *¿Es necesaria la confección de informes?*

Una **base de datos** por si sola no nos permite **obtener conclusiones** acerca de sus datos, para ello se necesita obtener información útil mediante la **correcta gestión y presentación de los datos** que contienen.

Lo más recomendable es obtener información útil de manera que **cualquier persona pueda entender el formato** de los datos y que además éstos **sean portables**. Para ello existe lo que conocemos como **informes**, que permiten **ver los detalles importantes a primera vista**.

La construcción de informes es, sin duda, un proyecto **más complejo que la creación de consultas o formularios**. Es una tarea que requiere un poco de **planificación**.

## Informes incrustados y no incrustados en la aplicación

Un **informe** es un **documento** que **recopila y muestra** diferente tipo de **información** procedente de **fuentes de datos** a los usuarios en función de las **consultas realizadas**.

La creación de estos informes **no es estática**, sino que, a través de las herramientas existentes para la generación de los mismos, será posible **seleccionar entre múltiples elementos y plantillas** de diseño que permiten crear los documentos finales. La creación de informes resulta un elemento clave en muchos **procesos**:

- Distribuir información con **formato legible**.
- Realizar **cálculos** y mostrar datos **útiles**.
- Los datos son claves para la **toma de decisiones**.

Es posible diferenciar entre dos **tipos de informes** atendiendo a la **forma en la que estos quedan vinculados con la aplicación**. Todo informe requiere de un origen de datos del que tomará la información necesaria para su construcción, ahora bien, en función de si esto se hace **desde la misma aplicación** en la que se desarrolla el proyecto o en **otra externa**, nos permite distinguir entre dos tipos de informes, incrustados y no incrustados:

### • Informes incrustados.

Este tipo de informe es el que se genera directamente desde la aplicación en la que el proyecto que se está desarrollando. Podemos decir que son herramientas **integradas en el propio IDE**.

Cuando se produce la creación de un informe, de forma automática **se genera una clase contenedora** con el mismo **nombre** que el **informe**. De esta manera, la **interacción** entre el proyecto y el informe se hace directamente a **nivel de clases**.

### • Informes no incrustados.

Este tipo de informe es **externo** al proyecto desarrollado, se ha creado en aplicaciones específicas para este fin, pero externas **al entorno de desarrollo** del proyecto.

No obstante, es posible habilitar la **interacción entre ambos**, aunque **no** se llegará a **crear una clase contenedora del informe** como sí ocurre en el caso de los incrustados.

Algunas de las **buenas prácticas** que se deben tener en cuenta a la hora de crear un informe es:

Ser **ordenados y estructurados** para ahorrar tiempo y no utilizar el método prueba y error.

Tener **claro el propósito** del informe: investigar quién lo **solicita** y a quién va **dirigido**.

Representar los **datos** de una manera **lógica**.

Abordar la pregunta de si será necesario un **único informe** o si será recomendable realizar **varios informes**.

## Herramientas gráficas integradas en el IDE

Las **herramientas gráficas** para la creación de informes permiten al usuario realizar de **forma visual el diseño** de los informes que van a ser generados.

Algunas de las herramientas más importantes son

- iReport,
- JasperReport,
- Eclipse Birt.

### iReport + JasperReport

Se trata de la **herramienta más utilizada** para los desarrollos utilizando **Java** en entornos como **Eclipse** o **Netbeans**. Podemos decir que **JasperReports** se encarga de la **encapsulación de informes** en estos entornos, mientras que **iReport** aporta la **interfaz gráfica** que permite un diseño mucho más rápido y fácil de utilizar.



### Eclipse Birt

Eclipse **Business Intelligence and Reporting Tools** es una herramienta de código abierto que permite la creación de informes de forma **dinámica** para **entornos de inteligencia empresarial**. Se utiliza, sobre todo, para el desarrollo de **aplicaciones en Java**, tanto de **escritorio** como aplicaciones **web**. Toda la información relativa a esta aplicación se encuentra disponible desde el sitio web oficial de Eclipse.



## Crystal report

Esta herramienta de **inteligencia empresarial** se utiliza para crear informes de forma dinámica, sobre todo, en el entorno de desarrollo **Microsoft Visual Studio**.

### Informes con Crystal reports

El programa **Crystal Reports** puede realizar acciones relacionadas con:

- **resumir** datos,
- **ordenarlos**,
- dividirlos en **grupos**.

El programa incluye una serie de **opciones de resumen** que son muy útiles, como:

**Sumar** los valores de cada **grupo**.

**Contar** todos los **valores** o sólo los valores que son **diferentes** el uno del otro.

Determinar el **valor máximo**, el valor **mínimo**, el valor **medio** o el valor **enésimo mayor**.

## Descarga e instalación de iReport y JasperReports desde Eclipse

Desde Eclipse, es posible realizar la instalación y descarga de este **plugin** de forma rápida. En primer lugar, desde el IDE de Eclipse vamos al menú **Help** y seleccionamos Eclipse **MarketPlace**.

A continuación, desde la ventana de búsqueda donde aparecen todos los plugins disponibles, buscamos **Jasper** y pulsamos el botón Go. Nos aparecerá, entre las opciones disponibles, **Jaspersort Studio** y basta con pulsar el botón **Install**.

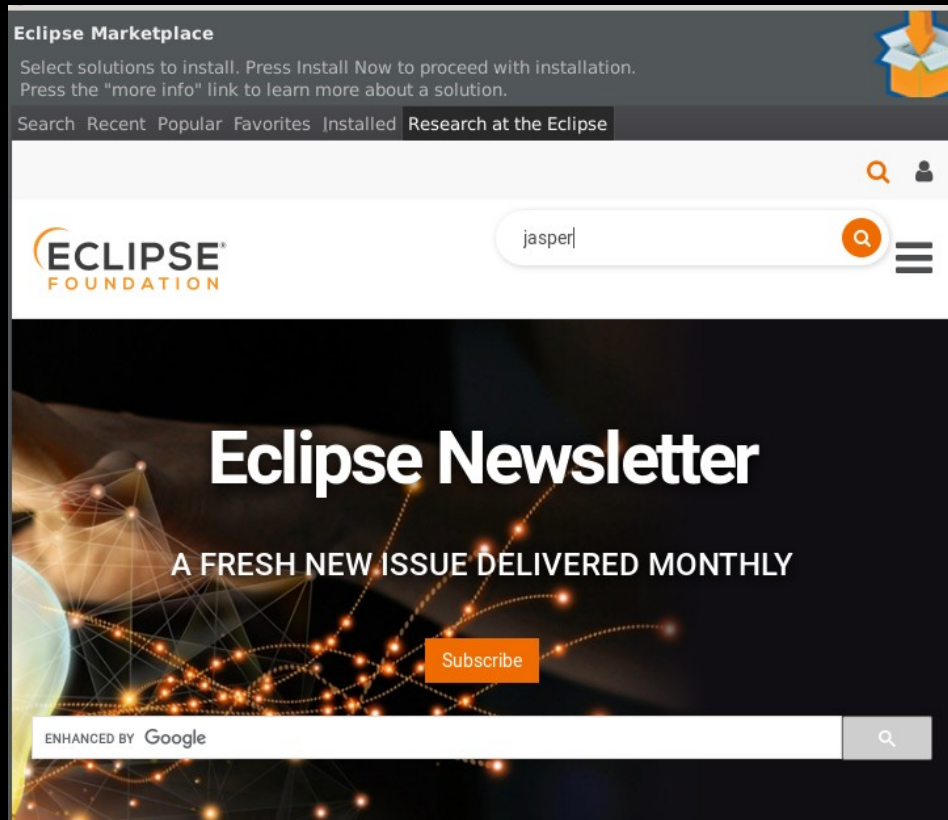
Finalmente, aceptamos las condiciones y dará comienzo el proceso de **instalación**. Para que la instalación se complete, será necesario **reiniciar Eclipse**.

Para comprobar que la instalación ha tenido éxito, se accede al menú **Open Perspective**, y como se puede ver en la siguiente imagen, ya tendríamos disponible el **plugin** para la **generación de informes iReport**.

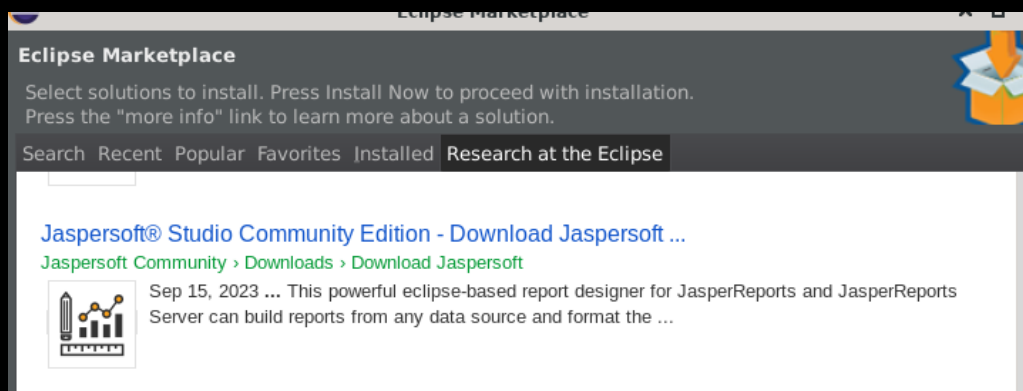
Esta opción funciona actualmente (diciembre 2023):

Instalación de JasperReports, Jaspersoft® Studio 6.20.6. en Eclipse.

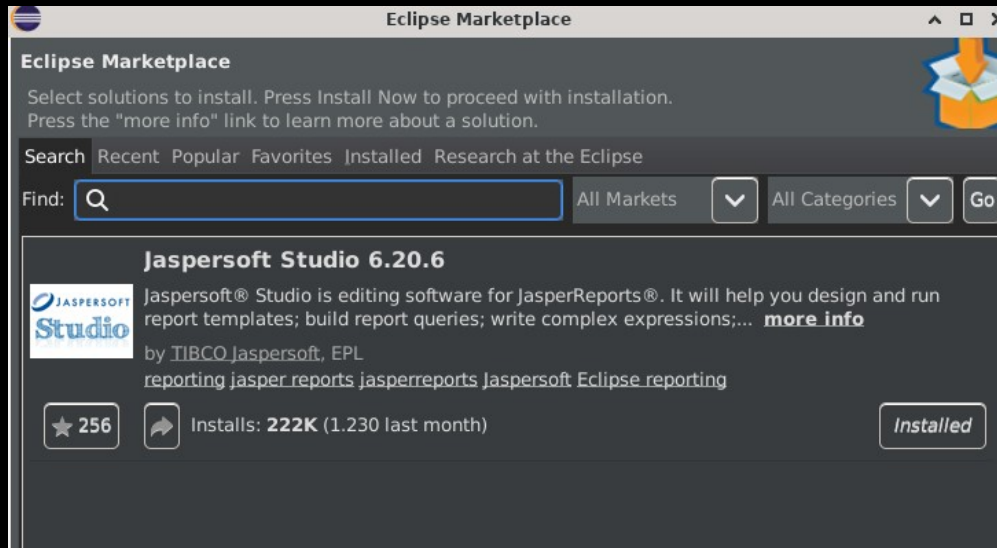
Se puede encontrar el instalable en Research at the Eclipse.



Al buscar por "jasper", aparece una lista de resultados, y es este resultado el del Report Design de Jasper...

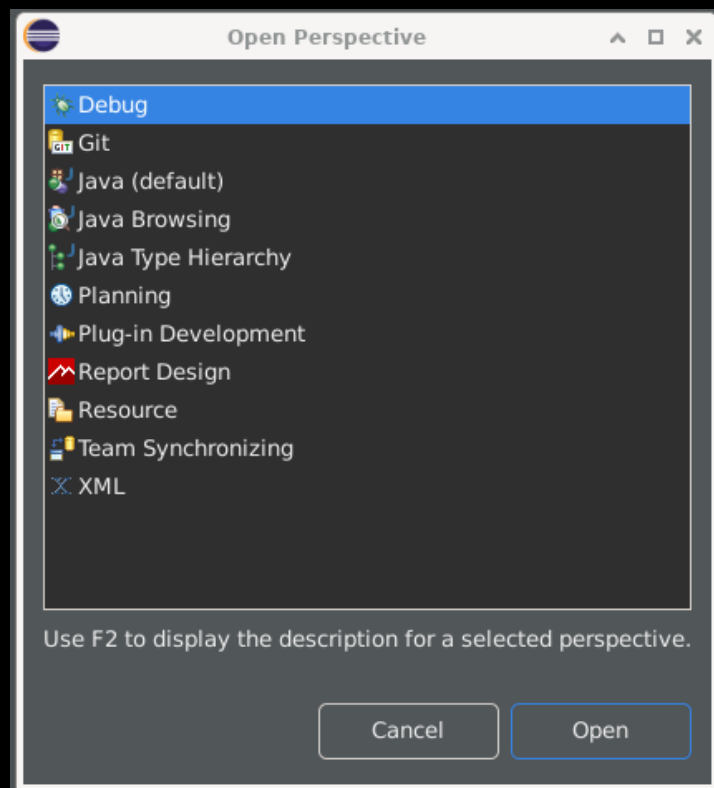


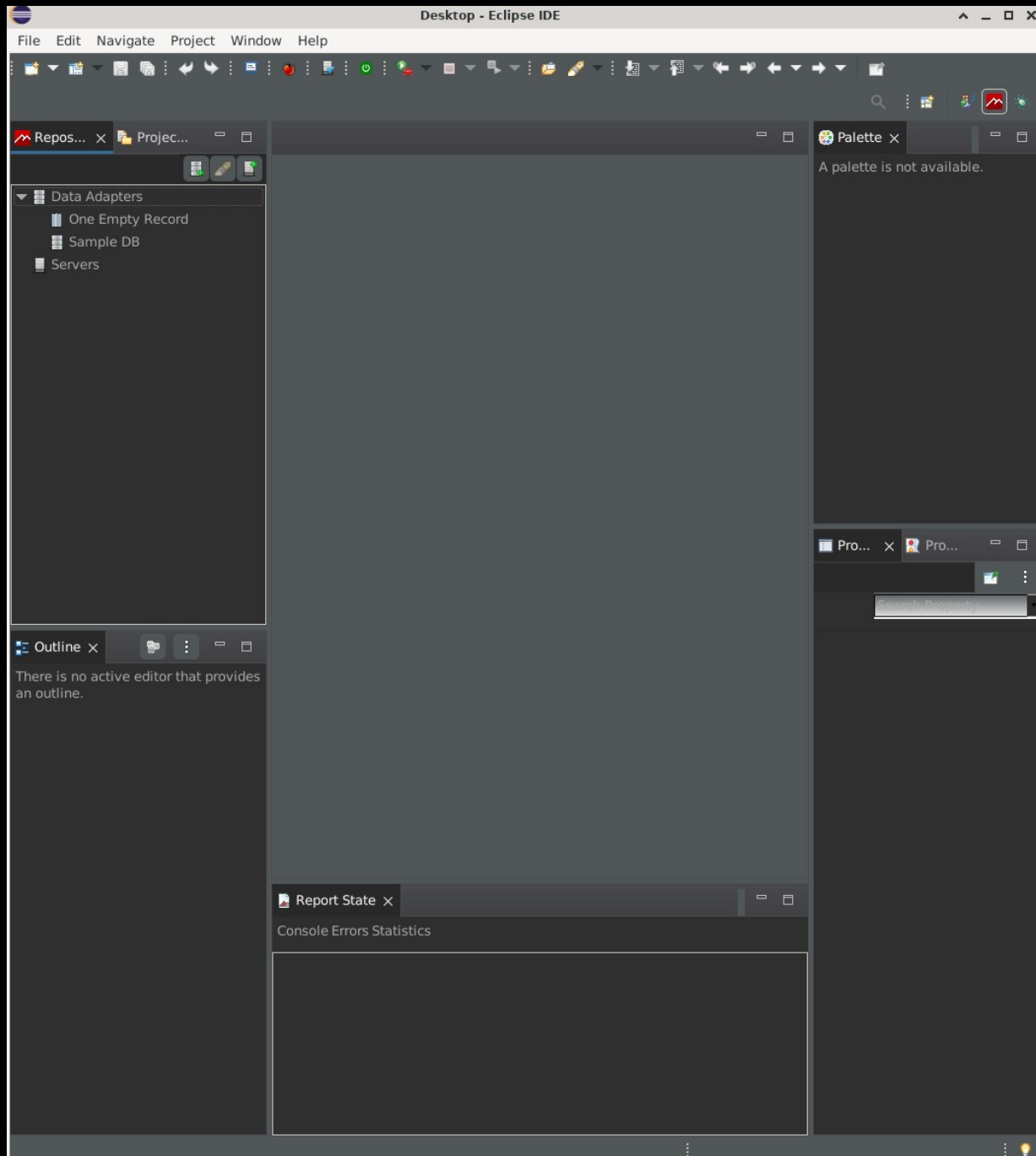
Nos abrirá el instalador del plugin; si aún no está instalado aparecerá el botón "install":



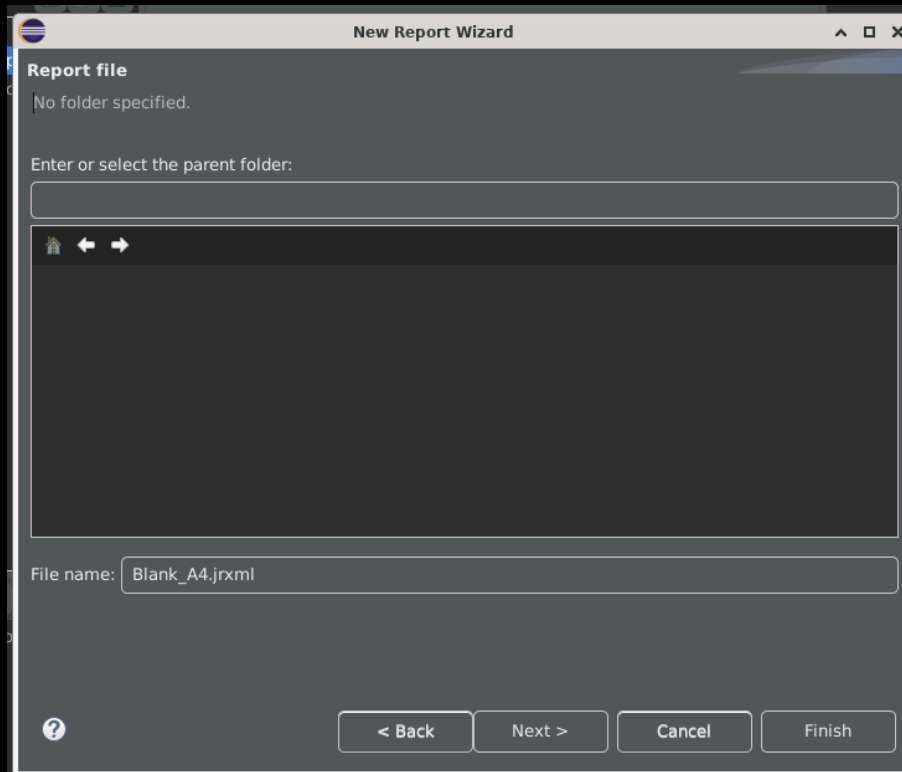
Una vez instalado ya aparece como perspectiva en nuestro Eclipse.

Y así se ve al abrir la perspectiva desde Window -> Perspective -> open perspective -> other:





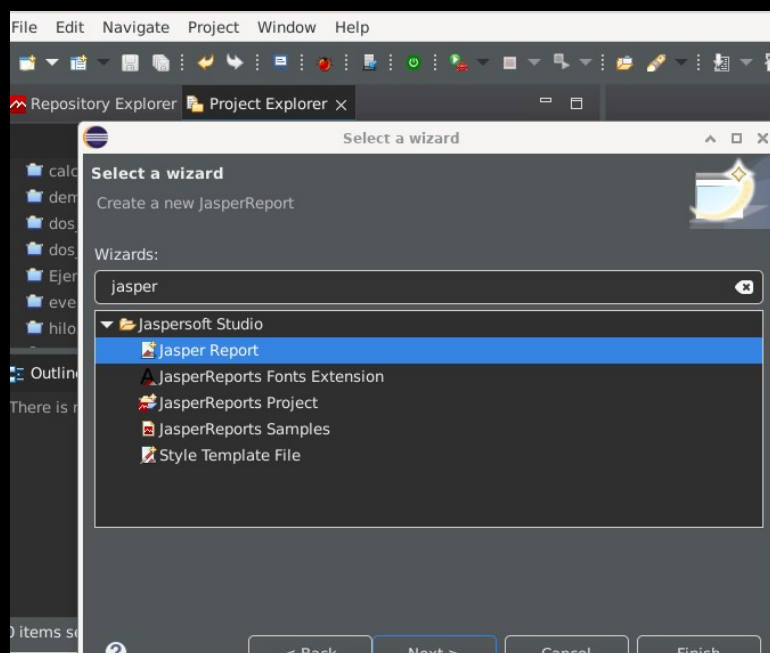
Sin embargo, no puedo hacer lo que hace el vídeo del punto 3.2 del tema, usar un proyecto que ya está creado para hacer un informe, no se puede elegir el proyecto relativo al informe, porque no hay proyectos para elegir en la ventana. Se lee "No folder specified." y en el cuadro "enter or select the parent folder:" no hay ningún directorio para elegir proyecto, es decir, no vienen ninguno por defecto como ejemplos...



Para poderlo usar:

En la perspectiva de Report Design, teniendo seleccionada la pestaña de Project Explorer (izquierda), vamos a menú File -> New -> Other

Y ahí tenemos proyectos de Jasper Report listos para usar como ejemplos:

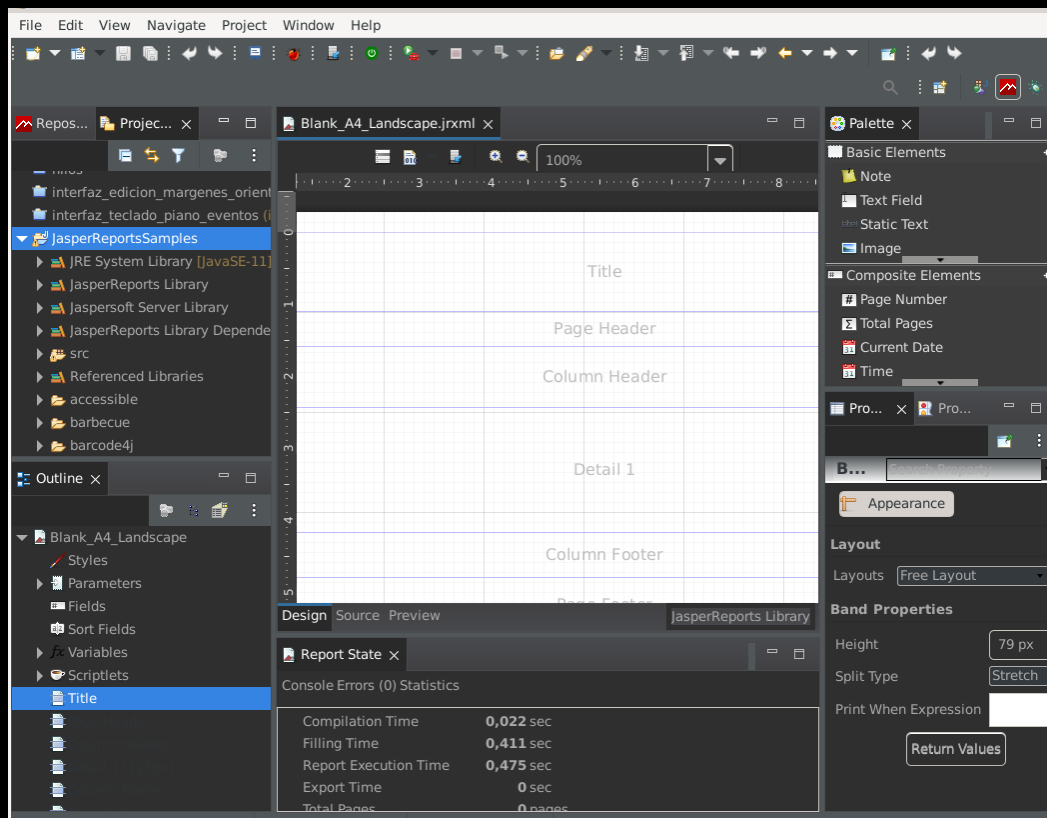




Si elegimos JasperReportsSamples, tendremos la posibilidad de hacer reportes nuevos o ver los que ya existen en este proyecto de ejemplo.

También podríamos crear nuestro propio proyecto conectado a nuestra base de datos (yo instalé la de postgresql pero puede ser cualquiera, mysql, etc...) y hacer lo mismo pero eligiendo nuestro propio proyecto. Por lo menos así podremos seguir los ejemplos del tema en Eclipse, quien quiera seguir los ejemplos de Eclipse.

Así se ve el modo diseño con el proyecto de ejemplo de Jasper, parecido al vídeo del tema:



Otra opción para la instalación consiste en descargar el paquete instalable desde la página de Jaspersoft Community.

Actualmente, está disponible desde el enlace (enlace roto, por cierto):

<https://community.jaspersoft.com/project/ireport-designer/releases>

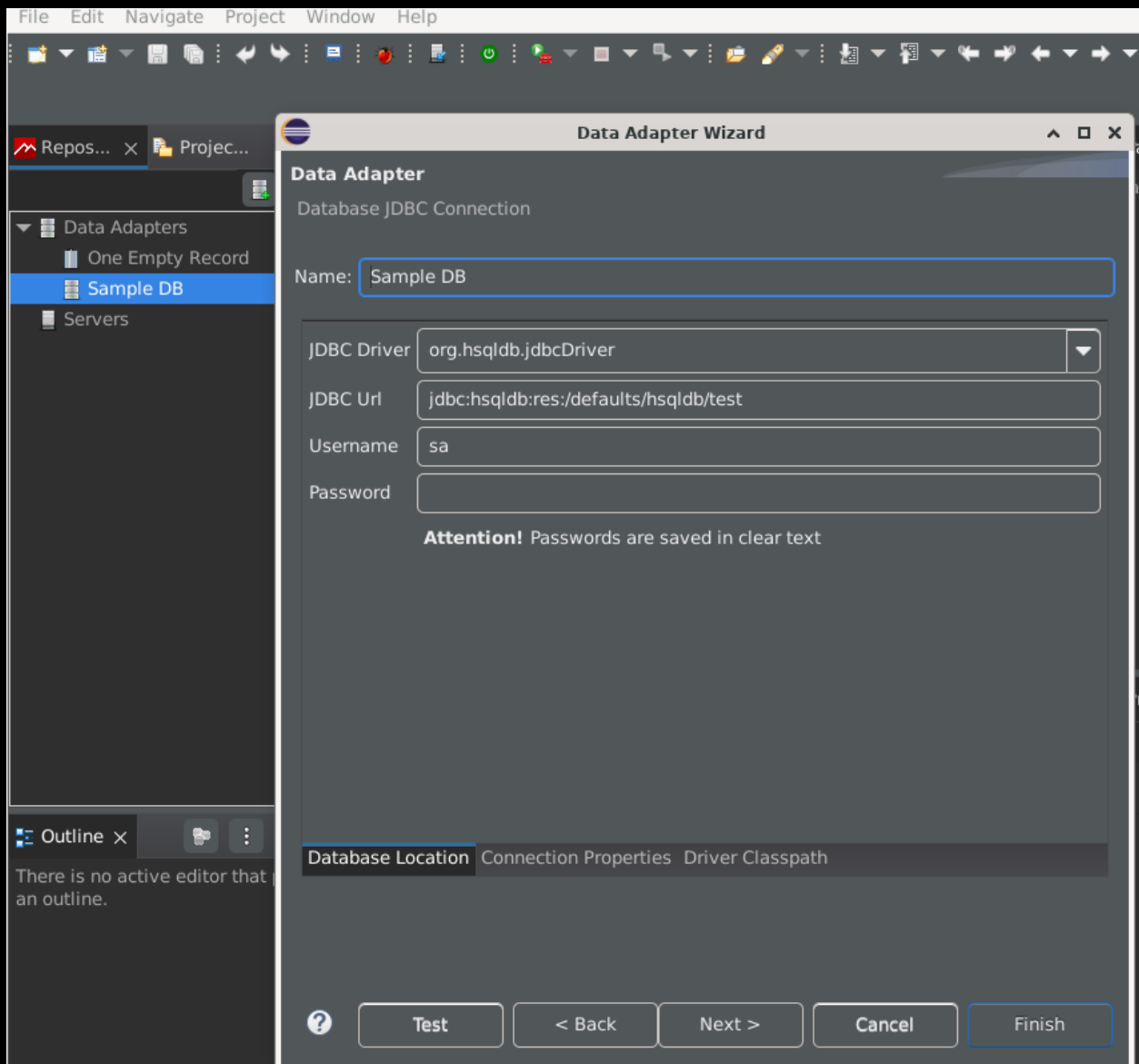
A continuación, seleccionamos la última versión y comienza la descarga de forma automática en nuestro equipo.

## Creación de un informe. Origen de datos

Para la creación de informes, es importante tener en cuenta el origen de datos, es decir, **de dónde se extrae la información** que se va a mostrar en un informe. Se dice que para la creación de estos iReports se pueden tener activos con **uno o varios orígenes de datos** (CSV, hoja de cálculo, XML...).

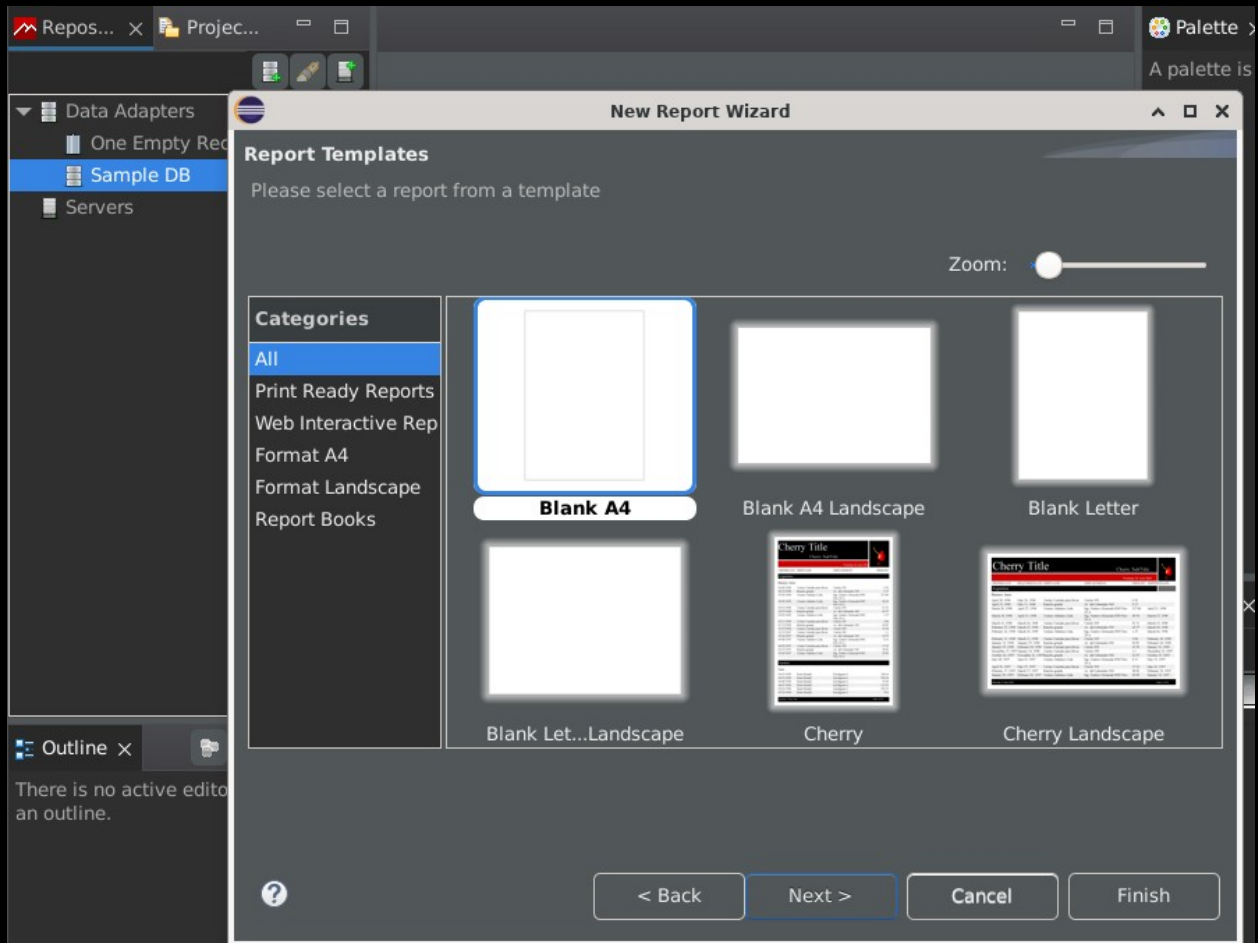
Para ilustrar la creación de un informe vamos a utilizar el que viene configurado por defecto, **Sample Db**.

Al pulsar sobre el botón **Test**, si todo está instalado correctamente, devolverá un **mensaje de éxito (sucessfully)**.



Hasta aquí ya se habría configurado todo el entorno para la creación de informes, ahora, para crearlos, basta con acceder al **menú File**, a continuación, **new File** y **JasperReport**.

Plantillas JasperReport:



Existen plantillas completamente en blanco para la generación de informes y otros documentos. En cada caso, se seleccionará el que más se adecúa al resultado que se desea conseguir.

Tras seleccionar el formato deseado, se pulsa el botón Finish. Finalmente, se solicita **seleccionar sobre cuál de los proyectos contenidos en el IDE se va a realizar el informe**.

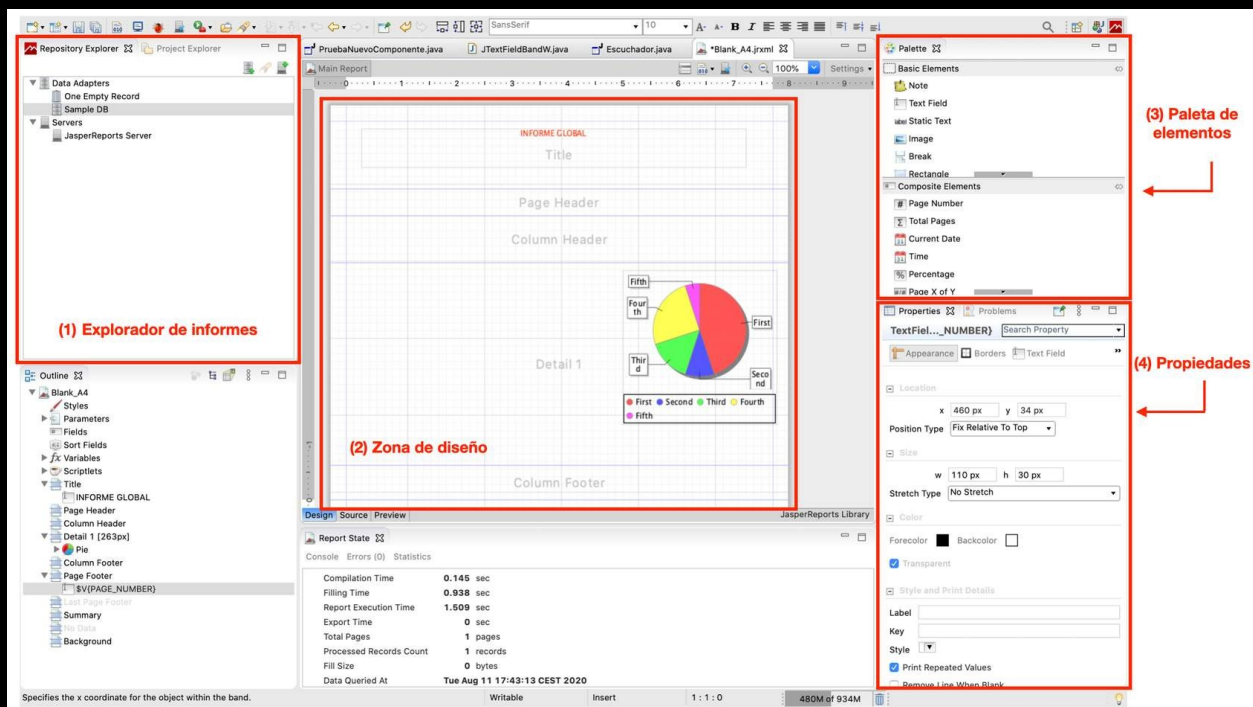
También se podría hacer primero el proyecto y luego agregar el informe sobre ese proyecto, de modo que ya apareciera el proyecto seleccionado en la misma ventana de creación del informe.

## Análisis de la herramienta. Estructura de la aplicación

La herramienta se encuentra dividida en varias secciones claramente diferenciadas:

- el **explorador** de informes (1),
- la zona de **diseño** (2),
- la **paleta** de elementos (3),
- la zona de **propiedades** (4).

Zonas interfaz JasperReport:



- **Explorador de informes:**

**Repository Explorer:** se accede a las conexiones a orígenes de datos.

**Project Explorer:** se muestra el resumen habitual de Eclipse en el que se muestran los proyectos y todos los ficheros que pertenecen a cada uno, incluyendo los generados por la herramienta iReport (presentan una extensión **.jrxml**).

- **Zona de diseño:**

Al igual que ocurría con el diseño de interfaces utilizando JSwing, esta zona es el **lienzo** sobre el que se va colocar y diseñar todo lo relativo al aspecto visual del informe. En la parte inferior, se observan tres **pestañas**:

- **Design** (utilizada para realizar el diseño),
- **Source** (donde se muestra el código generado),
- **Preview** (en la que se muestran una **visión preliminar** del diseño, siempre que no se produzcan errores durante el proceso de compilación).

- **Paleta de elementos:**

En esta sección, se muestran todos los **elementos** que se pueden utilizar para el diseño de un informe. Para colocar un elemento, se pulsa sobre él con el ratón y, a continuación, se coloca en la zona de diseño.

- **Zona de propiedades:**

Cuando se ha seleccionado y **colocado un elemento de la paleta** sobre la **vista** del informe, pulsando en cada uno de estos elementos aparece un nuevo menú, Properties, donde se recogen todas las propiedades que admiten **personalización** en cada uno de los **elementos**.

- **Zona de errores:**

En la **zona inferior** de la interfaz, aparecen los posibles **errores de compilación** que se deberán resolver. Recordemos que estos **informes** se **encapsulan como una clase**, por lo que cualquier **llamamiento o instanciación** de un objeto o variable debe realizarse **correctamente**.

## Estructura general y secciones de un informe

Una de las partes más importantes es la **zona de diseño** en la que se muestra la plantilla para el informe y está formada por diferentes bloques o bandas, sobre las cuales se podrán incluir los **elementos de diseño**. Podemos distinguir los siguientes **elementos**:

- **Title**: En esta sección, se coloca el **nombre** asignado al informe. Debe ser **claro y conciso** para que resuma en pocas palabras el tipo de datos que se recogen en el informe.
- **Page Header**: Esta zona se sitúa justo **debajo del título** del informe y, en ella, aparecen **datos generales** como la fecha de generación del mismo.
- **Column Header**: Es habitual que los informes se generen utilizando **columnas** bajo las cuales se organiza la **información a mostrar** en el informe. En esta sección, se colocan las **etiquetas identificadoras** de cada una de las **columnas**.
- **Details**: Podemos decir que esta es la **sección central** del informe en la que se detallan todos los **parámetros y valores** que se definen en él. En esta zona, se ubican los **datos completos**.
- **Column Footer**: Los pies de columnas se utilizan para expresar **datos relacionados con el contenido** de cada columna.
- **Page Footer**: Los pies de páginas se colocan en la parte inferior, y al igual que en el encabezado aparecen **datos generales**, aquí es habitual colocar el **número de página**.
- **Summary**: En esta última sección, se puede colocar los **valores calculados, recuentos** y demás **operaciones** realizadas sobre los datos. Si no se colocan aquí, el valor no resultante no se muestra.

Estructura plantilla en blanco:

"Campo de texto" Title			
Page Header			
Column Header			
Detail 1			
Column Footer			
Page Footer			
Summary			

## Formatos de salida de un informe

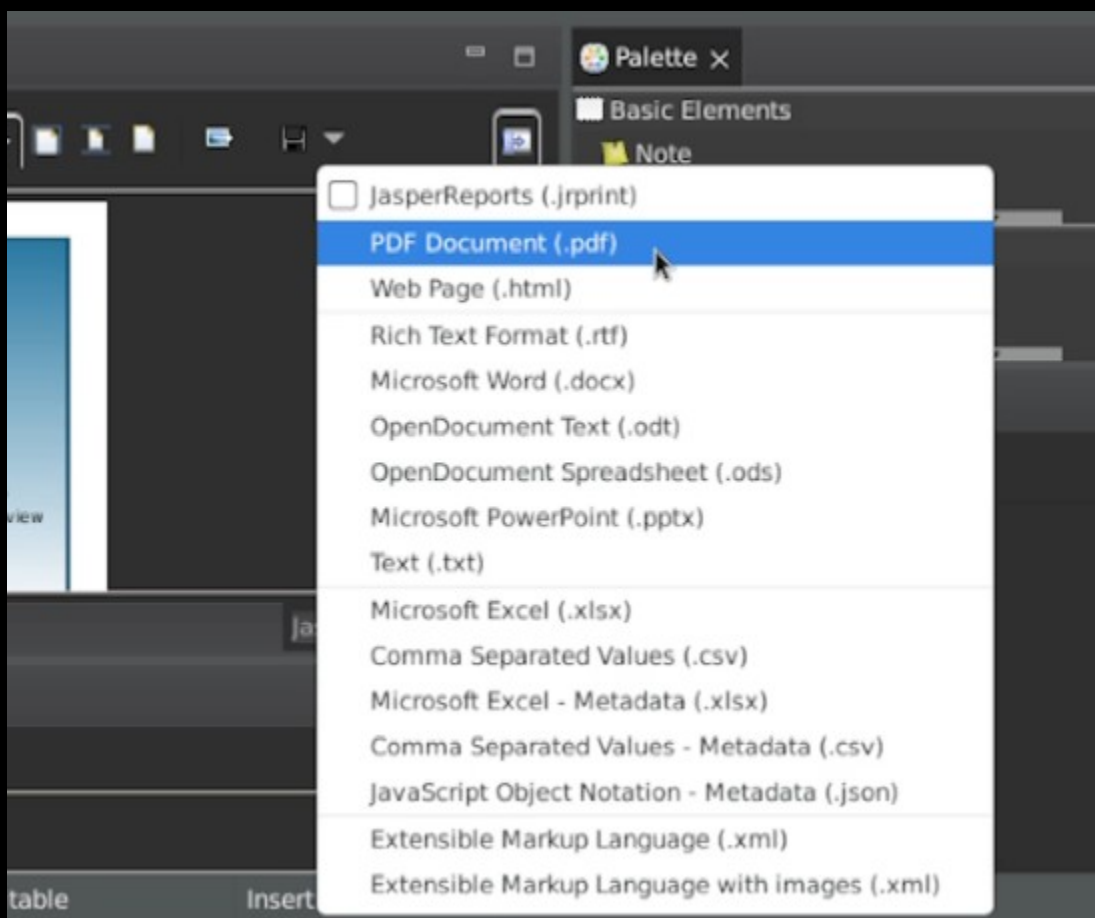
La herramienta iReport permite seleccionar entre múltiples opciones de formatos de salida para mostrar y generar el informe diseñado con los **datos y los cálculos escogidos**.

La **extensa paleta de formatos** que se incluyen en la mayoría de las herramientas para la creación de informes supone una de las principales **ventajas** de estas aplicaciones, puesto que será posible **adaptar el formato de salida a la finalidad** que se le va a dar al informe. Hay que tener en cuenta que, en muchas ocasiones, no solo es importante **qué se muestra**, sino también **cómo se muestra**.

Para visualizar el informe, desde Eclipse, en la pestaña **Preview**, es posible acceder a una **vista preliminar** del informe generado **como si de una imagen se tratara**.

Si lo que se desea es **ejecutar y guardar el informe**, es decir, obtener una versión “definitiva” del mismo, en la parte superior, aparece un **icono en forma de disquete** que nos permitirá **exportar el informe** en cualquiera de los **formatos de salida permitidos** en la aplicación.

Salidas posibles JasperReports:



Por ejemplo, si se desea enviar por correo electrónico, será aconsejable utilizar un formato como **PDF**, ya que permite su **descarga y visualización** de una forma **eficiente**.

Otro de los **formatos de salida** bastante utilizados son los **ficheros CSV**, puesto que permiten almacenar gran cantidad de información que habitualmente puede ser **procesada** por aplicaciones de **hojas de cálculo como Excel o Numbers**.



## Variables y valores calculados en un informe

El uso de las variables es clave para el cálculo de valores y su posterior inclusión en el informe. Es posible diferenciar dos tipos de variables:

- **Variables de usuario:** Este primer tipo de variables es el que puede crear el usuario. Normalmente, corresponde con el **nombre de los campos** del **origen de datos** vinculado al informe.
- **Variables predefinidas:** Estas variables **aparecen por defecto** en la herramienta, es decir, **siempre** van a estar presentes y **modelan elementos habituales** en la generación de cualquier informe.

### Variables predefinidas

Nombre	Descripción
<b>PAGE_NUMBER</b>	Objeto de clase Integer, recoge el número de páginas del informe.
<b>COLUMN_NUMBER</b>	Devuelve el número de columnas del informe (Integer).
<b>REPORT_COUNT</b>	Devuelve el <b>número de registros</b> para una determinada consulta (Integer).
<b>PAGE_COUNT</b>	Devuelve el <b>número de registros mostrados en cada página</b> (Integer).

También, es habitual el uso de determinados **valores calculados** que **resumen los datos** mostrados en un informe.

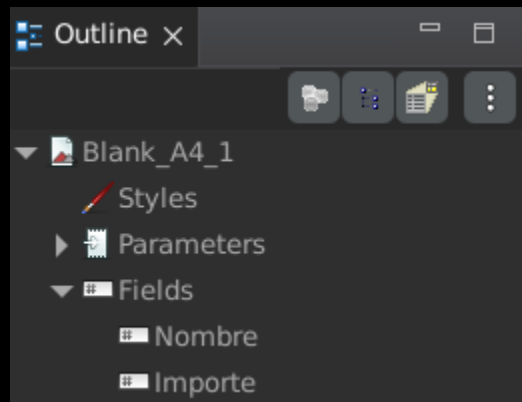
Podemos decir que se agrupan en función del criterio definido. Algunas de las operaciones más comunes son:

- **Numeración de líneas:** Se utiliza para numerar las líneas contenidas en un informe, habitualmente, en la zona de **Details**. Es necesario **crear una nueva variable** que irá **incrementando** su valor.
- **Recuentos y totales:** Este tipo de operaciones se utilizan sobre los **valores contenidos en una columna**, en concreto, se aplica la **operación suma** sobre todos los valores de una determinada columna.

La **operación SUM** devuelve un objeto java de tipo **java.lang.Number**.

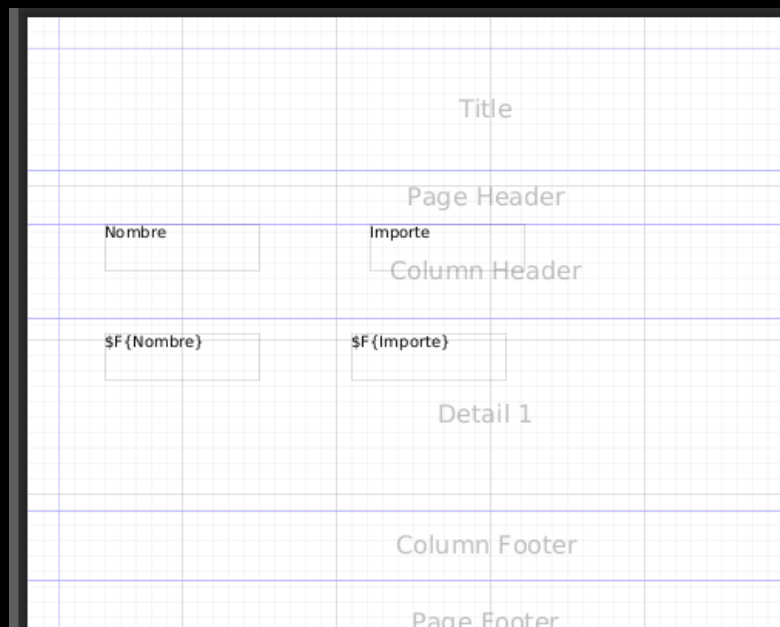
## Uso de variables en un informe

Para obtener valores calculados, es necesario utilizar variables. Para ello, en primer lugar, desde la zona del **explorador** donde se muestra el **resumen de todo el proyecto**, seleccionamos y **llevamos a la zona de diseño los campos** que se van a mostrar en el informe y sobre los que se va a realizar la operación deseada.

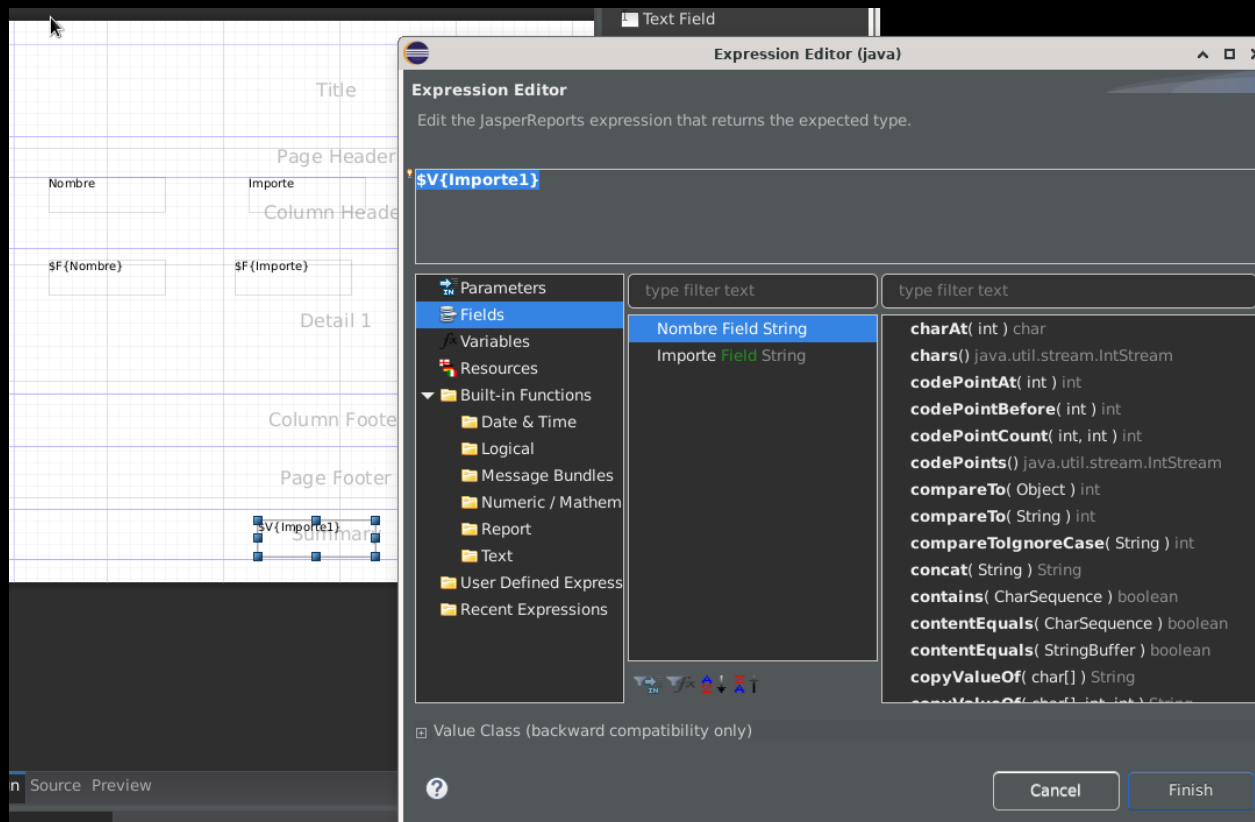


Cuando se **sitúan las variables** en la zona de diseño (sobre **Details**), se crea de forma **automática en Column Header** una etiqueta que indica lo que se muestra en cada columna.

Ejemplo de uso de variables en varias zonas de JasperReport:



Para mostrar el **valor resultante** de las operaciones realizadas, estas se deben **colocar en la zona de Summary**. En concreto, de la misma forma que se han colocado las **variables** en la zona de Details, también **se ubicarán en la de Summary**. A continuación, al hacer **doble clic sobre el nuevo elemento** situado en la zona de diseño, se abre un cuadro que permite seleccionar el tipo de operación a realizar.



Elección del formato de salida del informe:

Si el Informe es **mostrado en línea** al pulsar sobre un enlace en una página web: El formato escogido será **HTML** el cual permite descargar el informe con todas las etiquetas necesarias de creación de un sitio web.

Si se desea trabajar con los datos descargados utilizando **herramientas de cálculo como Excel o Number**: Sería posible escoger como **formato CSV o XLS**, y si es conocido que se va a utilizar con **herramientas tales como Excel**, podrían tomarse **directamente XLS**.

Y si el fichero descargado se va a enviar como adjunto en un **correo electrónico** a los miembros de una junta directiva y este no se puede modificar: Podríamos escoger **Docx o PDF**, pero dado que se especifica que el fichero **no se pueda modificar**, se seleccionará **PDF**.

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**Generación de informes a partir de una fuente de datos  
Filtros, gráficos, subinformes, editor de expresiones.**

Los informes obtenidos permiten utilizar elementos estáticos que se emplean sobre todo para definir la plantilla del informe, pero **necesitan de la conexión** a una fuente de **datos** para **iterar** sobre ella y **mostrar los datos** contenidos.

En función del tipo de la fuente de estos datos será necesario establecer una **configuración** u otra para la conexión, en cualquier de caso, se utilizan los llamados **adaptadores de datos**.

Cuando se conecta un informe a una tabla no siempre será necesario que se muestren todos los datos, campos o columnas, sino que en muchas ocasiones solo será necesaria **parte de la información**. Para conseguir esta selección se utilizará el **modelado de consultas** que permiten el **filtrado** de los datos.

JasperReport incluye muchas herramientas y elementos que permiten la creación de complejos informes en los que se muestran todos los datos necesarios, permitiendo además la personalización de los mismos, desde la inclusión de imágenes y gráficos, hasta la creación de subinformes o la edición de expresiones.

*Una **base de datos** por sí sola no nos aporta la suficiente información como para extraer **conclusiones**, para ello es necesario realizar un **informe** mediante el **filtrado** de datos que logre diferenciar un **subconjunto de registros** de otros, según si se cumplen una serie de condiciones o no.*

## Conexión con fuentes de datos y ejecución de consultas

Si vamos a trabajar con una **base de datos MySQL**, será necesario realizar una **configuración previa**.

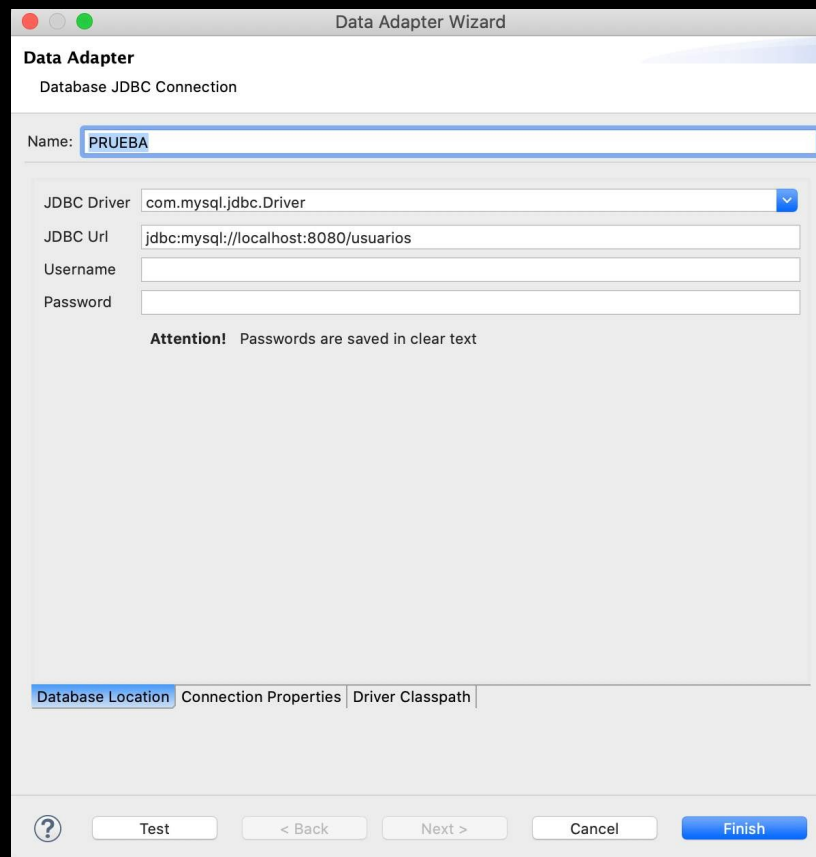
La **configuración de los parámetros para la base de datos** en MySQL es la siguiente:

En la pestaña **Database Location** (dentro del Data Adapter Wizard):

En primer lugar se ha de seleccionar el **JDBC Driver com.mysql.jdbc.Driver**.

A continuación, en la **JDBC URL** indicamos la **dirección exacta** de nuestra base de datos. En este caso se realiza en un **servidor local** por lo que aparece **localhost**.

Finalmente se añade el **nombre de la base de datos** (username) para la conexión, **usuarios**. Normalmente será root, y la contraseña que pusimos en la instalación.

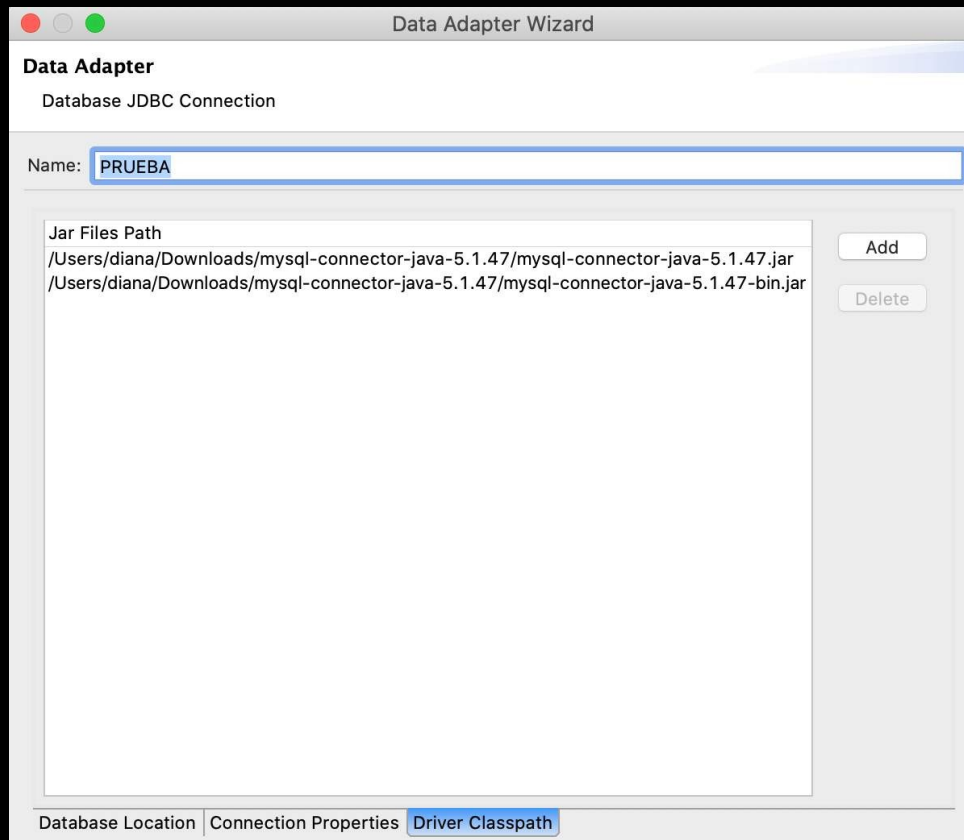


El último paso es la **configuración del Driver Classpath**:

Para ello desde la pestaña con el mismo nombre que aparece en la parte inferior, se seleccionará el **fichero .jar** para el **conector**. Es imprescindible la **instalación del conector** entre Eclipse, o el **entorno** de desarrollo escogido, y la **base de datos MySQL**.

Para **comprobar** que la **conexión** ha sido satisfactoria pulsamos el botón **Test**, obteniendo como resultado un mensaje donde se indica que la conexión se ha realizado con éxito.

### Classpath Data Adapter





## Diseño de consultas. filtrado de datos

El diseño base de consultas en lenguaje SQL se basa sobre todo en los **elementos** que se describen a continuación.

La sentencia **SELECT** es la utilizada para seleccionar los **datos** requeridos. Éstos pueden pertenecer a **una o varias tablas**.

La sintaxis de la sentencia SELECT consta de varias partes:

### Estructura completa SQL

```
SELECT *  
FROM a in alojamiento  
SELECT [* | DISTINCT] <campos>  
FROM <tablas>  
[WHERE <condicion> [AND | OR <condición>]]  
[GROUP BY <nombre_campo>]  
[HAVING <criterios_de_agrupación>]  
[ORDER BY <nombre_campo>|<índice_campo> [ASC|DESC]];
```

Tras la palabra **SELECT** se indican los **datos** que se quieren **mostrar**, indicando el nombre de las **columnas** o utilizando el **símbolo \*** para mostrar **todas** las columnas de la consulta. A continuación, se utiliza la cláusula **FROM**, que indica las **tablas en las que se encuentran los datos** solicitados tras la sentencia SELECT.

Existen multitud de **cláusulas opcionales**, son las mostradas **entre corchetes** en el código anterior. En ellas se definen distintos **criterios** sobre la manera de presentar la información o sobre **condiciones** que deben cumplir los datos.

Las tablas contienen múltiples datos que pueden no ser necesarios para cada informe, por lo que es habitual utilizar el filtro de datos a través de las instrucciones en SQL necesarias. El uso de **parámetros** para la construcción de las **sentencias de filtrado** será clave.

Para **delimitar y filtrar** los datos que se van a mostrar, se utiliza la **cláusula WHERE**, con el fin de indicar una condición determinada que se debe cumplir.

Por lo tanto, para la construcción de una **instrucción de filtrado** es imprescindible el uso de la palabra **WHERE**, tras la cual se indica el registro donde se va a realizar la selección y el **valor a filtrar**.

```
SELECT [* | DISTINCT] <campos>  
FROM <tablas>  
[WHERE <condicion> [AND | OR <condición>]]
```

Por ejemplo, en el siguiente código se mostraría **todas** las **filas de la tabla TablaDatos** siempre que se cumpla la **condición** recogida en **WHERE**.

Ejemplo filtrado

```
SELECT * FROM TablaDatos  
WHERE ID = $P{valor_filtrado}
```

### **Filtrado de datos**

En el filtrado de datos al **seleccionar un subconjunto** de los registros totales, el **informe** sólo se generará sobre los **registros que cumplan las condiciones** que se hayan establecido. Estas condiciones se crearán sobre el tipo de información que desee que aparezca en el informe final, por lo que no se incluirán todos los valores, sino **sólo un subconjunto**.

Por ejemplo, se pueden crear **filtros** para seleccionar los **elementos** siguientes:

Sólo un **grupo específico** de datos.

Los registros de un **rango de datos** seleccionado de la cantidad total de registros en la base de datos.

Sólo los valores de los registros que entran en un **rango de fechas** concreto.

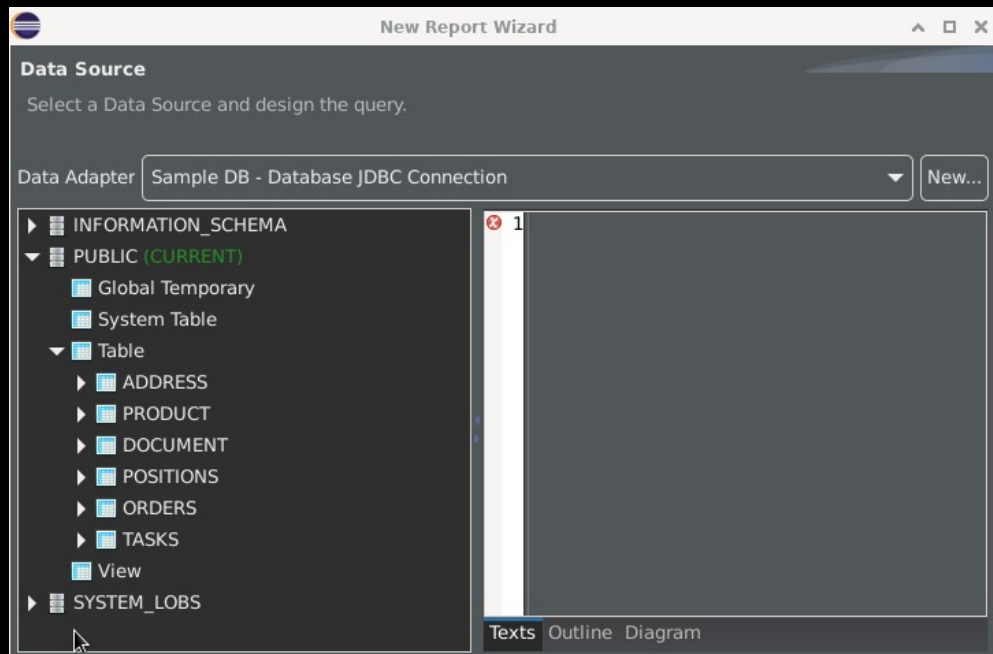
## **Ejecución de consultas**

Cuando se elabora un informe desde cero, en primer lugar, se solicita la selección de la base de datos y las tablas de las cuales se va a tomar toda la información necesaria para elaborar el informe.

Desde el menú **File** y **New Jasper Report** creamos un nuevo informe seleccionando la plantilla que más se ajuste a nuestro diseño.

A continuación, se ha de seleccionar el **Origen de datos (Data Source)**, en este caso escogemos **Sample DB**. Como se puede ver en la imagen, aparecen **todas las bases de datos de la conexión**. Si desplegamos la opción **Table** de cada una de ellas **aparecen las tablas que componen cada base de datos**.

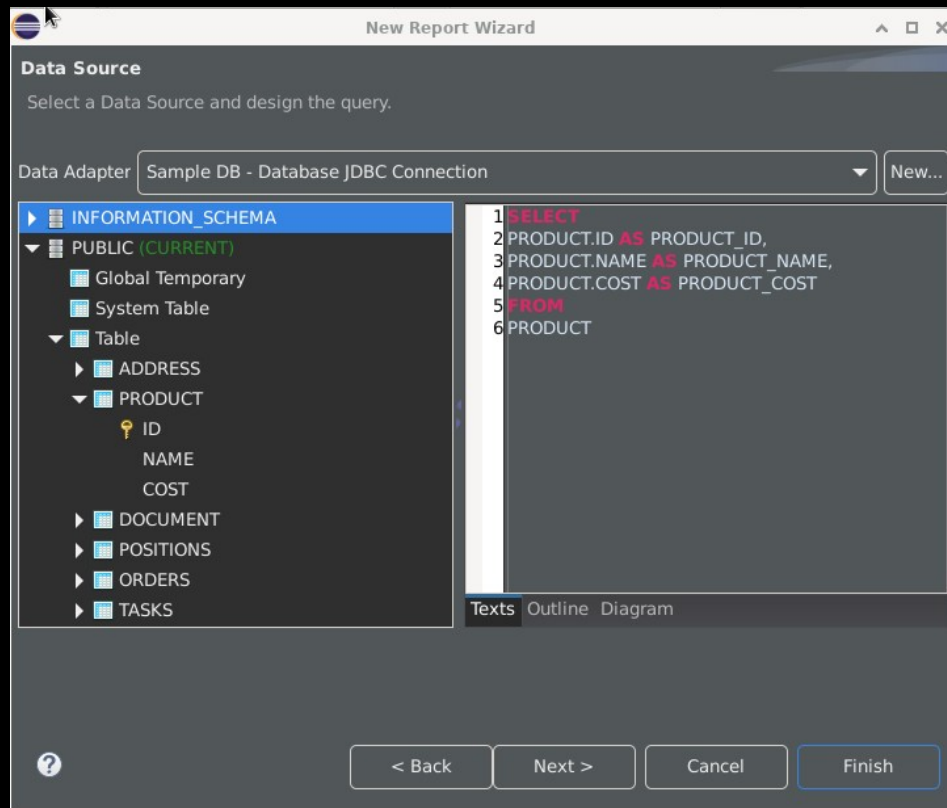
## Conexión Data Source



Para **seleccionar los campos** de las tablas que se quieran incluir en el reporte o informe, se lanzará una **consulta** para la selección de estos campos y se realizará el **filtrado** de datos oportuno.

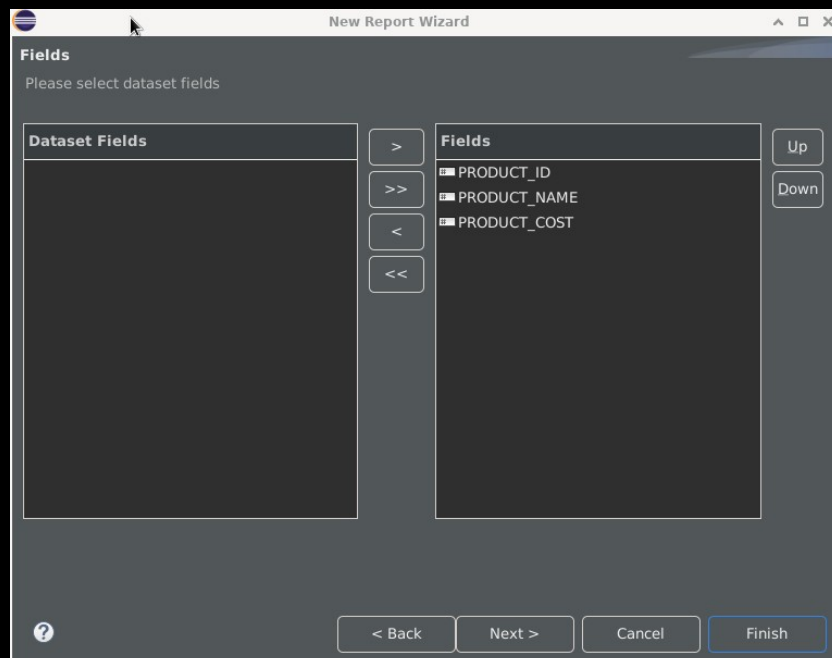
Consulta para volcado de datos ejemplo

```
SELECT
    PRODUCT.ID AS PRODUCT_ID,
    PRODUCT.NAME AS PRODUCT_NAME,
    PRODUCT.COST AS PRODUCT_COST
FROM
    PRODUCT
```



Finalmente, tenemos que **seleccionar todos los campos** que van a ser incluidos, para ello basta con **clickar la doble flecha** en las dos siguientes pantallas y posteriormente pulsar el botón **Finish**. Si todo ha concluido con éxito aparecerá un **mensaje de Congratulations**.

#### Selección de campos



New Report Wizard

Group By

Please select fields to Group By

Dataset Fields

>

>>

<

<<

Fields

PRODUCT\_ID

PRODUCT\_NAME

PRODUCT\_COST

Up

Down

☐ Use the group fields as sort fields. Select this option if you want to aggregate all the group fields with the same value and not only the consecutive ones

?

< Back

Next >

Cancel

Finish

New Report Wizard

Finish

We are ready to create your report

Congratulations!

All the information to create your new report have been successfully acquired.

Press Finish to generate the report.

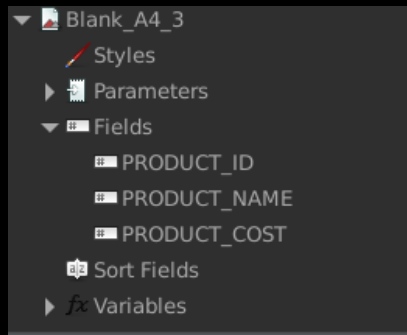
?

< Back

Next >

Cancel

Finish



## Inclusión de datos desde base de datos

Tras la **creación del informe** y la **conexión** de éste con las **tablas** contenedoras de los datos, será **posible utilizar estos campos** para su **importación en un informe**.

Para el diseño de un informe se debe diferenciar de los **textos estáticos (Static Text)** y los **campos de texto (Text Field)**. Los primeros se utilizan como si fueran etiquetas y son las habituales para la creación de los **títulos**.

Los **campos de textos** son los elementos que se van a utilizar para la inclusión de los **datos contenidos en las tablas**.

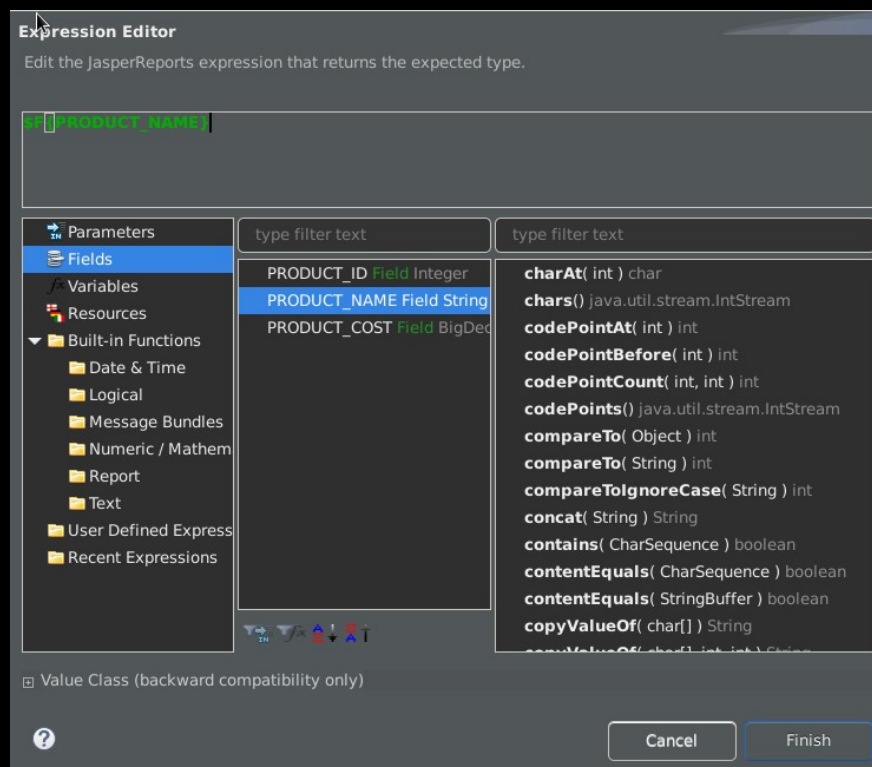
Hay que tener en cuenta que si éstos se colocan en la **sección Column Header** solo aparecerá el **primer elemento** de la tabla, si queremos que aparezcan **todos los elementos** se sitúa en **Details**.

Tras colocar los Text Field necesarios en la zona de diseño, hacemos **doble click sobre cada uno** de ellos.

Para seleccionar el **campo** que va a ser mostrado, **pulsamos** sobre el **menú Field** que aparece en la izquierda y en la **zona central** se cargarán todos los **campos** que se han **filtrado** durante el paso anterior.

Finalmente, hacemos **doble click** sobre el **campo**, y el **editor de texto** que aparece en la parte **superior** cargará el código necesario para **referenciar al campo** y **extraer su valor** desde el informe.

De esta forma, ya podemos **realizar todos los informes** con todos los **datos** que se necesiten.



## Subinformes

Para **mejorar la legibilidad** de la información y garantizar una estructura más óptima, podemos realizar también subinformes. Los subinformes son **informes incluidos dentro de otros informes**. Este tipo de informes son de gran utilidad a la hora de trabajar con datos relacionales puesto que permiten obtener **varias consultas diferentes** en un **mismo informe** de una forma más **organizada** y por lo tanto, más **fácil de comprender** por el usuario de este tipo de información.

Informes y subinformes presentan muchas características similares puesto que recogen y exponen la información y los datos **en base a una plantilla de diseño**. La combinación de ambos resulta clave para **combinar informes, coordinar datos y presentar varias vistas** en un mismo informe enriqueciendo el resultado final.

Entre ambos elementos existen **claras diferencias** que nos ayudarán a determinar de qué tipo de informe se trata:

### Diferencias Informes vs Subinformes

Informes	Subinformes
Puede contener subinformes.	No puede contener más subinformes.
Tiene encabezado y pie de página.	No tiene encabezado ni pie de página.
Existe como objeto independiente y principal.	<b>No tiene existencia por sí solo.</b>

Para incluir un subinforme se coloca en el visor del diseño desde la paleta de elementos el elemento **“Subreport”** (en Palette, Basic Elements), como si de otro elemento cualquiera se tratara.

En primer lugar, se debe seleccionar **qué tipo de informe** va a ser incluido, si se utilizará uno ya existente y previamente almacenado o si se creará uno nuevo.

Una de las principales características de esta herramienta es que permite la inclusión de otros informes **previamente diseñados** seleccionando la opción **“Select an existing report”**. A continuación, será posible escoger cualquiera de las siguientes opciones:



Subreport selection mode—

- ☒ Workspace resource (an element inside the workspace)
- ☐ Absolute Path in the filesystem (use only for quick testing, never use in real reports)
- ☐ URL (a remote URL referring to a subreport, will be the expression value)
- ☐ Select a resource from JasperReports Server
- ☐ Custom expression (enter an expression for the subreport using the expression editor)
- ☐ No subreport (no subreport reference will be set)

Options—

Select a resource from the workspace

Browse ...

Si escogemos la opción **“Create a new report”**, el proceso que sigue es igual al definido para la creación del informe principal.

Para concluir este proceso, aparece la siguiente ventana que nos permite definir el **tipo de conexión con el origen de datos**. Habitualmente seleccionaremos la primera de las opciones, **“Use same JDBC connection used to fill the master report”**:

### Selección conexión para subinforme

**Connection**

Please select the connection

- ☐ Use same JDBC connection used to fill the master report
- ☐ Use another connection

Browse ...

- ☐ Use an empty Data Source
- ☐ Use a JRDataSource expression

Browse ...

☒ Don't use any connection or Data Source

?

< Back

Next >

Cancel

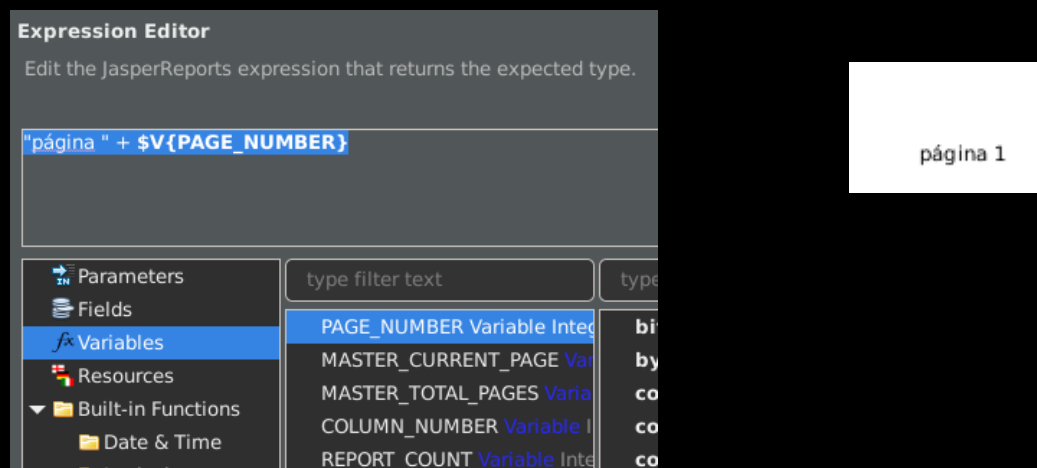
Finish

## Editor de expresiones

La inserción de valores en un informe no solo se restringe al contenido de los registros de una base de datos, sino que gracias al **editor de expresiones** es posible **personalizar la cadena de texto** con la que serán incluidos los datos en un informe.

Para acceder al editor basta con pulsar dos veces sobre cualquiera de los elementos contenidos en el visor del informe.

Interfaz Editor de expresiones



En la caja en blanco que aparece en la parte superior **podemos concatenar las variables**, así como otras **cadenas de texto** utilizando el operador +.

### Ejemplo concatenación valores en editor de expresiones

**"página " + \$V{PAGE\_NUMBER}**

A modo de aclaración: en la expresión `$V{PAGE_NUMBER}` `$V` significa el valor de la variable de `page_number`, si fuese `$F`, la `F` sería referida a `Field` (campo) que habríamos extraído de la base de datos o del fichero de datos. Igualmente si fuera `$P` sería referido a un parámetro.

Por ejemplo, en este caso se ha incluido la palabra 'página' al parámetro `PAGE_NUMBER`, por lo tanto, la salida debe incluir el valor del contador de páginas junto a la palabra 'página'. Para comprobarlo pulsamos `Finish` en el Editor de Expresiones y a continuación, vamos a la previsualización (pestaña `Preview`), como vemos en las imágenes de arriba.

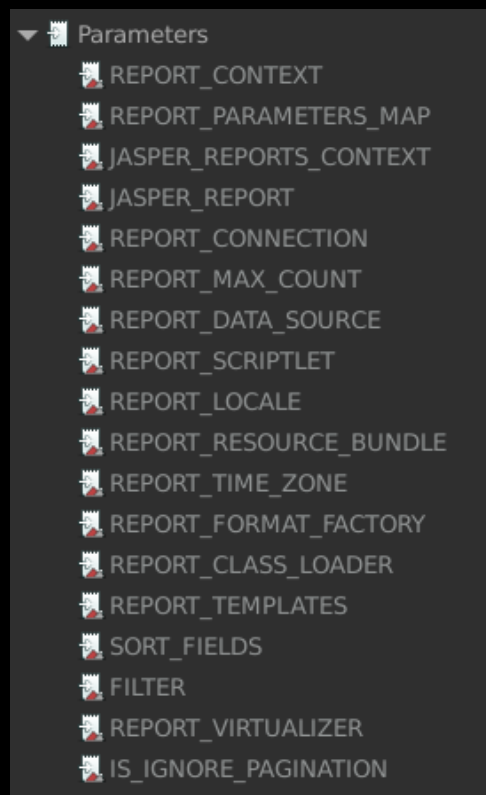
## Informes parametrizables

Los parámetros incluidos en la herramienta permiten la creación de **distintos resultados** utilizando como origen un **mismo fichero** de diseño. Estos parámetros nos permiten **seleccionar el dato mostrado en cada momento**.

Todo parámetro queda definido por un **nombre que lo identifica** y la **clase a la que pertenece**. Por ejemplo, el parámetro **Report Connection** permite mostrar el **nombre de la conexión**.

Todos los parámetros aparecen recogidos en el **menú Parameters**. Para seleccionar uno de ellos basta con pulsar sobre él y colocarlo en la posición exacta de la zona de diseño.

Menú desplegable con parámetros disponibles



Como casi todos los elementos, estos también permiten **modificar sus propiedades** para adecuarse al diseño. Desde la zona de propiedades será posible modificar ciertos datos; recuerda que estas opciones solo se muestran si está **marcado el elemento** en la zona de diseño.

The image shows a software interface for editing a 'TextField' component. The title bar reads 'TextField: \$P{REPORT\_CONNECTION}'. The 'Appearance' tab is selected, showing various configuration options:

- Location:** x: 0 px, y: 10 px. Position Type: Fix Relative To Top.
- Size:** w: 100 px, h: 30 px. Stretch Type: No Stretch.
- Color:** Forecolor (black square), Backcolor (white square). ☒ Transparent.
- Style and Print Details:**
  - Label: (empty text box)
  - Key: (empty text box)
  - Style: (dropdown arrow icon)
  - ☒ Print Repeated Values
  - ☐ Remove Line When Blank
  - ☐ Print In First Whole Band
- Print When:**
  - ☐ Detail Overflows
  - Group Changes: (dropdown arrow icon)
  - Print When Expression: (empty text box with a small icon to the right)

An 'Edit Properties' button is located at the bottom of the panel.

Existen varios tipos de parámetros:

integrados y de usuario:

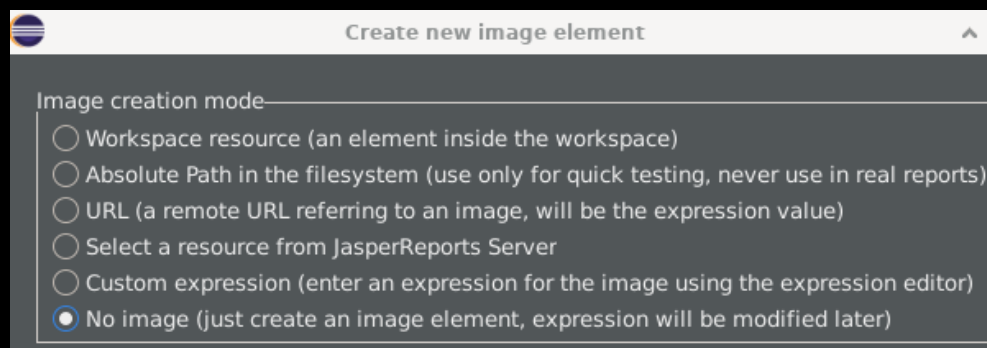
- **Integrados:** son aquellos que están disponibles de forma predeterminada en el entorno de **desarrollo**,
- **de usuario:** son los que vienen determinados por el usuario, es decir, pueden crearlos desde cero. Permiten tomar los campos y operar con ellos sobre el informe, se describen de la siguiente forma: **\$F{nombre\_campo}**

## Inclusión de imágenes en un informe

Si seleccionamos una imagen y la colocamos en el documento solo estará disponible en el ordenador dónde se desarrolló inicialmente el informe.

Para colocar una imagen en un informe basta con seleccionar el **elemento “Image”** de la paleta de elementos y colocarla en la posición deseada del informe. Cuando realizamos esta acción aparece un **cuadro diálogo** que permite seleccionar el **tipo exacto de inclusión**, en cuanto al **origen** de la imagen se refiere.

Menú selección ruta origen imagen



Las opciones posibles, siguiendo el mismo orden que el mostrado en la imagen anterior son:

- Tomar una imagen almacenada en la **zona de trabajo** del proyecto (Workspace resource).
- Indicar la **ruta absoluta** en local del fichero de la imagen. Esta opción **solo se aconseja para pruebas**, y no para un entorno real ya que si se utiliza se estará apuntando todo el tiempo a una ruta en un ordenador concreto.
- **URL**. Permite indicar la dirección URL de una imagen.
- Indicar una ruta concreta en el **servidor JasperReports**.
- Diseñar una imagen utilizando el **editor de expresiones** de la propia herramienta.
- No imagen. Este elemento permite **definir “el hueco”** de la imagen, solo el elemento imagen, pero no se indica ninguna. La expresión se modificara después.

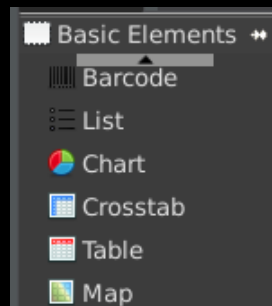
Código generador por la aplicación para inclusión de imagen

```
<image>
  <reportElement x="330" y="10" width="90" height="79" uuid="7f97ce2a-
560d-43ee-ac50-b772a2964f7a"/>
  <imageExpression>
    <![CDATA[""]>
  </imageExpression>
</image>
```

## Colocar gráficos en un informe

En este tipo de elementos se suele mostrar de forma visual los mismos datos que aparecen en forma de texto o cifras.

Para incluir un gráfico en la herramienta iReport en primer lugar seleccionamos el **elemento Chart** en la paleta de elementos.

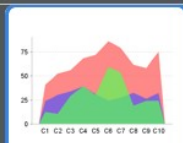


A continuación, se muestra una paleta con los tipos de gráficos que podemos utilizar para el informe. Como se puede ver en la imagen existen multitud de tipos de gráficos:

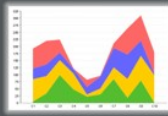
- **Gráficos de barras:** En los que la información se representa con barras verticales u horizontales que muestren los datos agrupados.
- **Gráficos lineales:** Este tipo de representaciones muestran valores en el **eje x e y** que aparecen unidos linealmente. Son muy útiles para **representaciones temporales**, y en ocasiones se superponen líneas para realizar **comparativas**, por ejemplo, el número de ventas entre departamentos a lo largo de un año.
- **Gráficos circulares:** Este último tipo se suele utilizar para representar **distribuciones**, ya que divide el total en diferentes fragmentos representados de la distribución.

## Chart Wizard

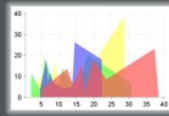
Select the chart you want.



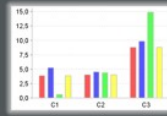
Area Chart



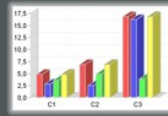
Stacked Area



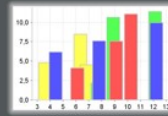
XY Area



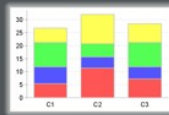
Bar Chart



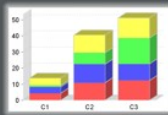
Bar3D Chart



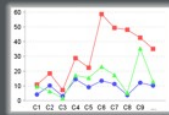
XY Bar



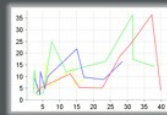
Stacked Bar



Stacked Bar 3D



Line Chart



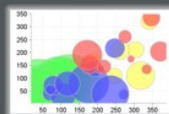
XY Line



Pie Chart



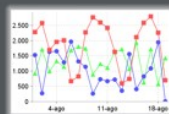
Pie3D Chart



Bubble Chart



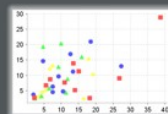
Candlestick Chart



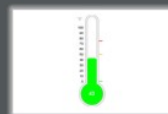
TimeSeries Chart



HighLow Chart



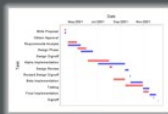
Scatter Chart



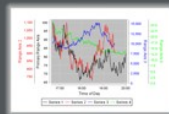
Thermometer Chart



Meter Chart



Gantt



MultiAxis Chart

Tras la selección del que más se adecúa a los datos que van a ser mostrados es posible **personalizar** el gráfico utilizando la **opción Chart Wizard** (botón derecho sobre el gráfico incluido).

## Ejemplo de creación de un informe

Se pide generar un nuevo informe utilizando una plantilla de diseño en blanco y la base de datos de **Sample DB** incluida en JasperReports que incluya los siguientes datos:

- NOMBRE DEL PRODUCTO (Tabla Product)
- PRECIO DEL PRODUCTO (Tabla Product)

En primer lugar, se realiza la conexión con la tabla que contiene esos datos, para ello se utiliza la siguiente Query.

Diseño Query

```
SELECT
    PRODUCT.ID AS PRODUCT_ID,
    PRODUCT.NAME AS PRODUCT_NAME,
    PRODUCT.COST AS PRODUCT_COST
FROM
    PRODUCT
```

Para que los datos se muestren, **arrastramos los campos** a la zona de diseño. Si se desea que en la parte superior **aparezca el nombre del campo** hay que colocarlo tanto en **Column Header** como en Details. Esto se consigue colocándolo primero desde “Fields” de la ventana “Outline” directamente hasta la zona de diseño, en Detail, y automáticamente se coloca también el nombre del campo (field) arrastrado en el espacio de “Column Header”.

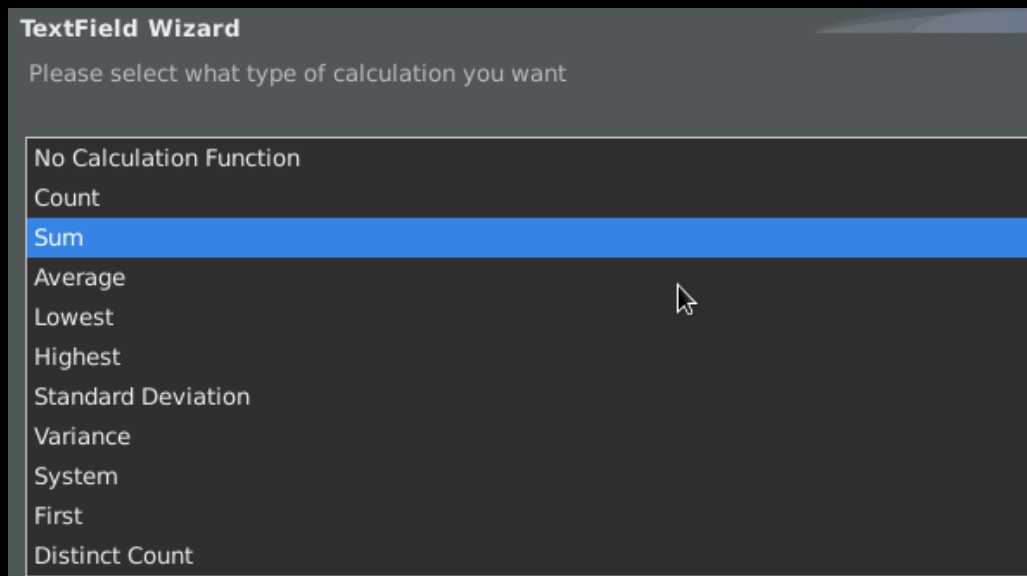
Finalmente, se añade un **título** utilizando **Static Text**.

El diseño podemos **visualizarlo** a través de la pestaña **Preview**, siempre y cuando **no se produzcan errores** de compilación.



## Realizar operaciones sobre los datos mostrados en un informe

Si queremos la **suma de los costes** de todos los productos mostrados en el informe, y queremos que este valor se incluya en la **parte inferior del informe**, justo debajo de la **columna de los precios**, tomaremos el campo **PRODUCT\_COST** (de "Fields", en la ventana "Outline") y se coloca en **Columna Footer**. Cuando se realice esta acción aparecerá el siguiente cuadro de diálogo y se escoge **Suma**.



Ahora, cómo se puede ver en la siguiente imagen, el valor del campo **PRODUCT\_COST** aparece **precedido por una V de variable**, puesto que la creación de este tipo de cálculos **generan una variable con el campo escogido**.



El resultado final **incluirá el total de la suma**.

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**Documentación de aplicaciones: Ayudas**

**JavaHelp**

Cuando se habla de documentación de aplicaciones software, nos referimos a todos aquellos elementos que detallan las características de una aplicación, y que pueden ser necesarios para su **uso, mantenimiento o modificación**.

En concreto, se trata de ficheros de ayuda, manuales y demás guías de uso y/o mantenimiento, ya sean dirigidos para los **usuarios** de la aplicación, para **equipos** de **soporte** o incluso para otros **diseñadores**.

Para un sistema software, es especialmente importante contar con una documentación completa y que a la vez sea sencilla de consultar. Los motivos para ello son varios, por ejemplo, conseguir la mejora de la comprensión de uso de una aplicación por parte de un usuario, independientemente de su perfil.

A pesar de tratarse de un trabajo que puede resultar tedioso, se le debe otorgar toda la importancia que tiene. No es suficiente con desarrollar aplicaciones funcionales y atractivas, sino que, además, estas deben contar con una documentación adecuada. Se trata de una tarea tan importante como la que puede realizar el propio código del programa.

## Ficheros de ayuda y formatos

Un **fichero de ayuda** es un documento que contiene toda la información que puede servir de ayuda o de manual para los usuarios de la herramienta sobre la que se ha desarrollado.

Estructura general de cualquier fichero de ayuda:

Suelen estar compuestos por **dos partes claramente diferenciadas**:

- **Mapa de fichero:**  
Constituye una especie de **mapa de navegación** del sistema de ayuda, **asociando identificadores** para cada uno de los temas contenidos a las **URL** donde se encuentra el contenido relativo al tema seleccionado.
- **Vista de información:**  
Es la parte que **se muestra al usuario**, habitualmente en forma de **índice, glosario, tabla de contenidos** e incluso **buscador** de temas.

**Formatos digitales para la creación** y posterior consulta de **ficheros de ayuda**:

- **HLP**

Actualmente, este tipo de formato de fichero de ayuda **se encuentra en desuso**, fue **sustituido por CHM**. Se trata de los ficheros utilizados habitualmente para la generación de ayuda de **Windows**.

- Puede incluir **tabla de contenido** en fichero **.cnt**.
- Incluye **información extra** en fichero **.gid**.
- Utiliza ficheros **RTF** para **generar la ayuda**.
- Necesita **compilación** (por ejemplo, mediante **HTML Help Workshop**)

- CHM (Ayuda **HTML Compilado**)

Este nuevo formato incluía **ciertas mejoras sobre HLP**, por eso quedó incluido a partir de **Windows Vista**. Si bien es cierto que desde **Windows 7 solo** aparece como ayuda para **algunas** aplicaciones.

- Generado **a partir de HLP**.
- Utiliza **HTML** para generar la ayuda.
- Enlaces mediante **hipervínculos** a la tabla de contenido.
- Permite **fusionar varios ficheros** de ayuda.
- Puede ser creado a partir de **HTML Help Workshop**.

- HPJ

Los ficheros HPJ, al igual que los anteriores, utilizan:

- ficheros de tipo **.cnt** para la **representación de la tabla** de contenido,
- ficheros de tipo **.shg** para la representación de **gráficos**.

Este tipo de ficheros son creados utilizando herramientas del tipo **Help Workshop**.

- IPF (Information Presentation Facility)

Se trata de ficheros que **utilizan IPF**, un lenguaje muy similar a HTML. Son utilizados sobre todo para **ayuda en línea** e **hipertexto**.

- JavaHelp

Finalmente, encontramos este último tipo de formato de fichero de ayuda que será tratado en el desarrollo de este capítulo. Reciben este nombre tanto el **formato de ficheros de ayuda** en línea como el **sistema** que se encarga de su generación.

Este tipo de ficheros están implementados en **Java** y se utilizan para la generación de ayuda de aplicaciones desarrolladas en Java.

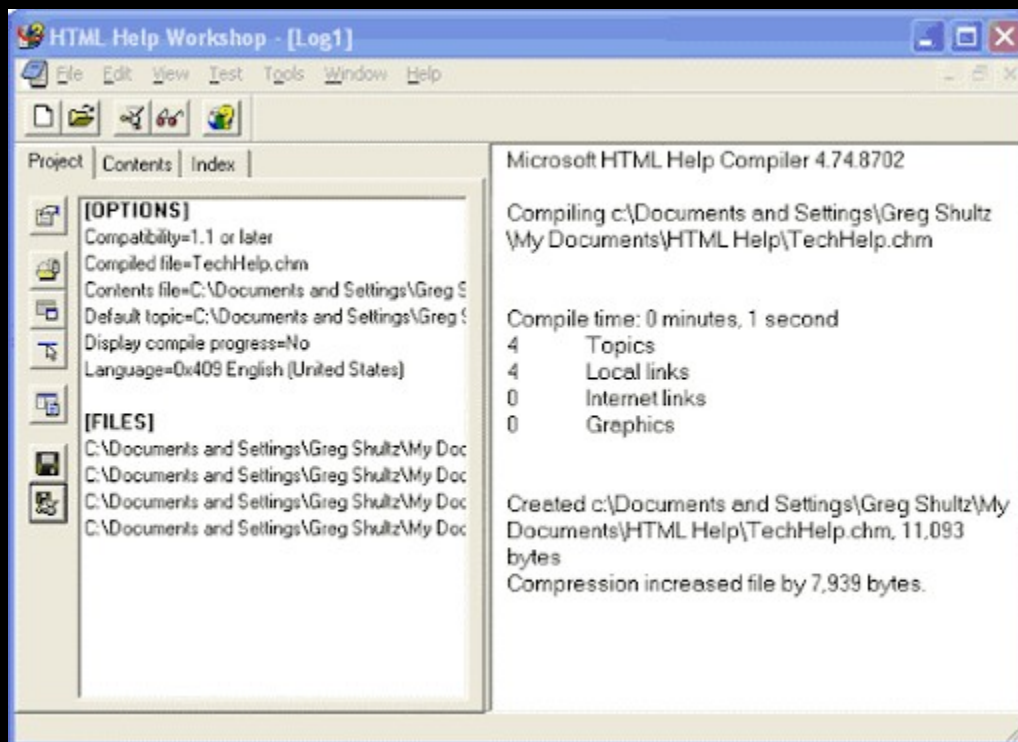
## Herramientas de generación de ayuda

### Help Workshop

Esta herramienta permite la creación de **ficheros de ayuda para Windows**. Está formada por:

- un editor de imágenes,
- el administrador de proyectos,
- el compilador, clave para el desarrollo de manuales y ayuda, puesto que permite reducir el tamaño final para su posterior distribución.

Para descargar, esta herramienta está disponible en este [enlace](#). Tras la descarga, se ejecuta el **instalador de software htmlhelp.exe**.



## Help Maker

Help Maker es una herramienta gratuita que permite la creación de ficheros de ayuda a través de un **editor de archivos**. Incluye diferentes opciones de personalización que permiten dar **formato al texto**, lo cual posibilita conseguir documentos más organizados y fáciles de leer.

Otra de las características de esta herramienta es que toda la información queda contenida en **un único fichero**, por lo que es posible **exportar** toda la ayuda en **un solo documento PDF**.

## Shalom Help Maker

Herramienta gratuita que permite la **creación de ficheros de ayuda para aplicaciones**. Al igual que en el caso anterior, este tipo de herramientas permite la creación de documentos de ayuda a través de **editores de texto** que resultan sencillos de utilizar.

Además de las características propias de un editor de texto, **permite** la creación de **índices**, contenidos **enlazados** a otras páginas, **links externos**, así como la **creación de imágenes** que enriquecen el contenido del documento final.

## Tablas de contenido

Una tabla de contenido permite reflejar la **estructura y contenido** de un documento. Se encuentra **esquemática** en varios niveles de elementos entre los que se distinguen **títulos y subtítulos**. Aunque el diseño dependerá del desarrollador y de otros muchos factores, algunas de las características típicas son:

- Pueden **mostrar el número de página o no**, en función del diseño.
- **Casi siempre** incluye un **enlace directo** en cada uno de los **títulos o subtítulos**, por lo que aparece compuesta, ya que apunta al contenido de cada uno de esos títulos.
- Se sitúan **al inicio** de cualquier documento, como si de un índice de libro se tratara.
- Su elaboración no es trivial, requiere de un **proceso de análisis completo** de la documentación que se va a exponer, así como de los títulos y subtítulos que a continuación recogen.
- **No debe duplicarse** la información.
- Los **títulos** deben ser lo **suficientemente claros** para que un usuario no necesite de otro documento para entender cómo funciona el manual de ayuda.

Las pautas de diseño de una buena tabla de contenidos son:

### Diagrama de pautas de diseño de una tabla de contenidos

1. Selección de los **temas y subtemas** que aparecerán en la tabla.
2. **Numeración** de los temas y subtemas en función del patrón de numeración escogido.
3. Creación de la tabla de contenido añadiendo los **enlaces** oportunos.
4. Constante revisión de la tabla para **actualizar y modificar** la información.



## Tipos de documentación

En función del tipo de aplicación, del tipo de usuario o de la fase concreta en la que se encuentra la aplicación, es posible realizar diferentes **clasificaciones** en cuanto los tipos de documentación se refiere.

A grandes rasgos, podemos dividir la documentación de aplicaciones software en **dos grandes grupos**:

- **Documentación de pruebas:**

Documentar las pruebas realizadas sobre un programa determinado es fundamental para detectar y corregir posibles errores. Podemos distinguir, a su vez, dos tipos:

- **Documentación de entrada:** En la que se especifican los **escenarios de prueba** y se detallan los **procedimientos** de las mismas.
- **Documentación de salida:** Se trata de los **informes resultantes** de aplicar las pruebas, definidas en el punto anterior, sobre el programa.

- **Documentación técnica:**

Pertenece a este grupo el resto de documentación: guías, hojas de especificaciones, manuales, etc. Al igual que en el caso anterior, puede dividirse en **dos grupos**:

- **Documentación interna:** Se trata de los comentarios incluidos por el desarrollador **en el código**, que deben describir distintos aspectos sobre el mismo. Se debe incidir en la importancia de realizar un programa ordenado y claro, en el que los comentarios ayuden a entender el código, pero este, de por sí, debe ser claro. En este ámbito, debe **tenerse en cuenta** lo siguiente:

- **Comienzos de módulos.**

- **Variables, constantes, procedimientos y funciones.**

- **Comentarios introductorios** sobre **bloques** de código, funciones o módulos.

- **No comentar lo obvio**, ya que el exceso de comentarios puede ser contraproducente.

- **Documentación externa:** En este caso, suele tratarse de un **manual técnico** orientado a programadores para **facilitar el mantenimiento y desarrollo** de la aplicación en un futuro.

## Tipos de manuales

Existen tipos de manuales y guías diferentes entre sí con relación al contenido de cada uno y a su manera de utilización.

### Manual o guía de usuario

El manual de usuario contiene la información necesaria para **facilitar al usuario la comprensión y utilización** del programa desarrollado en **diferentes ámbitos**:

- **formación** del usuario,
- guía de consulta ante **dudas**,
- ayuda para detectar y corregir **errores**,
- etc.

Aunque no hay ninguna norma sobre su elaboración, debe tratarse, por tanto, de un documento que sea **amigable y claro** para el usuario, ya que, al fin y al cabo, es quien va a utilizarlo. Para ello, es aconsejable el uso de **imágenes y gráficos, textos claros y concisos, ejemplos** ilustrativos, etc.

### Tutoriales visuales

Debemos de tener en cuenta al realizarlos:

1. Analizar las necesidades del público al que va dirigido.
2. Hay que ser claros y concisos. Mejor cortos y avisar del contenido al comienzo.
3. Analizar otros contenidos similares
4. Guionizar el vídeo antes de ser grabado. No basta con dar al botón de grabar.

Herramientas para captar la pantalla:

Screecast, LightShot, Animaker (nos permite crear película con imágenes) ...

Nota: A veces puede grabarse el pase de diapositivas con la propia aplicación de diapositivas, por ejemplo en Microsoft Power Point se puede lograr esto.

## Guía rápida

Puede orientarse a usuarios y/o encargados de mantenimiento, es decir, para una misma aplicación, pueden desarrollarse **varias guías rápidas** en función de la **complejidad**.

Se caracteriza por proporcionar **información muy concreta** sobre diversos procedimientos que puede realizar una aplicación. En aplicaciones muy sencillas, puede constituir **por sí misma un manual**.

## Guía de referencia

Estas suelen desarrollarse **para usuarios** con cierto conocimiento y **experiencia** en el uso de la aplicación.

Por ello, suelen contener información asociada a **aspectos más técnicos**: relación de **mensajes de error** y su posible origen, tipos de **datos de entrada** permitidos en la aplicación, **comandos** aceptados, etc.

## Manual y guía de explotación

Contienen la información necesaria para **poner en uso** la aplicación. Es decir, están muy orientados a la **instalación, configuración y puesta en marcha**, por lo que, evidentemente, irá muy ligado al contexto en el que se vaya a realizar la **instalación** (tipo de organización, requerimientos, número de usuarios, etc).

En función de la complejidad de la aplicación, puede:

- **dividirse** en Manual de **instalación** y Manual de **configuración**,
- o bien constituir un **único documento**.

## JavaHelp

### Generador de sistema de ayuda

JavaHelp es una aplicación que permite la creación de sistemas de ayuda para ser integradas posteriormente en una aplicación. Esta herramienta está orientada al uso de **aplicaciones desarrolladas con** lenguaje de programación **Java**.

Esta herramienta permite la creación de **documentos de ayuda** muy completos que incluye

- **tablas** de contenidos,
- motor de **búsqueda**,
- **glosario**,
- sección de **favoritos**...

#### Pasos secuenciados para crear un sistema de ayuda con JavaHelp.

1. En primer lugar, se ha de **dar forma** a la ayuda, lo que implica
  - **especificar la organización**,
  - **diseñar los temas** que van a tomar la ayuda.
2. Descarga e **instalación de JavaHelp**, o del sistema escogido, desde:
  - diversos repositorios,
  - la página de Sun.
3. Creación de los **ficheros de JavaHelp** necesarios en función de, o según, la organización de la ayuda determinada (según organización determinada).
4. Se construye un **fichero JAR** que incluya **todos los ficheros** y permita **mejorar su distribución**.
5. Se **añade** la ayuda a la aplicación.

## Ficheros de aplicación JavaHelp

La herramienta requiere del uso de **diferentes ficheros** de **varios tipos** para el diseño de cada una de las partes que forman la ayuda final.

- **Índice:** XML, Incluye la **distribución** del sistema de ayuda.
- **Map:** JHM, Asocia **elementos** (imágenes, ficheros HTML, etc.) del **fichero HTML** con un **identificador**.
- **HelpSet:** HS, Contiene la información necesaria para que el **sistema de ayuda se ejecute**.
- **Temas o topics:** HTML, Para crearlos, se puede utilizar cualquier herramienta para generar HTML. Contienen la **información de ayuda** como tal, debiéndose realizar uno por cada tema. Se muestra uno de ellos por pantalla. Deben tener organización **jerárquica**.
- **Base de datos de búsqueda:** Se debe utilizar la herramienta **jhindexer** para generarla.
- **Tabla de contenidos:** XML, Incluye el **contenido** de la ayuda y su **distribución**.

Tras la creación de todos los ficheros, el directorio contenedor queda de la siguiente forma:

### Estructura de directorios y ficheros en JavaHelp

```
./help
  indice.xml
  Map.jhm
  Helpset.hs
  TablaDeContenidos.xml
  help/html
    Tema1.html
    Tema2.html
./JavaHelpSearch
```

Es fundamental que los **ficheros HTML** estén en un directorio bajo **./help**, ya que, en caso contrario, la aplicación no funcionará.

## Sistema de ficheros JavaHelp, vinculación a interfaz en Java

Una vez creada la estructura del sistema de ayuda, la invocaremos desde el fichero correspondiente de java. Hay que tener claros ciertos pasos:

La estructura de las carpetas y la forma de ser invocada la estructura desde java.

La estructura de carpetas es la siguiente:

- Carpeta **bin**:
  - librerías que necesitaremos en formato .jar para implementar javahelp.
- Carpeta **src**:
  - fichero .java: con este fichero desplegamos el sistema de ayuda. Se trata de un programa habitual donde modelamos una interfaz, y el método `ponLaAyuda()`, en primer lugar carga el fichero de ayuda para que cargue el fichero `help_set.hs.`, en el que vinculábamos todo. Además crea `HelpSet helpset` y `HelpBroker hb`, y a continuación `hb.enableHelpOnButton` y `hb.enableHelpKey` hacen que aparezca la ayuda dependiendo de dónde se pulse.
- Carpeta **help** (la más importante): Aquí desplegamos el sistema de ayuda que hemos programado. Dentro de help:
  - carpeta **html**: con ficheros que implementan la ayuda, en html (p.e. `main.html`, `principal.html`, `secundaria.html`). Cada fichero puede llamar a otro dentro de la misma carpeta.
  - Carpeta **JavaHelpSearch**: se crean automáticamente sus elementos al compilar: `DOCS`, `DOCS.TAB`, `OFFSETS`, `POSITIONS`, `SCHEMA` `TMAP`.

El **resto de ficheros** (de **configuración**) dentro de help son los que nosotros vamos a implementar:

- `map_file.jhm`: mapeo entre todas las páginas e identificador de cada una de ellas.
- `toc.xml`: mapeo para la tabla de contenidos.
- `indice.xml`: modelado del índice de nuestro sistema de ayuda.
- `help_set.hs`: se vinculan todos los ficheros anteriores.

De esta manera tendríamos creado el sistema de ayuda y vinculado a nuestra aplicación en java.

## Implementación de ficheros JavaHelp

Para desarrollar cada una de las páginas o ficheros de ayuda contenidos en esta aplicación, se deben escribir tantos ficheros en HTML como se necesiten para la documentación de la aplicación. Además, son **necesarios cuatro ficheros de configuración** que se detallan a continuación:

1. **map\_file.jhm**: Este archivo se encarga de **mapear** cada uno de los HTML de ayuda creados indicando su **ruta** y un **identificador** para cada uno de estos ficheros, a través del cual serán **referenciados desde el resto de la aplicación**.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>

<map version="1.0">

    <mapID target="Principal" url="html/principal.html" />

    <mapID target="Tema 1" url="html/tema1.html" />

    <mapID target="Tema 2" url="html/tema2.html" />

</map>
```

2. **toc.xml**: Es un fichero que recoge la **tabla de contenidos y su índice**. La estructura se basa en 'tocitems' para indicar cada uno de los **elementos** en los que se organiza la tabla.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<toc version="1.0">

    <tocitem text="Principal" target="Principal"/>

    <tocitem text="Tema 1" target="Tema 1"/>

    <tocitem text="Tema 2" target="Tema 2"/>

</tocitem>

</toc>
```

3. **indice.xml**: es un fichero que recoge la **tabla del índice**. La estructura se basa en 'indexitems' para indicar cada uno de los elementos en los que se organiza el índice.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<index version="1.0">
    <indexitem text="Principal" target="Principal"/>
    <indexitem text="Tema 1" target="Tema 1"/>
    <indexitem text="Tema 2" target="Tema 2"/>
-</index>
```

4. **help\_set.hs**. Este último fichero de configuración contiene a los anteriores. En él se **describen** las secciones relativas a las **vistas**.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<helpset version="1.0">
    <title>Ayuda JavaHelp</title>
    <maps>
        <homeID>Principal</homeID>
        <mapref location="map.jhm"/>
    </maps>
    <view>
        <name>Busqueda</name>
        <label>Buscar</label>
        <type>javax.help.SearchView</type>
        <data
            engine="com.sun.java.help.search.DefaultSearchEngine">
JavaHelpSearch
        </data>
    </view>
```



```
<view>
    <name>Indice</name>
    <label>indice</label>
    <type>javax.help. IndexView</type>
    <data>Indice.xml</data>
</view>
<view>
    <name>Tabla de Contenidos</name>
    <label>Tabla de Contenidos</label>
    <type>javax.help.TOCView</type>
    <data>tablaDeContenidos.xml</data>
</view>
</helpset>
```

## Incorporación de la ayuda en Eclipse

Finalmente, se incluye la ayuda desarrollada **en la aplicación de Java** en la que va a estar contenida. En primer lugar, se deben instalar los **paquetes de JavaHelp** en el entorno de desarrollo.

- `Import java.net.*`: Permite **representar URL**, es decir, una ruta al fichero.
- `Import javax.help.*`: Permite **utilizar los ficheros** que hemos creado **desde** una aplicación **Java**. Incluye la **clase HelpSet** y **HelpBroker**, que se utilizarán posteriormente.

La inclusión de estos paquetes permitirá trabajar con las siguientes **clases y métodos**:

- Clase `HelpSet`: Posibilitará **usar los ficheros** del sistema de ayuda. **Métodos**:
  - `findHelpSet`: Devuelve la **URL del fichero HelpSet**.
  - `createHelpBroker`: Crea un objeto **HelpBroker asociado al HelpSet**.
- Clase `HelpBroker`: Permite **visualizar el contenido** de la ayuda desde la aplicación.

Principales **métodos**:

- `enableHelp`: Hace **referencia al tema o topic** que se debe visualizar al pulsar una tecla de ayuda sobre un componente determinado.
- `enableHelpKey`: Permite habilitar la **tecla de ayuda**.
- `enableHelpOnButton`: Produce que se **despliegue la ayuda** al pulsar un elemento.

## Método para vincular a interfaz la ayuda en Eclipse

```
/**  
 * Hace que el item del menu y la pulsacion de F1 visualicen la ayuda.  
 */  
private void ponLaAyuda() {  
    try {  
        // Carga el fichero de ayuda  
        File fichero = new File("../help/help_set.hs");  
        URL hsURL = fichero.toURI().toURL();  
        // Crea el HelpSet y el HelpBroker  
        HelpSet helpset = new HelpSet(getClass().getClassLoader(), hsURL);  
        HelpBroker hb = helpset.createHelpBroker();  
        // Pone ayuda a item de menu al pulsarlo y a F1 en ventana  
        // principal y secundaria.  
        hb.enableHelpOnButton(itemAyuda, "aplicacion", helpset);  
        hb.enableHelpKey(principal.getContentPane(), "ventana_principal", helpset);  
        hb.enableHelpKey(secundaria.getContentPane(), "ventana_secundaria", helpset);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

## Ejemplo sistema de ayuda con JavaHelp

### Página principal y Capítulos

#### Ficheros map\_file.jhm, toc.xml e index.xml

Los elementos contenidos en la ayuda se son:

- Página principal
- Capítulo 1, 2, 3 y 4. (Todos los ficheros de ayuda en formato .html se encuentran dentro de la ruta html/capitulos).

Se diseñan los ficheros indicando la ruta en la que se encuentra los ficheros en los que aparece la ayuda.

Fichero map\_file.jhm

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<map version='1.0'>
    <mapID target="Página Principal" url="html/principal.html" />
    <mapID target="Capítulo 1" url="html/capitulos/cap1.html" />
    <mapID target="Capítulo 2" url="html/capitulos/cap2.html" />
    <mapID target="Capítulo 3" url="html/capitulos/cap3.html" />
    <mapID target="Capítulo 4" url="html/capitulos/cap4.html" />
</map>
```

Fichero toc.xml

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<toc version="1.0">
    <tocitem text="Página" Principal="" target="Página">
        <tocitem text="Capítulo" target="Capítulo"/>
        <tocitem text="Capítulo" target="Capítulo"/>
        <tocitem text="Capítulo" target="Capítulo"/>
        <tocitem text="Capítulo" target="Capítulo"/>
    </tocitem>
</toc>
```

```
</tocitem>
```

```
</toc>
```

Fichero index.xml

```
<?xml version='1.0' encoding='ISO-8859-1'?>
```

```
<index version="1.0">
```

```
<indexitem text=""Página" Principal="" target=""Página">
```

```
<indexitem text=""Capítulo" target=""Capítulo"/>
```

```
<indexitem text=""Capítulo" target=""Capítulo"/>
```

```
<indexitem text=""Capítulo" target=""Capítulo"/>
```

```
<indexitem text=""Capítulo" target=""Capítulo"/>
```

```
</indexitem>
```

```
</index>
```

## Fichero help\_set.hs

Para la inclusión de un sistema de ayuda generado con JavaHelp, se elabora un último fichero, **help\_set.hs**, en el que se **incluyen los archivos creados** anteriormente. De esta forma, quedan **vinculados tanto el índice como la tabla de contenidos**.

Los ficheros implementados: map\_file.jhm, toc.xml y index.xml, vistos anteriormente, son los referenciados desde el archivo help\_set.hs.

Este documento relaciona todo el sistema de ayuda, incluye el índice y la tabla de contenidos.

En este documento, se diferencia claramente **dos partes**:

- **<maps>**. Esta etiqueta referencia el sistema de mapeo donde se recogen cada uno de los **HTML de ayuda creados** indicando su **ruta (location)** y un **identificador** para cada uno de estos ficheros.
- **<view>**. En este caso, se desarrollan dos vistas: una de ellas, la correspondiente al **fichero toc.xml**. Este es el fichero que recoge la **tabla de contenidos y su índice**. También se utiliza para incluir el fichero **Índice.xml**, bajo el cual se recoge la **estructura del índice**.

### Fichero help\_set.hs

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<helpset version="1.0">

    <title> Sistema de ayuda </title>

    <maps>

        <homeID> Página principal </homeID>

        <mapref location=""map.jhm""/>

    </maps>

    <view>

        <name> Índice </name>

        <label> Índice </label>

        <type>javax.help.IndexView </type>

        <data> Indice.xml </data>
```

</view>

<view>

<name> Tabla de contenidos </name>

<label> Tabla de contenidos </label>

<type>javax.help.TOCView </type>

<data> toc.xml </data>

</view>

</helpset>

## Ayuda genérica y ayuda sensible al contexto

Al instalar una aplicación solemos encontrarnos con un conjunto de manuales y guías de instalación incluidas en los ficheros de ayuda. Aunque la lectura de los mismos es más que recomendable, en la mayoría de las ocasiones sólo se recurre a éstos ante un problema en la instalación o la necesidad de realizar una configuración muy específica de la aplicación.

En cuanto al tipo de información contenida en un fichero de ayuda, cabe diferenciar entre dos bloques de información claramente distintos, hablamos de la **ayuda genérica** y la **ayuda sensible** al contexto.

- La **ayuda genérica** es el tipo de ayuda que recoge **toda la información** de la aplicación. Podríamos decir que se trata del manual completo de uso del sitio. Para localizar la información que sea de utilidad en cada momento, habrá que **navegar por el mapa del sitio**.
- La **ayuda sensible al contexto** es una ayuda **concisa sobre la funcionalidad u operación** que se está realizando en cada momento. Habitualmente, aparece **accesible** o cada vez es más habitual y aconsejable que así se haga, desde cada ventana, cuadro de texto o acción en la que se incluya. Suele aparecer enlazada en botones con el **símbolo de interrogación ?** y, en ocasiones, con la **letra i (información)**.



DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**Distribución de aplicaciones**  
**Empaquetar la aplicación**  
**Paquetes de instalación**

Toda aplicación va a requerir de un proceso de instalación, el cual puede variar en función del sistema operativo o del fichero de instalación diseñado.

Antes de proceder con la instalación, será completamente necesaria una correcta **distribución de la aplicación**. Para ello están los llamados **paquetes**, dentro de los cuales se **recoge todo el contenido requerido para una correcta ejecución** de la aplicación.

### **Empaquetado y distribución de aplicaciones**

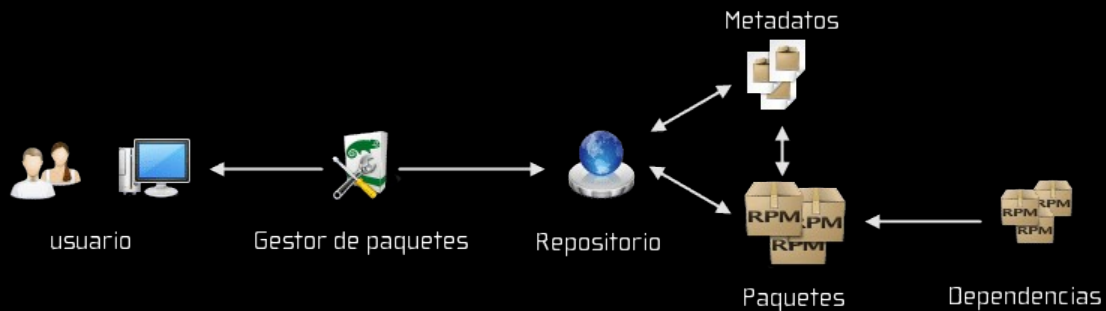
Una vez que se tiene una aplicación desarrollada y probada, es el momento de su distribución. En esta fase del desarrollo de software es importante tener en cuenta todos los **tipos de clientes finales** a los que va dirigida la aplicación, así como el **tipo de escritorio** y diferentes **sistemas operativos**.

En ocasiones la distribución será para un grupo reducido y en otros casos puede ser para miles de usuarios. En ambos casos la distribución de la aplicación consistirá en hacer llegar al usuario final el producto desarrollado.

Esta distribución debe realizarse mediante el **empaquetado de la aplicación** y por ello en este tema se verán diferentes **tipos de formatos** y las **herramientas** que permitirán llevarlo a cabo.

## Componentes de una aplicación

Los **sistemas de gestión de paquetes** permiten **automatizar** los procesos relativos a la **instalación**, **configuración** y **borrado** de los **paquetes software** en un determinado sistema.



Para poder llevar a cabo la distribución de aplicaciones, es necesaria la creación de **paquetes** en los que se incluyen todos los **componentes de diseño** de una aplicación software.

Es decir, el proceso de instalación de un proyecto requerirá de un paquete software que contenga toda la información necesaria para la ejecución de una aplicación. Por lo tanto, estos paquetes no solo contienen el código que modela el programa o aplicación, sino que está formado por **todo aquello que se necesita para desplegar** de nuevo la aplicación y que esta funcione **en cualquier otro entorno**.

Los **componentes principales** son:

- Los **ficheros ejecutables** de la aplicación.
- Las carpetas de **elementos multimedia** usados en el código de la aplicación.
- Las **bibliotecas y librerías** necesarias.

## Paquetes en Linux

En Linux, existen **varios formatos** que permiten empaquetar y distribuir aplicaciones.

- En **otros sistemas** operativos, se utilizan ciertas **herramientas de escritorio** que permiten la instalación (como **Windows Installer**).
- En **Linux**, se utilizan algunos **tipos de paquetes** que requieren de operaciones específicas a través de **línea de comandos** para su **creación**.

Algunos de los **formatos más habituales** son:

- **DEB**: usado en aquellas distribuciones que están basadas en Debian, como Ubuntu o Kubuntu. Una de las ventajas de este tipo de paquetes, a diferencia de tar o tgz, es que los de tipo deb pueden ser **instalados directamente**, mientras que los otros no lo están, han de ser extraídos en primer lugar.
- **RPM**: formato que genera la **herramienta Redhat Package Manager**.
- **TGZ**: específico de **UNIX**. Se trata de paquetes **TAR** con **compresión** a través de **GUNZIP**.
- **TAR**: paquetes **sin compresión**.

El empaquetado con formato DEB es uno de los más utilizados en este sistema operativo. La **creación de los paquetes** en Linux se basa en la ejecución de las siguientes instrucciones:

- En primer lugar, para poder realizar este tipo de empaquetado, es necesario disponer de una **distribución que permita la creación** de este tipo de paquetes, en concreto, hablamos de **checkinstall**. Para realizar esta **instalación**, se utiliza la siguiente instrucción:

```
sudo apt-get install checkinstall
```

- A continuación, **desde la carpeta** en la que se encuentran **todos los ficheros del proyecto** que se va a empaquetar, se ejecutan las siguientes **instrucciones**:

### Configuración checkinstall

```
./configure
```

```
make
```

```
checkinstall
```

- Finalmente, se **abrirá un asistente** que permite definir ciertos **parámetros** en cuanto a la creación del paquete.

Tras completar esta configuración, se pulsa el botón Enter y se **inicia la compilación del paquete .deb**, que ya estará listo para su distribución.

## **Empaquetado TAR Y TGZ**

Se describen a continuación los paquetes de tipo tar y tgz:

- **tar**: Formato utilizado para creación de **paquetes SIN COMPRESIÓN**.

La instrucción para **crear el paquete tar**:

```
tar -cvf archivo.tar carpetaContenidoParaEmpaquetar
```

Para **extraer el contenido** sería:

```
tar -xvf archivo.tar
```

c = crear un archivo.

v = detalles del funcionamiento de tar.

f = nombre del archivo contenedor.

x = extraemos el contenido.

- **tgz** o **tar.gz**: Se trata de un paquete tar pero **con COMPRESIÓN** a través de gunzip. Para crear paquete tgz, **en primer lugar se crea el paquete tar**.

Para **comprimir** el contenido::

```
gzip archivo.tar
```

En **una sola instrucción empaquetar + comprimir**:

```
tar -cvzf archivo.tar.gz carpetaContenidoAEmpaquetarComprimir
```

Para **extraer** el contenido sería:

```
tar -xvzf archivo.tar.gz
```

## Paquetes en Windows

En función del sistema operativo, existen diferentes mecanismos y formatos de empaquetado de las aplicaciones.

En el caso de Windows, podemos encontrar:

- MSI (Microsoft Silent Installer):

Se trata de un **formato para paquetes** de software que también permite la **instalación** de su contenido. Este tipo de paquetes incluyen **toda la información** necesaria para que el proceso de **instalación** se lleve a cabo de forma satisfactoria. Los paquetes de tipo MSI incluyen ficheros **.exe** que son los **instaladores** en sí.

En la **actualidad**, podemos encontrar **MSIX**, que al igual que su antecesor, es un formato para la creación y distribución de paquetes **para todas las aplicaciones de Windows**.

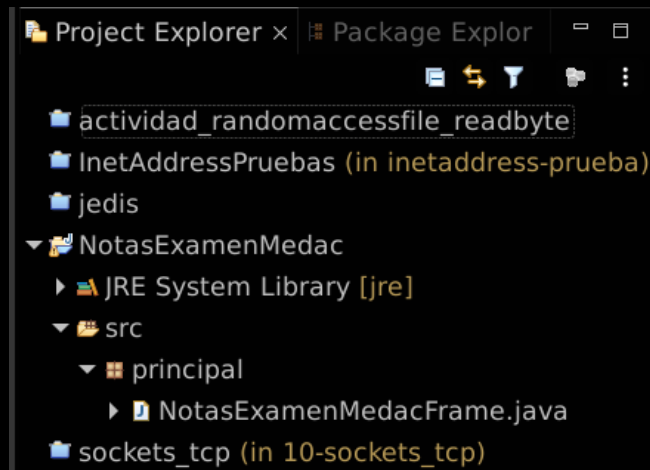
- AppX: Este tipo de método **se utiliza con menos frecuencia** que el anterior. Permite realizar el empaquetado de **aplicaciones universales de Windows**.

## Empaquetado de aplicaciones Java con Eclipse

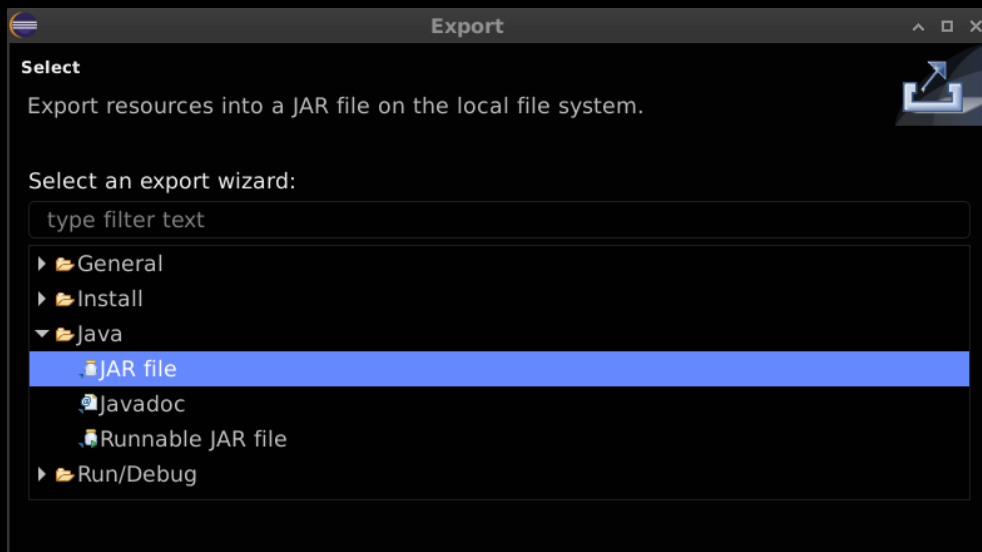
Como se ha visto en capítulos anteriores, las aplicaciones Java a través de los IDE de desarrollo como Eclipse o Netbeans permiten la creación de ficheros .jar, el empaquetado típico de los desarrollos en Java. Los pasos para **generar un jar desde Eclipse** son:

1. En primer lugar, desde la **carpeta del proyecto**, se mantendrá una estructura como la que se muestra a continuación.

### Interfaz Eclipse, selección de proyecto

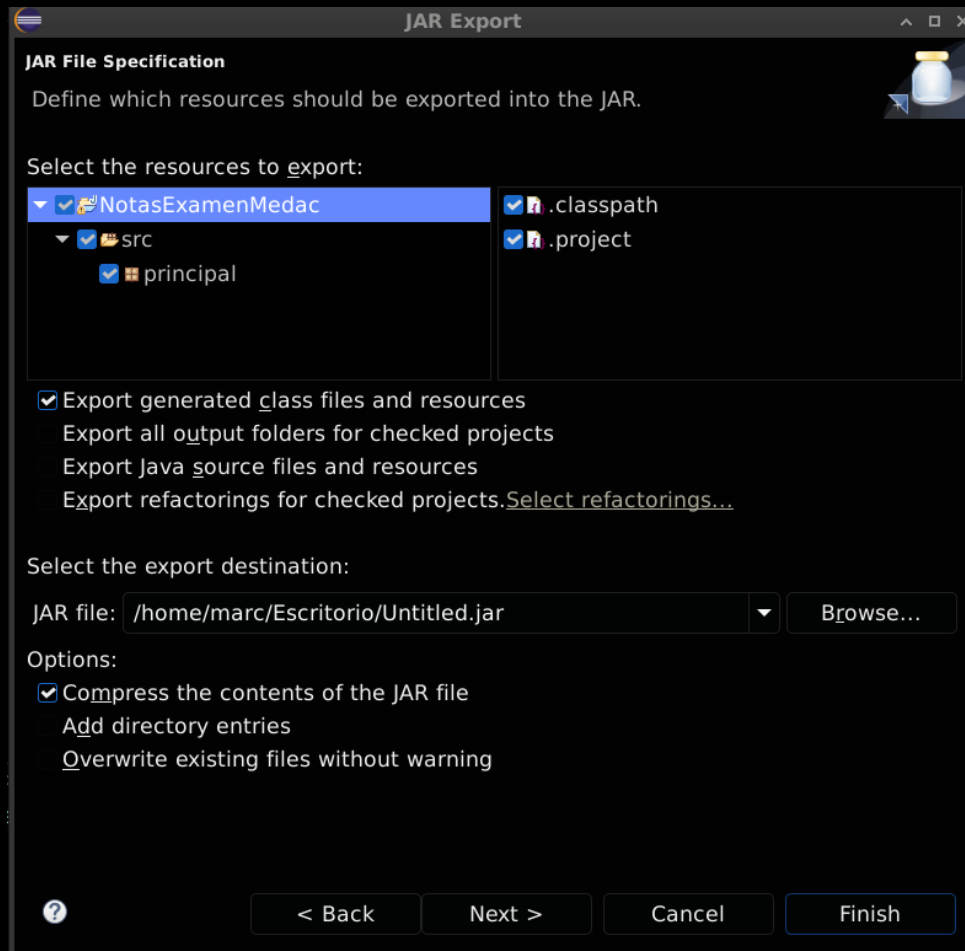


2. Con el **botón derecho** sobre el nombre del proyecto, seleccionamos la opción **Export**.
3. Se abrirá una ventana como la que sigue y se selecciona la opción **JAR File** contenida en el **menú Java**.



4. En la siguiente ventana, se selecciona todo lo que depende del proyecto sobre el que se está trabajando y se indica la **ruta en la que se colocará el fichero** resultante.

### JAR File Specification



5. Finalmente, en la ventana **JAR Manifest Specification**, se selecciona, en **Main Class**, el nombre de la **clase del proyecto** que contiene el **método main** y se pulsa Finish.



## Creación JAR

Para crear un [instalador](#), por ejemplo .exe, a través de aplicaciones tales como Launcher4J es imprescindible haber **generado un paquete jar previamente**.

Cuando se accede a la aplicación [Launcher4J](#), seleccionamos la **ruta** en la que se va a ubicar la carpeta y además indicamos el **nombre que se le va a asignar al fichero .exe**. Cuando completamos estos pasos previos de configuración lo que necesitamos es indicar en un campo, un **archivo jar**. Ahí necesitaremos **seleccionar el fichero** de este tipo situado en nuestra carpeta de desarrollo del proyecto. Por lo tanto, **imprescindible para poder crear cualquier instalador es haber creado previamente nuestro jar**.

Estamos hablando de la [distribución de aplicaciones en Java](#). ¿Cómo vamos a hacerlo?

Vamos a regresar a [Eclipse](#), y nos vamos a situar encima de cualquiera de los proyectos que tengamos abiertos.

¿Cómo vamos a [generar este fichero .jar](#)?

Es un proceso muy sencillo, pero que conviene tener en cuenta.

1. Pulsamos el **botón derecho** sobre el nombre del proyecto y en el menú que se despliega vamos a **Export**. Ahora podemos seleccionar de entre todos los tipos un fichero jar, un jar file. ¿Dónde lo vamos a ubicar? Dentro de la carpeta. Si desplegamos la carpeta Java tenemos un [jar.file](#). Lo seleccionamos y pulsamos siguiente.
2. En función de la distribución de eclipse en ocasiones no aparecen marcadas todas las casillas, aparecen marcadas unas sí y otras no. Lo aconsejable siempre es que [todas las casillas aparezcan marcadas](#). Vamos a marcar todo para crear nuestro jar. Vamos navegando por todo ello. Vamos seleccionando y estaría todo marcado.
3. Dejamos esas opciones tal y como nos aparecen y a continuación lo que vamos a elegir es [dónde se va a colocar](#). Pulsamos el explorador y en este caso lo queremos ubicar simplemente en el escritorio. Vamos a elegir el escritorio. Con el nombre que queramos. Pulsamos a guardar y a continuación finalizar.
4. En la ruta en la que lo hayamos puesto ya [tendríamos nuestro paquete jar](#).
5. Para seguir [trabajando sobre Launch4J](#), que es a lo que comentábamos al principio para crear nuestros instaladores .exe, simplemente lo vamos a colocar en la carpeta de un proyecto o vamos a recordar que lo tenemos situado en el escritorio y cuando nos pida **seleccionamos el jar** a través del cual se va a poder crear el instalador, y ya tendríamos el proceso completado.

## Instaladores y paquetes autoinstalables

Los instaladores realizan **todo el proceso** necesario para **desplegar una aplicación** software. Los instaladores llevan a cabo una serie de **operaciones sobre los archivos** contenidos en el **paquete** de distribución que permite la **instalación de cualquier software** de forma automática.

Algunos de los instaladores más conocidos son:

- **InstallBuilder,**
- **Windows Installer,**
- **MSI Studio,**
- entre otros...

Las características de este tipo de aplicaciones las convierten en unas herramientas clave para la instalación de aplicaciones.

Aunque cada sistema operativo va a presentar sus propias particularidades, el funcionamiento de los **instaladores** presenta un **algoritmo de pasos común**:

1. Se comprueban las **especificaciones de software y hardware** del equipo.
2. Se comprueba la **autenticidad** del software.
3. Construcción de los **directorios necesarios** para el **despliegue** de la aplicación.
4. Se **extraen los ficheros** del paquete de distribución.
5. Se **compilan las librerías** necesarias.
6. Se definen las **variables de entorno**.

### Windows

Formas de empaquetar las aplicaciones en Windows: son habituales EXE o MSI.

- **EXE.** Se trata de un **archivo binario ejecutable**, este tipo de instaladores es uno de los más comunes entre los usuarios. Una de las características más notables de este instalador es que permite al usuario **seleccionar** las **rutas de instalación**, así como **escoger qué componentes** de los incluidos en el paquete **se instalarán** y cuáles no.
- **MSI.** Permite la **creación de paquetes de software**, pero también permite la **instalación** de los mismos. A diferencia de los de tipo EXE, que permiten al usuario escoger entre varias opciones de configuración, MSI **realiza la instalación de forma predefinida**.

## Linux

En este apartado, vamos a analizar el proceso de instalación de un paquete .deb. Ya vimos cómo crear un [paquete deb](#) utilizando la funcionalidad checkinstall.

Este paquete será el que sea **distribuido a los usuarios**, que utilizarán la siguientes instrucciones para poder realizar la **instalación del mismo, configuración o** posterior **eliminación**:

### Instalar un paquete DEB

```
dpkg -i paquete.deb
```

### Desinstalar un paquete DEB

```
sudo apt-get remove paquete.extensión
```

### Eliminar todos los archivos descargados con la aplicación instalada

```
sudo apt-get clean paquete.extensión
```

### Eliminar los archivos de configuración del paquete instalado

```
sudo apt-get purge paquete.extensión
```

## Instalación de aplicaciones desde un servidor

La **distribución de aplicaciones** software puede realizarse **desde un servidor web**, es decir, los paquetes software pueden quedar alojados en estos servidores a los que se podrán acceder en cualquier momento para realizar la **descarga a través de un conjunto de hipervínculos**.

Para realizar este tipo de instalaciones, una de las herramientas es **AptUrl** que permiten la **descarga e instalación de paquetes** alojados en un servidor web. Desde una página web, se habilitará un hipervínculo de descarga, para ello se utiliza la siguiente sintaxis:

### Código de inserción de paquetes para descarga desde la web

**<a href="apt:nombrePaquete"> Texto enlace </a>**

Tras la **descarga** del fichero, se procede a **hacer realizar la instalación** del mismo. En función del tipo de instalador, como se vio al inicio de este tema, la instalación se realizará de diferente manera.

**EXE:** en este caso, el instalador permitirá realizar la instalación **ejecutando el propio fichero**.

**ISO:** en este tipo de archivos, será necesario utilizar un **dispositivo externo** (PenDrive, CD o DVD) para **montar la imagen ISO** y, posteriormente, poder **instalar el software**.

**DEB:** para este tipo de paquetes utilizados en Linux, será necesario el uso de un **gestor de paquetes** para la instalación, por ejemplo, **dpkg**.

## Creación de un instalador EXE

Existen multitud de aplicaciones que nos permiten crear este tipo de ficheros ejecutables, por ejemplo, para crear uno de tipo EXE, es posible utilizar **Launch4j a través de un paquete JAR**.

Para la creación de un instalador EXE a través de la aplicación Launch4j, se siguen los siguientes pasos:

1. Se crea una **nueva carpeta** en la que se almacenará el **resultado del programa**. Por ejemplo, le asignaremos el nombre de output.
2. Además, has de tener acceso a la **carpeta en la que se encuentran todos los ficheros y carpetas del proyecto** desarrollado en el IDE, en la que se encontrará:
  - el paquete **jar**,
  - las **librerías**,
  - una imagen que será utilizada para crear el **icono del ejecutable**,
  - una **imagen** que se va mostrar **antes de comenzar la ejecución** de la aplicación sobre la que se está construyendo el fichero .exe.
3. A continuación, **accedemos a Launch4j** y seleccionamos la **ruta** en la que se ha ubicado la primera carpeta, e indicamos el **nombre que se le va asignar al fichero .exe**.
4. En el **campo jar**, seleccionamos el **fichero** de este tipo situado en **nuestra carpeta de desarrollo del proyecto**.
5. En el campo **Icon**, seleccionamos la imagen que se va a utilizar como **icono del programa**.
6. Desde la **pestaña Header**, seleccionamos **GUI**, es decir, para que la aplicación se ejecute desde la **interfaz gráfica** del usuario y **no desde** línea de **comandos**.
7. En la **pestaña JRE**, indicamos la **versión de la aplicación**. Esto se hará en función del caso.
8. Finalmente, desde la **pestaña Splash**, marcamos la casilla **"Enable splash screen"** y seleccionamos la **imagen que se va a mostrar al usuario** cuando **comience** la ejecución de la aplicación. Esto es opcional.
9. Para concluir el proceso, se selecciona el botón de **"Construcción de la aplicación"**.

## La herramienta Launch4j

La herramienta **multiplataforma** Launch4j permite empaquetar aplicaciones Java .jar utilizando **ejecutables ligeros**. Permite crear **ejecutables .exe** realizando un **lanzador que trabaja junto al JAR** o envolviendo **clases del JAR dentro de un ejecutable**.

Este tipo de empaquetado proporciona una mejor experiencia de uso de la aplicación gracias a:

- El **ícono** de aplicación.
- Una **pantalla de presentación**.
- La opción de **descarga de Java** de la página en caso de que sea **necesaria su actualización**.

Además Launch4j es un software de código abierto que permite **empaquetar aplicaciones compatibles para los sistemas operativos de Windows, Linux y MacOS**.

## Interacción con el usuario

El diseño de los asistentes de instalación requiere de un conjunto de pautas que se deben tener en cuenta para su desarrollo. El desarrollo de interfaces en cuanto a la interacción entre el usuario y la aplicación ha de implementarse tras un exhaustivo análisis de la situación.

Para el desarrollo de este tipo de asistentes, se tienen en cuenta los siguientes **menús y diálogos** que se integrarán en el **asistente de instalación** para la configuración de la misma, y que permiten la interacción con el usuario.

1. En primer lugar, si la aplicación se ha desarrollado para **varios idiomas**, se muestra al usuario un menú para llevar a cabo la **elección del idioma deseado**.
2. En la actualidad, es cada vez más común que para **continuar con el proceso de instalación**, el asistente muestre la **licencia de uso de la aplicación** que el usuario ha de aceptar.
3. Existen aplicaciones que permiten al usuario **seleccionar** o bien todos, o solo algunas de las **herramientas** contenidas en el paquete. Se modela, por tanto, un **menú** que permite la selección de las mismas.
4. A continuación, se selecciona la **ruta** en la que se van a situar los archivos de la aplicación. Habitualmente, en función del sistema operativo, se utiliza una **ruta por defecto**, pero el usuario **debe poder escoger** una nueva.
5. Tras indicar todos los parámetros de configuración anteriores, comienza el **proceso de instalación**, que suele estar acompañado de **algún tipo de indicador** del porcentaje **instalado** sobre el total.
6. Cuando el **proceso concluye**, se le ha de **notificar al usuario**. En función del tipo de herramienta, puede ser necesario **reiniciar** el sistema operativo; en este caso, se ha de **preguntar al usuario** si desea hacerlo en ese momento o más tarde. Del mismo modo, también suele ser habitual habilitar una **opción** para **iniciar la ejecución** de la aplicación tras finalizar el proceso de instalación.

## Ficheros firmados digitalmente

Tal y como se indica desde el sitio web oficial, una **firma electrónica** es un **conjunto de datos electrónicos** que acompañan a un **documento electrónico** y permite **identificar al firmante** de forma inequívoca, asegurando así la **integridad del documento** firmado, entre otras.

¿Para qué es necesario este tipo de firmas en la distribución de software? En la actualidad, en muchas ocasiones, la **descarga** de nuevas aplicaciones se realiza **a través de internet**, por lo que será necesario utilizar mecanismos que garanticen la **autenticidad del software**.

Existen ciertas **herramientas específicas** que permiten el firmado digital de ficheros, y que son las más recomendables, por ejemplo,

- **AutoFirma** utilizada para **ficheros PDF**,
- **Ksi Secure** que permiten la firma de **cualquier tipo de archivo**.

En el caso de la distribución de paquetes de software, en concreto, de aquellos que se han desarrollado utilizando **Java**, sería posible realizar la **firma digital sobre los ficheros Jar**, lo que permite verificar la autenticidad del software descargado. De esta forma, cuando se va **instalar cualquier aplicación**, si se comprueba la **autenticidad de la firma**, se le **permitirá acceder a ciertos datos** que necesite para su funcionamiento.

Por ejemplo, cuando se descarga una aplicación en el teléfono móvil y esta nos pide acceso a los contactos, si se verifica la autenticidad del desarrollador, se le dará acceso a estos.

Algunas **características de la firma digital** son:

- Toda firma digital queda constituida por una **clave privada** y una **clave pública**.

**Clave privada:** es **conocida solo por el usuario dueño de la firma** y se utiliza para firmar de forma inequívoca un fichero.

**Clave pública:** es necesaria para **corroborar en el destino** que esta **firma es auténtica**.

- Para verificar que las firmas corresponden realmente con quien dicen ser, se envía de forma adicional un **certificado** en el que el **usuario afirma ser el dueño de la clave pública**. Este tipo de certificados son **emitidos por una entidad de confianza**. Por ejemplo, los certificados digitales de la **Fábrica de Moneda y Timbre**.

Ejemplo:

1. Perico envía a Juana un documento **cifrado con la clave pública** de Juana.
2. Se envía junto al documento un **certificado** de la Fábrica Nacional de Moneda y Timbre.
3. Juana **descifra el documento con su clave privada**.



## JarSigner

Para realizar la **firma de ficheros Jar**, se utiliza la herramienta JarSigner. Para entender el funcionamiento de este proceso, se deben **tener en cuenta los siguientes conceptos**:

**keystore**: se denomina así un **almacén de claves** en el cual puede haber contenidas muchas **firmas**.

**clave**: cada **par de firmas (pública y privada)** están identificadas **en el keystore** con una clave, conocida como **alias**.

**.SF: fichero de firma**. Si no especifica el nombre, utilizará las **primeras ocho letras del alias** en mayúsculas.

**.DSA: fichero del bloque de firmas**. Si no especifica el nombre, utilizará las **primeras ocho letras del alias en mayúsculas**.

Opciones JarSigner	Descripción
<b>keystore &lt;nombreAlmacen&gt;</b>	Indica el <b>fichero keystore</b> que se va a utilizar en cada caso, si no se indica, utiliza el almacén por defecto. La <b>contraseña del keystore es solicitada</b> a continuación en una nueva línea por comando.
<b>storepass password</b>	En este caso, permite <b>añadir la contraseña del keystore</b> en la <b>misma línea de comandos</b> en la que se añade el resto de la instrucción.
<b>keypass password</b>	Permite <b>añadir la contraseña del alias</b> en la <b>misma línea de comandos</b> en la que se añade el resto de la instrucción.
<b>sigfile file</b>	Permite especificar el <b>nombre de los ficheros .DSA y .SF</b> , de lo contrario, se crea utilizando el alias.
<b>signedjar file</b>	Permite especificar el <b>nombre del fichero Jar firmado</b> . <b>Si no se indica</b> , se utiliza el mismo nombre que el Jar sin firmar, quedando <b>sobrescrito</b> .

El proceso de firma utiliza el **comando JarSigner**, que recibe por **parámetro** el nombre del **archivo Jar** que se va a firmar, así como el **identificador de la clave privada** que se va a utilizar para realizar la firma: el **alias**.

A continuación, se solicitan las **contraseñas** necesarias para llevar a cabo la firma.

```
jarsigner <opciones> jar-file alias
```

## Cómo crear un asistente de instalación

### Creación de un instalador .EXE y paquete JAR

En muchos casos, para realizar la instalación de una aplicación, se muestra al usuario un asistente que permiten ir seleccionando diferentes opciones para completar la instalación, por ejemplo, la ruta, los componentes que se desean instalar, entre otros.

Para poder **crear un asistente de instalación**, es necesario **haber generado un instalador .exe**.

Una de las herramientas más utilizadas para la creación de ficheros de tipo EXE es **Launch4j**. Esta herramienta requiere del JAR correspondiente al proyecto sobre el que se está trabajando.

Desde Eclipse, escogemos alguno de los proyectos ya desarrollados a lo largo de este módulo y se **crea un paquete JAR**. Para realizar este proceso, se pulsa con el botón derecho sobre el nombre del proyecto y, a continuación, se escoge la opción Export. Seguidamente, se abrirá una ventana de selección y se escoge la opción JAR File.

El proceso completo requiere del siguiente flujo de pasos a seguir:

Para la **creación de un instalador EXE** a través de la aplicación Launch4j, se siguen los **pasos** del apartado estudiado, y de este caso práctico:

1. Se crea una **nueva carpeta** y se identifican en la **carpeta del proyecto** todos los **elementos necesarios** para la generación del instalador.
2. Se accede a **Launch4j** y se **selecciona la ruta de salida**.
3. Escogemos el **fichero jar** creado previamente.
4. Seleccionamos la imagen que se va a utilizar como **icono**.
5. Escogemos la **opción GUI** en la pestaña Header.
6. Finalmente, se selecciona el botón de **construcción de la aplicación**.

## Asistente de instalación

### Creación de un asistente de instalación

Una de las herramientas utilizadas para la creación de este tipo de asistentes es [Inno Setup Compiler](#).

**Tras la creación del instalador .exe** necesario para el desarrollo de un asistente de instalación, se describe el **proceso completo de construcción**:

1. En primer lugar, accedemos a la carpeta en la que se encuentran **todos los componentes de un desarrollo (lib, imágenes de icono, jar)**, y copiamos la carpeta de librerías y el fichero .jar. Estos elementos se colocan **en la carpeta donde se encuentra el fichero .exe** creado para este proyecto.

2. Ahora se [ejecuta](#) el software descargado, [Inno Setup Compiler](#), y seleccionamos:

**File > New > Next**

En los campos de datos colocamos:

- el **nombre** que se le va a asignar a esta aplicación,
- la **versión**,
- otros datos que se rellenarán a criterio del desarrollador.

3. En la siguiente ventana, se selecciona en el menú desplegable “[Program Files folder](#)” y se pulsa Next.

4. A continuación, seleccionamos la **ruta** donde se haya colocado el **fichero .exe**, sobre el que se está elaborando el asistente, y en el cuadro de texto que aparece más abajo, pulsamos el botón [Add folder](#), y seleccionamos la **ruta de la carpeta** en la que se encuentra el **fichero .exe**. Finalmente, pulsamos de nuevo Next.

5. Desde la siguiente ventana se permite definir la inclusión de [imágenes](#) para la creación de **iconos** personalizados. La selección de **opciones** queda a criterio del diseñador del asistente.

6. En las siguientes ventanas, seleccionamos alguna [licencia](#), si la hubiese, y se escogen las opciones de [idioma](#) oportunas para el asistente.

7. En la ventana [Compiler Settings](#), se selecciona la **ruta** en la que se va a almacenar el **ejecutable**. Desde esta misma ventana, vamos a indicar cuál es el **icono** que se va a utilizar para la **creación del asistente**. Concluimos la creación pulsando el botón Next hasta que aparezca Finish. Como respuesta a la **pregunta de si queremos compilar el script**, se pulsa **No**.

Cuando ha realizado todo el proceso anterior, aparece un **fichero con la implementación** de código generada que guardamos en la ruta deseada con el nombre que se le asigne. Ya se habría concluido el proceso, así que pulsando dos veces sobre este asistente, comenzará la instalación de la aplicación en el equipo.



## Actualizaciones

Las actualizaciones de una aplicación pueden ser debidas a:

- la voluntad propia de los **desarrolladores**,
- por reclamación de los **usuarios**,
- porque el **entorno** ha cambiado.

En este tipo de actualizaciones, se suele realizar una **evolución de la aplicación** para:

- adaptarse a un **nuevo sistema operativo**,
- adaptarse a **modificaciones en las leyes** o normas que afectan a un país.

Las bibliotecas o librerías son un conjunto de programas o archivos que se utilizarán para el desarrollo de programas informáticos. Muchas de estas bibliotecas serán específicas para un determinado sistema operativo y, por tanto, deberán de incorporarse en el desarrollo del software para poder distribuirlo correctamente.

Por otro lado, los **paquetes de software** están formados por las **bibliotecas de las que dependen los programas ejecutables** y otros **archivos necesarios** para el desarrollo correcto de las aplicaciones implementadas.

De esta manera, el usuario final puede instalar la aplicación a partir de un único archivo que contiene las diferentes carpetas necesarias y sin necesidad de realizar ninguna otra acción.

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**Pruebas de integración, regresión, volumen, estrés, seguridad y recursos.**

**Documentar pruebas**

El desarrollo de aplicaciones y sus interfaces requiere de la elaboración de un conjunto de fases, prestando especial atención a la fase de pruebas, es decir, se debe establecer una estrategia de pruebas.

En este tema, veremos todos los **tipos de pruebas** que pueden llevarse a cabo y que forman parte de una estrategia de evaluación compleja, distribuida en múltiples fases. Algunas de las pruebas más importantes son:

- las de **regresión**, puesto que permiten evaluar si al introducir cambios para solucionar errores se han **provocado otros**;
- las pruebas de **seguridad**, que permiten **garantizar la integridad** de la aplicación,
- o las pruebas **capacidad y uso de recursos**, que permiten evaluar el **comportamiento** de una aplicación ante diferentes escenarios, puesto que puede funcionar ante un usuario, pero no con diez, lo que supone un grave problema.

Sin el desarrollo de una buena estrategia de pruebas, el resultado raramente será exitoso, por eso, en este capítulo, veremos en detalle todos los **tipos de evaluaciones** que existen en la actualidad, así como algunas **herramientas útiles**.

### **Fase de pruebas en el desarrollo del proyecto**

Todas las fases establecidas en el desarrollo del software son importantes. La falta o mala ejecución de alguna fase puede provocar que el resto del proyecto tenga varios errores que serán decisivos para el buen funcionamiento del proyecto.

Una aplicación informática **no puede llegar al usuario final con errores**, y menos aún si han sido detectados previamente por los propios desarrolladores. Si se diese este supuesto, la impresión por parte de los usuarios sería de **falta de profesionalidad** y su confianza en el producto y la marca disminuiría.

Para prevenir esta situación, es importante en el desarrollo del proyecto no saltarse la fase de prueba de software antes de que llegue al usuario final.

## Objetivo y valoración del proceso de prueba

Las pruebas son una fase indispensable en el desarrollo de cualquier aplicación, herramienta o sistema, puesto que permiten inspeccionar el software desarrollado atendiendo a todos los posibles escenarios que se pueden contemplar.

Casi todos los desarrollos de software están basados en dos grupos de pruebas claramente diferenciados.

- Por un lado, las pruebas de **caja negra** en las cuales se evalúa la aplicación desde un **punto de vista externo**, es decir, sin preocuparnos del “interior”, son las habituales para la **prueba de interfaces**.
- Por otro lado, encontramos las pruebas de **caja blanca**, estas se basan en la evaluación del **código interno** del software. Un buen diseño para estas pruebas implica la evaluación de **todos los posibles caminos** que se han implementado en el diseño de un programa.

Son múltiples las **ventajas** que aportan a cualquier desarrollo el uso de un buen sistema de pruebas:

- Permiten **validar el funcionamiento** del software en base a las especificaciones de diseño.
- Permiten **detectar errores** en el software y **corregirlos** antes de la implantación del mismo.
- Un buen sistema de pruebas también permite **evaluar** el software en **distintos escenarios** probables para el uso de la aplicación desarrollada.

Podemos diferenciar entre:

- las pruebas realizadas a nivel de **módulos**,
- o las pruebas de **funcionamiento general**, que se realizan cuando todos los **módulos** del sistema se encuentren **integrados**, puesto que lo que se pretende es validar el **funcionamiento global** de la aplicación antes de ser entregada al cliente.

Para contemplar todos los posibles escenarios, es recomendable realizar la fase de pruebas desde múltiples puntos de vista, provocando **errores habituales**, o realizando pruebas de las **acciones más comunes**.



## Tipos de pruebas

Además de la clasificación genérica de pruebas de caja negra y pruebas de caja blanca, es posible dividir las fases de pruebas en diferentes tipos **atendiendo al tipo de funcionalidad evaluada** en cada caso. Algunas de las más habituales son:

- Pruebas **unitarias**:

Este tipo de pruebas evalúan **funcionalidades concretas**, examinando todos los caminos posibles implementados en el desarrollo de un **algoritmo, función o clase**.

- Pruebas de **integración**:

**Tras realizar las pruebas unitarias**, se utilizan las de integración en toda la aplicación ya totalmente integrada.

- Pruebas de **regresión**:

Este tipo de pruebas es muy común y muy importante. Implica volver a **verificar** aquello que ya había sido **evaluado previamente** y había **generado** algún tipo de **error**. La superación con éxito de este tipo de pruebas es imprescindible para validar que los **cambios han sido correctos**.

- Pruebas de **seguridad**:

Este tipo de pruebas se centran, sobre todo, en la evaluación de los sistemas de **protección y autenticación** de una aplicación.

- Pruebas de **volumen y de carga**:

En algunas ocasiones, es necesario manejar una **gran cantidad de datos**, puesto que puede que el software no soporte un **acceso masivo**, por lo que será necesario comprobar este tipo de acceso. Este tipo de pruebas son especialmente útiles en **tiendas virtuales** que pueden ver **sobrecargado el servidor** ante un gran número de accesos.

- pruebas **manuales**:

Son realizadas por un **desarrollador**, experto o no en el software a testear.

- pruebas **automáticas**:

**Utilizan herramientas** que permiten agilizar este proceso de evaluación.

## Depuración del código

La depuración de código, como su propio nombre indica, se centra en la revisión del código para la **detección de posibles errores** y la **corrección** de los mismos.

Podemos encontrar **tres tipos de errores atendiendo al desarrollo** del código:

- Errores de compilación:

Este tipo de errores se producen por un **fallo en la sintaxis** del lenguaje de programación utilizado. Dado que **no está escrito de forma correcta**, este no puede ser interpretado por completo por el compilador y devuelve este tipo de errores. Por ejemplo, si no utilizan los ';' (punto y coma) para concluir las instrucciones, se producirá un error de compilación.

- Errores de ejecución:

Este tipo de errores se producen cuando la **sintaxis es correcta**, pero se ha implementado algún tipo de operación cuyo **resultado es erróneo**. Son errores en tiempo de ejecución.

- Errores de lógica:

Por último, este tipo de errores son producidos por un fallo en el diseño de la lógica del programa, es decir, el **resultado no es el esperado**. Su detección es más compleja que los anteriores, puesto que mientras que los otros tipos devuelven algún mensaje de error, este tipo de errores no lo hace. Este tipo de errores **no devuelve mensaje alguno de error**.

El diseño de un buen programa de pruebas es clave para la detección del último tipo de errores. Los dos primeros serán detectados, habitualmente, en tiempo de desarrollo.

### Diagrama completo para depuración de código

**Resultados →**

**Localizar error →**

**Diseñar reparaciones →**

**Reparar errores →**

**Probar de nuevo el programa →**

**Casos de prueba**

## Pruebas de integración

Las pruebas de integración permiten evaluar el funcionamiento de todos los módulos desarrollados de manera conjunta. Hasta ahora es posible que solo se hayan realizado ciertos paquetes de pruebas que incluían la evaluación de los módulos y funciones como entes aislados.

Para garantizar el correcto funcionamiento de una aplicación tras la **integración de todos sus módulos**, serán necesarias las denominadas pruebas de integración.

Podemos distinguir entre **dos tipos de pruebas de integración**:

- Pruebas de integración **ascendente**:

Estas pruebas **comienzan** con la evaluación de los **niveles más bajos**. Este tipo de pruebas se realizan **en grupo** y requieren de un **controlador** que se encarga de la coordinación de las **salidas y entradas** de los casos de prueba.

- Pruebas de integración **descendente**:

Estas pruebas se realizan en el sentido inverso a las anteriores, **desde el módulo principal hasta los subordinados**.

Es habitual encontrar **dos subtipos**:

- una integración **primero en profundidad**,
- otra integración **primero en anchura**.

### **Pruebas de integración y pruebas de humo**

Las **pruebas de integración** son un tipo de pruebas de software. Estas pruebas buscan detectar posibles nuevos errores o problemas que puedan surgir **por haber introducido mejoras o cambios** en el software. Por esta razón **hay que llevar a cabo pruebas de integración al finalizar el resto de pruebas**.

Las **pruebas de humo** (ver si no echa humo...) se usan para **describir la validación de los cambios** en el software **antes de** que los cambios en el código se registren en la **documentación** del proyecto. Su origen es bastante curioso, se encuentra relacionado con la fabricación de hardware, porque antiguamente, si después de reparar un componente, éste **no “echaba humo”, significaba que el funcionamiento era correcto**.

## Pruebas de sistema

Son aquellas que evalúan **todo el sistema de forma conjunta**, integrando todas las partes, pero **sin diferenciar** las pruebas entre **módulos**.

## Pruebas de regresión

Se denominan pruebas de regresión a la nueva ejecución de un conjunto de pruebas que ya había sido ejecutado con anterioridad para **evaluar si las nuevas modificaciones y cambios** que se hayan realizado sobre el código **han podido provocar fallos** que antes no existían.

Este tipo de pruebas implica la ejecución completa de **toda la batería de pruebas o solo de algunas** concretas.

### Tipologías en pruebas de regresión:

- Pruebas de regresión locales:

En este caso, se localizan errores que se han producido por la introducción de **últimos cambios y modificaciones**.

- Pruebas de regresión desenmascarada o al descubierto (de pura chorra):

Este tipo se produce cuando la introducción de cambios muestra **errores que nada tienen que ver con las modificaciones** realizadas, pero que su inclusión los ha dejado al descubierto.

- Pruebas de regresión remota o a distancia:

Reciben este nombre los errores producidos cuando al realizar la integración de las diferentes partes de un programa, se producen errores que **de forma individual no ocurrían**.

La superación con éxito de este tipo de pruebas es imprescindible para **validar que los cambios han sido correctos**.

## Pruebas funcionales

Las pruebas funcionales se basan en la evaluación de todas las **funcionalidades especificadas en los requisitos de diseño** de una herramienta o aplicación software. Es aconsejable que el proceso quede **documentado**.

Tal y como se indica en la norma ISO 25010, son deseables ciertas **características relativas a la evaluación de la funcionalidad**:

### Características de pruebas funcionales

1. **Exactitud.**
2. **Seguridad.**
3. **Idoneidad.**
4. **Interoperatividad.**

(nemotécnica: easy, esii...)

Este tipo de pruebas son llevadas a cabo por **expertos** capaces de interpretar los resultados obtenidos y realizar **propuestas de modificaciones y cambios** necesarios para que la funcionalidad obtenida se adecúe a las especificaciones de diseño.

## Pruebas de capacidad, rendimiento, de recursos y seguridad

Este tipo de pruebas pertenecen a las llamadas **no funcionales**. Mientras que las funcionales se centran en el comportamiento interno de la aplicación, este tipo de pruebas se utilizan para **evaluar el comportamiento externo** de la aplicación. Podemos decir que las primeras son de caja blanca y las segundas, **de caja negra**.

### Tipos de pruebas

#### importante

#### Prueba de **capacidad**:

Se utilizan para la evaluación del software y su comportamiento ante un **aumento de peticiones**, es decir, ante un **incremento de la carga** de trabajo.

#### Prueba de **rendimiento**:

Se utilizan para la evaluación del **tiempo de respuesta** y la **velocidad de procesamiento** del software.

#### Prueba de **estrés**:

Se utilizan para la evaluación de la **capacidad de recuperación** del software ante una **sobrecarga** de datos.

#### Prueba de **volumen**:

Se utilizan para la evaluación de la **capacidad de procesamiento** del software ante la llegada de una **cantidad grande de datos**.

#### Pruebas de **seguridad**:

Este tipo de pruebas están diseñadas para dos claros propósitos:

- Garantizar los aspectos relativos a la **integridad** de los datos.
- Evaluar los diferentes mecanismos de **protección** que pueden estar incorporados en una determinada aplicación software.

## Pruebas manuales

Son aquellas que se realizan **por el desarrollador** para probar el código que se está implementando. En este tipo de pruebas, el propio **programador será el que introduzca los valores de entrada**, y en función de estos, se evalúa si la **salida es la esperada** al desarrollo realizado hasta el momento.

Este tipo de pruebas **no utiliza herramientas concretas**, sino que será el propio desarrollador el que lleve a cabo la **depuración** y **prueba del código** en base a la situación que se esté evaluando.

## Pruebas automáticas

Utilizan herramientas que permiten **agilizar** el proceso de evaluación de las aplicaciones implementadas. Este tipo de pruebas resultan especialmente **útiles para la realización de pruebas de regresión**, puesto que **optimizan el proceso**.

Algunas de las **herramientas más comunes** para la realización de este tipo de pruebas son:

- Jmeter:

Se trata de un proyecto de **Apache** que permite realizar **pruebas de carga** para la evaluación del **rendimiento** de una determinada aplicación.

- Bugzilla:

Herramienta **online** utilizada para el **seguimiento de errores y defectos** del software y sus módulos, a través de las diferentes **versiones** de una misma aplicación.

- JUnit:

Conjunto de **librerías en Java** utilizadas para la realización de **pruebas unitarias** sobre las aplicaciones desarrolladas sobre este lenguaje de programación, como las interfaces.

## Herramientas de pruebas automáticas, rendimiento y unitarias

<https://bit.ly/3kGKV3L>

La generación de **pruebas automáticas** es una herramienta esencial para **agilizar** la evaluación de aplicaciones ya implementadas. Estas pruebas son especialmente importantes para las **pruebas de regresión**, ya que **optimizan** la implementación de dichas pruebas:

Las pruebas de regresión permiten evaluar si los **cambios introducidos** en la aplicación para corregir errores o realizar modificaciones han generado **nuevos errores** en el desarrollo previo que **funcionaba correctamente**.

En este capítulo, nos enfocamos en tres herramientas específicas para realizar pruebas automáticas, todas comunes en el ámbito de desarrollo. A continuación, se describen brevemente estas herramientas:

### 1. JUnit:

- JUnit es un conjunto de librerías para Java, utilizado para implementar pruebas unitarias en aplicaciones desarrolladas en Java.
- Sitio web: <https://junit.org/junit5/>
- Ofrece una **guía de usuario en inglés** que explica qué es JUnit, los **tipos de soportes** necesarios para las versiones de Java, y cómo empezar con el proceso de **descarga**.
- Proporciona **proyectos de ejemplo** y explica la **creación de casos de prueba**.

### 2. Bugzilla:

- Bugzilla es una herramienta que permite realizar un **seguimiento de errores y defectos** de software de los módulos a través de **diferentes versiones** de una misma aplicación.
- Proporciona información sobre **descargas y documentación** para cada versión.

<https://www.bugzilla.org/download/>

### 3. JMeter:

- JMeter es un proyecto de **Apache** que facilita la realización de **pruebas de carga** para evaluar el rendimiento de una aplicación.
- Sitio web: <https://jmeter.apache.org/>
- Incluye una sección de **descargas y documentación**, que abarca el manual de usuario, información sobre cómo comenzar y mejores **prácticas para mejorar el rendimiento** evaluado por la herramienta.

Es fundamental destacar que la mayoría de estos recursos están en inglés. No obstante, esta característica ofrece a los desarrolladores la oportunidad de adquirir **habilidades de búsqueda de**



**información**, repositorios y guías, competencias esenciales para el desarrollo informático. Es crucial dedicar tiempo a explorar el amplio universo del desarrollo y aprender a localizar información en los sitios web de librerías y frameworks.

## Pruebas de usuario

Las pruebas de usuarios, en contraposición a las pruebas de expertos, se basan en el **análisis y evaluación** de una herramienta o aplicación software mediante un **grupo de usuarios** reales que pueden detectar errores que los **expertos no han** sido capaces de **encontrar**.

Los métodos de test con usuarios se basan en el uso de **cuestionarios** tipo. Según el **Diseño Centrado en el Usuario (DCU)**, los test de usuario se basan en pruebas que **observan la forma de interacción** de los usuarios con el producto objeto del test.

Por ejemplo, a un experto le puede resultar intuitiva y fácil de aprender la forma de uso de una determinada funcionalidad, pero no dejan de ser expertos en desarrollo. Desde el punto de vista de un usuario, esta forma de acometer la acción puede no ser tan intuitiva.

Es aconsejable que el número de usuarios que participen en este test sea de **al menos quince** para poder garantizar una tasa de **detección de cerca del 100%**. La elección de estos se debe basar en los **perfiles** hacia los que está dirigida la aplicación, no tendrá sentido probar una aplicación para la gestión logística de un almacén con un grupo de usuarios que no tienen ninguna vinculación con este tipo de áreas. Las pruebas se realizan **de forma individual** y se deben tener en cuenta todas las observaciones que se tomen, **desde la primera toma de contacto** hasta la realización de la prueba completa. Algunos **criterios de diseño** son:

- Pruebas razonables: es decir, que un **usuario real** realizará.
- Pruebas específicas: es aconsejable realizar pruebas concretas y **no muy genéricas**.
- Pruebas factibles: que **pueden realizarse**, no se trata de un examen que los usuarios no deben superar.
- Tiempo de realización razonable.

## Pruebas de aceptación

Las pruebas de aceptación se utilizan para comprobar si una aplicación, en el caso de desarrollo de software, **cumple con el funcionamiento** contenido en las **especificaciones de diseño**, tanto desde un punto de vista **funcional** como de **rendimiento**. Es importante tener en cuenta ambos aspectos, puesto que si el producto cumple con la funcionalidad deseada, pero el tiempo necesario para completarla es excesivo, no será aceptable. Igualmente, si no cumple con la funcionalidad requerida, aunque el tiempo empleado en su uso sea óptimo, tampoco cumplirá con los criterios de aceptación.

### Características de las Pruebas de Aceptación:

- Las pruebas de aceptación son definidas por los clientes: esto es así puesto que el **grado de aceptación adecuado** para un desarrollo concreto viene **determinado por los clientes** o usuarios finales del producto.
- Se ejecutan antes de la implantación final de la aplicación: este tipo de pruebas se realizan antes de ser implantadas de forma definitiva, puesto que de lo contrario, las modificaciones serán más costosas.
- Los **planes** de pruebas de aceptación han de ser correctamente documentados.

Por lo tanto, el proceso de evaluación y análisis de cualquier desarrollo antes de ser implantado de forma definitiva está constituido por una variada tipología de pruebas. La realización de todas estas es importante para ofrecer la mayor garantía posible sobre el funcionamiento de una herramienta.

En el siguiente diagrama, podemos observar uno de los **flujos habituales de ejecución de pruebas** en el desarrollo de productos software.

### Diagrama de diseño de estrategia de pruebas

#### Unitarias



## Flujo de pruebas para el desarrollo software. Diseño estrategia de pruebas

<https://bit.ly/32V0yym>

Para desarrollar y definir un buen sistema y una buena estrategia de pruebas es necesario crear diferentes tipologías en cuanto a las pruebas. No basta con una única, sino que va a ser necesario realizar diferentes tipos de pruebas, diseñar una estrategia que desde la primera vayamos pasando por el resto de pruebas, además en un orden ya establecido para poder llegar al final y entregar ese producto, subir, publicar ese producto en producción con los requisitos y especificaciones que se nos habían solicitado desde el principio.

Vamos a analizar **en qué consisten todas esas pruebas** que forman parte de una estrategia de pruebas, de esta forma vamos a establecer todos los pasos necesarios para esta estrategia.

### 1. En primer lugar hablamos de las **pruebas unitarias**:

Son las que tendríamos en primer lugar, son pruebas que van a evaluar **funcionalidades concretas**, por ejemplo vamos a tomar un fragmento de un código, una función concreta de un código, a eso se refieren las unitarias a pequeñas funciones, pequeños fragmentos y vamos a evaluar **todos los caminos posibles** que se podrían implementar, que se podrían ocurrir en el desarrollo de un determinado algoritmo, de una función o de una clase, esto es llevado a cabo **por los desarrolladores** y se trata de realizar pruebas, imprimir por pantalla, todo lo que el desarrollador puede llegar a hacer, pero a nivel de **función, de clase, de algoritmo**.

### 2. Cuando éstas se han concluido pasaríamos a las **pruebas de integración**:

Las pruebas de integración son las cuales vamos a probar de **forma conjunta** toda la aplicación ya totalmente integrada, en este caso que es lo que queremos decir, ya no vamos a probar una función aislada, sino que vamos a probar todo el funcionamiento en su conjunto, en este caso recordamos que tendríamos pruebas de integración **ascendente** y pruebas de integración **descendente**.

### 3. Además cuando continuamos las pruebas de integración pasaríamos a las **pruebas del sistema**:

En este sistema son similares a las de integración, lo que ocurre es que en el caso de integración, como vemos que tenemos ascendente y descendente, lo que hacemos es evaluar las diferentes ramificaciones de toda la funcionalidad de toda la aplicación en su conjunto, ahora cuando hablamos de las de sistema, lo que hablamos es que evaluamos el sistema de forma conjunta, integrando todas las partes pero **sin diferenciar las pruebas entre módulos**, aquí radica la **diferencia entre las pruebas de sistema y de integración**, en el caso de integración estamos probando el funcionamiento de la aplicación en su conjunto pero diferenciando por módulos y en el caso del sistema estamos probando su funcionamiento pero sin distinguir entre módulos.

4. Cuando hemos concluido estas pruebas vamos a pasar a **las de regresión**:

¿Por qué? Hasta ahora en las pruebas de unitaria, integración y sistema vamos a haber encontrado errores y lo vamos a haber cambiado, ahora necesitamos hacer las pruebas de regresión, estas pruebas lo que nos permiten es evaluar si antes los cambios que hemos realizado para solucionar ciertos errores hemos **producido nuevos errores**, en la capítulo tenéis los diferentes tipos de pruebas de regresión que podemos encontrar, esto es muy importante porque si resolvemos determinados problemas pero esa solución lo que va a hacer es generarnos nuevos problemas pues podríamos no terminar nunca, entonces hay que tener mucho cuidado, estas pruebas son imprescindibles, las de regresión son muy importantes.

5. Tenemos a continuación cuando ya hemos concluido esta, hemos resuelto errores y demás vamos a ver las **pruebas funcionales**:

Se basan en la evaluación de todas las funcionalidades específicas que estaban recogidas en los **requisitos de diseño** de la herramienta o del software, ya no estamos hablando de una evaluación en su conjunto sino que además ahora nos vamos a centrar en comprobar si todo aquello que aparecía en los requisitos en las especificaciones de nuestro cliente se está cumpliendo porque podemos haber diseñado una aplicación que funciona perfectamente pero no tiene nada que ver con lo que se había solicitado por parte del cliente, entonces ahora deberíamos comprobar si lo que se ha **especificado es lo que nosotros hemos construido**.

6. A continuación, bueno, esto sí que las podemos, se van a ver un poco de forma conjunta, hablamos de las **pruebas de capacidad**:

las pruebas de **rendimiento, de estrés, de volumen**, de recursos, por ejemplo en el caso de las pruebas de capacidad lo que nos hace es **comprobar**, evaluar si el **comportamiento** de un software es **igual de bueno ante pocas peticiones que ante muchas**, si necesitamos implementar una tienda online y funciona para 10 usuarios pero no para 100, bueno pues deberíamos, si se prevé que va a haber 100 usuarios deberíamos volver a revisar esa parte de la implementación, tenemos pruebas de rendimiento, tenemos **pruebas de estrés**, en las pruebas de estrés lo que se hace es ver cómo se va a evaluar la **capacidad de recuperación** de un software cuando se sobrecargamos con datos. Todas estas pruebas, **las de capacidad y las de recursos**, son **de forma conjunta** y casi siempre lo que vamos a evaluar en este caso es cómo responde nuestro sistema **ante muchas peticiones, ante muchos datos**, modificando todas estas casuísticas.

7. Podemos observar también las **pruebas de seguridad**:

Se van a basar en garantizar todos los aspectos relativos a la **integridad de los datos**, no podemos realizar una aplicación en la que los usuarios almacenen sus datos y estos datos queden públicos, entonces también hay que prestar mucho cuidado al tema de las pruebas de seguridad.

8. A continuación vamos a pasar a las **pruebas de usuario**:

Son las que ya **probamos la aplicación**, siempre aquí vamos a diferenciar dos tipos de usuarios, por un lado son los usuarios **expertos** que van a evaluar errores típicos en la aplicación para ver si se han implementado bien y luego también vamos a encontrar otro tipo de usuarios que van a ser más similares a los que nos vamos a encontrar ya en la implantación, no tanto desde el punto de vista del desarrollador que va a ir buscando unos errores, sino más bien del **usuario** que no tiene por qué conocer el funcionamiento de la aplicación interno y va a realizar diferentes acciones con ello, puede ser que al realizar estas funciones no habían sido valoradas por los desarrolladores, por lo tanto no se están implementando y volveríamos a necesitar modificar esa implementación de la aplicación y de nuevo volver a pasar por todo nuestro sistema de pruebas.

7. Si también superamos las pruebas de usuario, finalmente vamos a llegar a las **pruebas de aceptación**:

Concluiríamos en este paso nuestra estrategia de pruebas; en las pruebas de aceptación lo que vamos a hacer es valorar si se cumplen todas las **características propias del diseño**. ¿Qué quiere decir esto? que esté funcionando y que además se contemplen todas las características que habían sido definidas por el cliente al inicio, tendríamos un combo formado por la aplicación **funcione perfectamente** y además todo lo que en ella se puede realizar **coincida** con lo que había sido **solicitado por un usuario**.

Podemos ver a través de este diagrama que la definición de una estrategia de pruebas no es trivial, tampoco se puede realizar en 5 minutos, sino que requiere de un algoritmo de pasos al que hay que prestar especial atención.

## Versiones alfa y beta

Las pruebas alfa y beta son aquellas que permiten **encontrar aquellos errores propios en el cliente final**. Es decir, hasta ahora se han desarrollado pruebas desde el punto de vista del experto, del desarrollador o a través de los usuarios, pero siguiendo un guión pautado durante la prueba.

Existen errores que solo van a ser descubiertos cuando se simule el **funcionamiento habitual** de la aplicación, por lo tanto, en estos casos, lo que se hace es probar la herramienta al completo creando para ellos **versiones muy parecidas a las definitivas**, pero que van a servir como pruebas con las llamadas **versiones alfa y beta**.

### Versión alfa:

La versión alfa de un producto, desarrollo de herramientas o aplicaciones consiste en la **primera versión de la aplicación**. Esta será probada por un **grupo de individuos** que **simulan ser el cliente final**. Esta versión aún **no está preparada para** su implantación en **producción**, sino que debe ser evaluada previamente por un grupo de **expertos que simulan ser clientes**.

Este tipo de pruebas se realizan, habitualmente, **desde las oficinas** de trabajo donde un producto ha sido desarrollado, puesto que **aún no ha sido entregado al cliente**.

### Versión beta:

La versión beta, al igual que la anterior, es una versión casi definitiva del producto, en este caso, se trata de la **primera** versión de un producto que será **probada por los clientes** que habían realizado el encargo del desarrollo. A diferencia de la versión alfa, las versiones beta sí serán evaluadas por los clientes y otros **usuarios ajenos al desarrollo**.

Además, lo habitual será hacerlo fuera del entorno de desarrollo para evitar condicionamientos en el manejo de la aplicación por parte del grupo de desarrolladores. Aparecen los llamados **usuarios beta tester**, que son los encargados de **comprobar que todo funciona** correctamente, si no es así, también serán los que **indiquen los fallos** producidos, sobre todo **antes de hacer pública la aplicación** (cuando sería más costoso la corrección de estos fallos).

## Estrategia de diseño

**Estrategia de diseño completa** para la creación de una aplicación.

- Fase de **planificación**: En esta, se definen los **criterios y necesidades** básicas de la aplicación.
- Fase de **diseño**: Durante esta fase se realizará el diseño **en base a los criterios** definidos en el apartado anterior.
- Fase de **implementación**: En esta fase, se escribe y desarrolla el **código** diseñado en los apartados anteriores.
- Fase **evaluación**: Fase de **pruebas** para evaluar el estado de la aplicación antes de ser entregada al cliente. Al concluir esta fase, será necesario **regresar a la fase anterior**, en el caso de haber aparecido **errores o cambios**.
- Fase de **producción**: En esta fase se **entrega** el desarrollo a los **usuarios** que la utilizarán para el fin que estaba diseñada.
- Fase de **mantenimiento**: Tras ser entregada la aplicación, es posible incorporar **nuevas funcionalidades** o **resolver casuísticas** que no se habían tenido en cuenta durante su diseño e implementación.

Por lo tanto, todas las fases son necesarias para optimizar la creación de una aplicación, tanto en tiempo como en coste económico.

Si, por ejemplo, omitimos la fase de evaluación, cuando esta suba a producción, es decir, esté en funcionamiento, podrán aparecer errores que no se habían detectado al no ser evaluada, y se requerirá volver a la fase de evaluación para determinar todos los errores o cambios necesarios, y, a continuación, regresar a la fase de implementación para solventarlos, incluso a la de diseño, puesto que puede que el cambio en una parte de la aplicación, implique el cambio en otros elementos del desarrollo .



## Estrategia de pruebas

Tras describir la secuencia de fases necesarias para el desarrollo de una aplicación, nos centramos en la **fase de pruebas o fase de evaluación**, la cual es utilizada para evaluar el estado de la aplicación antes de ser entregada al cliente que solicitó su desarrollo.

Al concluir esta fase, será necesario regresar a la fase anterior en el caso de haber aparecido errores o cambios.

Se pide diseñar un plan de pruebas en el que indiques de forma estimada los tiempos que se van a asignar a cada una de ellas, puesto que siempre se va a trabajar con plazos de entrega que han de estar convenientemente acotados.

Se realiza un **listado de todas las pruebas** que se van a realizar para evaluar una aplicación.

Por ejemplo: pruebas unitarias, pruebas de integración, pruebas de sistema, pruebas de regresión, pruebas funcionales, pruebas de capacidades y recursos, pruebas de usuario y pruebas de aceptación.

Para el diseño de este calendario de tiempos, se debe tener en cuenta la **casuística de cada prueba**:

- Pruebas **unitarias**: Es conveniente que estas pruebas se realicen de **forma paralela al desarrollo** de la aplicación, por lo tanto, no se le asigna un intervalo específico durante la fase de evaluación, pero sí debe tenerse en cuenta en la **estimación del tiempo de desarrollo**.
- Pruebas de **integración**: Estas son muy importantes, ya que se evalúa el comportamiento de la aplicación **por módulos**, así que, de nuevo, sería recomendable realizarlas **en paralelo al desarrollo** de la aplicación.
- Pruebas de **sistema**: Estas pruebas evalúan **todo el comportamiento** de la aplicación de forma completa, sería conveniente dedicarle un **25% del tiempo** reservado para pruebas.
- Pruebas de **regresión**: Las pruebas de regresión evalúan si las modificaciones han creado **nuevos cambios**, y sería conveniente reservar un **15% del total**.
- Pruebas **funcionales**: Funcionamiento general en base a las **especificaciones**, en torno a un **15% del tiempo**.
- Pruebas de **capacidades y recursos**: En función del tipo de aplicación, sería necesario definir un intervalo concreto, entre **10% y el 20%**.
- Pruebas de **usuario**: Claves para cualquier desarrollo, **al menos un 20%**.
- Pruebas de **aceptación**: Se trata de volver a evaluar de nuevo la herramienta, hasta aquí debería funcionar, ahora solo debemos verificar si todo lo **especificado está implementado**, un **10% de tiempo**.

Además de la clasificación genérica de pruebas de caja negra y pruebas de caja blanca, es posible dividir las **fases de pruebas** en diferentes **tipos** atendiendo al tipo de funcionalidad evaluada en cada caso. Algunas de las **más habituales son**: pruebas **unitarias**, de **integración**, **regresión**, **seguridad**, **volumen** y **carga**.

En ocasiones, de manera errónea, se usan bucles en los que sus condiciones para su funcionamiento no tienen un final, por lo que el número de alternativas y combinaciones posibles para desarrollar las pruebas pasa a ser exponencial.

Cuanto más tiempo se tarda en detectar y solucionar un error en el código, más costes conllevará solucionarlo, y la gráfica puede llegar a ser exponencial. Además, la **mayor parte de los errores se concentran en las primeras fases** del proyecto de creación de aplicaciones, por lo que es importante **no descuidar** estas fases.

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

## **Desarrollo de interfaces en Android**

## Android Studio. Conceptos

Vamos a utilizar la herramienta Android Studio que es el entorno de desarrollo integrado oficial creado para la plataforma Android.

En primer lugar, vamos a realizar el proceso de [descarga e instalación](#) de la herramienta, que aunque es sencilla, es conveniente tener presentes las pautas esenciales de instalación.

1. Desde la página web oficial, se [descarga](#) la última versión para el sistema operativo en el que se está trabajando el desarrollo de aplicaciones e interfaces con Android.
2. Tras la descarga, se inicia la instalación del software en el equipo, **ejecutando el instalador** que se ha guardado (posiblemente en la carpeta de descargas).
3. Para completar el proceso, se escogen las diferentes **opciones de configuración** que se muestran al usuario, hasta completar el proceso pulsando el botón Finish. En función del sistema operativo, es posible que nos solicite la concesión de ciertos **permisos de seguridad**.

Tras completar el proceso de instalación, vamos a realizar un repaso sobre la **creación de aplicaciones Android utilizando Android Studio**. Al igual que para el caso del desarrollo de interfaces vistos en los capítulos iniciales de este módulo, es necesario conocer los componentes principales para optimizar su uso y, por tanto, el resultado final.

### Creación del proyecto

En primer lugar, vamos a realizar un breve repaso sobre la creación de aplicaciones Android utilizando Android Studio. Tras completar el proceso de instalación, basta con ejecutar la aplicación.

Para crear un nuevo proyecto desde cero,

1. en la página de bienvenida se selecciona la opción “**Start a new Android Studio Project**”,
2. a continuación, desde el menú File, accedemos a **New Project**.
3. Tras iniciar el nuevo proyecto, aparece la opción de **selección de la plantilla** para el proyecto. Será posible escoger desde la opción en blanco en la que se realiza el diseño desde cero hasta todo tipo de pantallas prácticamente completas que nos agilizarán mucho el diseño en algunas situaciones.
4. Finalmente, tras escoger la plantilla, se realiza la **configuración del proyecto**:

Name:

Package name:

Save location:

Language:

Minimum SDK:

Use legacy android.support libraries: (checkbox)

## Entorno de diseño y desarrollo

Tras completar los pasos anteriores, ya se habría creado el nuevo proyecto. **Por defecto**, cuando se accede por primera vez, el fichero que se muestra al desarrollador es el llamado **MainActivity.kt**.

Será necesario acceder al **fichero activity\_main.xml**, desde el cual estará disponible la **pantalla de diseño** y el código asociado a la creación de cada **nuevo componente**.

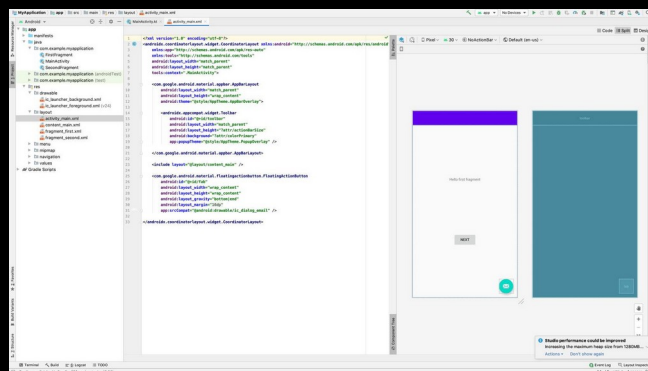
1. Desde el **menú Project** que aparece a la izquierda, se selecciona **app, res, layout** y, finalmente, **activity\_main.xml**.
2. Como se ha visto en anteriores capítulos, es posible tener una vista del código generado o de la vista de diseño.

En este caso, será posible **visualizar** el entorno de desarrollo **de tres formas distintas**:

- **Code** (se muestra el código de desarrollo),
- **Design** (aparece el lienzo de diseño sobre el que se sitúan y distribuyen los elementos),
- **Split** (una combinación de las dos anteriores). La **conmutación** entre estas tres modalidades se lleva a cabo desde la **parte superior** de la pantalla a la derecha.

El entorno de diseño queda dividido en **tres zonas de interacción** si se escoge la **opción Split**:

- **Explorador** de proyectos y ficheros (a la izquierda).
- **Zona de desarrollo** del código de implementación (zona **central**).
- **Zona de diseño** y lienzo donde se colocarán los elementos (a la **derecha**).



## Creación de un proyecto con Android

En los temas 13 y 14, abordaremos el desarrollo de interfaces para aplicaciones en Android. Ahora recordaremos cómo crear un proyecto nuevo en Android Studio para implementar su interfaz. Es necesario tener descargado e instalado Android Studio; pueden acceder al enlace disponible aquí: <https://developer.android.com/studio?hl=es-419>

Una vez instalado, ejecutaremos la aplicación.

En caso de tener un proyecto abierto, podemos crear uno nuevo desde cero yendo a "**File**", seleccionando "**New**", y luego escogiendo "**New Project**". Android Studio ofrece varias opciones, como importar proyectos, crear módulos, importar módulos, entre otras. En este caso, nos enfocaremos en la creación de un nuevo proyecto.

¿Qué nos proporciona Android Studio? Nos permite elegir diferentes plantillas predefinidas que agilizan el desarrollo de aplicaciones y sus interfaces. Por ejemplo, si queremos desarrollar una aplicación para una tienda online con una interfaz que indique la posición exacta, podemos utilizar una plantilla que implemente la actividad con Google Maps y adaptarla según nuestras necesidades.

Android Studio ya incluye plantillas para la pantalla de inicio de sesión, facilitando el desarrollo de la aplicación y aumentando la satisfacción del usuario al almacenar datos de sesión. Al elegir una plantilla, como "Basic Activity", podemos personalizarla proporcionando datos como el nombre, el lenguaje de programación (por ejemplo, Java), el package name, y finalizar el proceso de creación del proyecto.

Navegando por las carpetas, encontramos "drawable" para crear animaciones o imágenes, "layout" que define la distribución de componentes de la aplicación, y otras secciones. Es fundamental tener en cuenta el "AndroidManifest" con datos sobre la aplicación y la carpeta "Java" que contiene diferentes paquetes. En capítulos posteriores, analizaremos en detalle los principales archivos necesarios y exploraremos las distintas partes de la interfaz.

## Tipos de componentes

El diseño de interfaces de usuario centrada en el desarrollo de una aplicación móvil en Android es en un conjunto de **objetos contenedores y de objetos contenidos**. Es la combinación de estos elementos lo que permite la consecución de cualquier tipo de interfaz:

- **ViewGroup:**

Estos objetos son los **contenedores** que permiten **controlar la posición y distribución** del resto de elementos utilizados para la construcción de la interfaz de usuario de la aplicación. Se trata del **elemento clave** para lograr cualquier tipo de diseño. Un elemento ViewGroup podrá contener otros elementos ViewGroup y componentes simples de tipo View.

- **View:**

Reciben este nombre los **componentes** de la interfaz de usuario que implementan una **función concreta** sobre el diseño global de la aplicación.

Por ejemplo, los elementos View pueden ser las cajas de texto, lienzos en blanco o botones. Este tipo de elementos **no puede contener nada**, es decir, no pueden contener “en su interior” otros elementos View ni ViewGroup.

Uno de los elementos clave para el desarrollo de interfaces en Android son los **layout**. Este elemento permite la **adaptación al esquema de distribución** que van a seguir los componentes dentro de un determinado contenedor, por lo tanto, se trata de un **elemento de tipo ViewGroup**. En concreto, permiten **definir las siguientes características de diseño**:

- Posición de los elementos.
- Dimensión de los elementos.
- Distribución de los elementos.

Sin los layout, los elementos ocuparían todo el contenedor. El uso de los layout nos **permite modificar** de forma **automática**:

- **el tamaño** de los componentes,
- y su **posición**.

## Layout vs widget

Un **layout** define la **estructura visual** de una interfaz de usuario.

En Android se proporcionan mediante **lenguaje XML clases y subclases** que son muy útiles, como los widgets o los layouts.

Los **layouts** sirven para **controlar la posición de los diferentes widgets** secundarios que se puedan encontrar en el diseño. Además, sirven para **asegurar** que las **dimensiones** del widget principal son las **correctas** y que **cumple las restricciones** especificadas.

Si se crea un **layout sin un widget principal**, en cuanto el layout **se adjunte a un widget** con `setLayout()`, **se establecerá el tamaño** de ese widget principal.



## Paleta de componentes

La paleta de componentes se encuentra disponible desde la isla de diseño **Design**, desplegando la **pestaña Palette**.

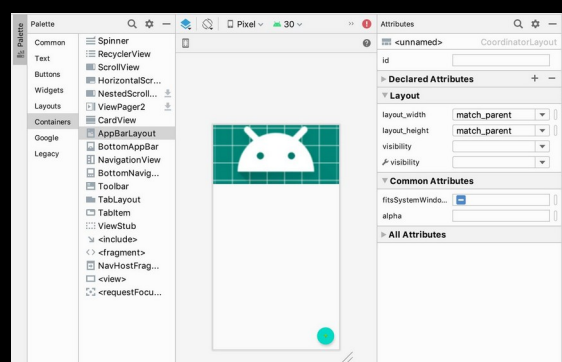
Existen diferentes tipos de componentes, organizados en **ocho grupos**:

### Tipos de componentes

1. **Common**: En este bloque, aparecen los componentes cuyo **uso es más habitual**, como cajas de texto o botones.
2. **Text**: Elementos que permiten añadir a la interfaz cadenas de texto tales como números de teléfono, fechas...
3. **Buttons**: Elementos de tipo botón o un compartimento similar (checkbox, togglebutton, radiobutton...).
4. **Widgets**: Elementos “**extra**” que permiten personalizar la interfaz de desarrollo (calendarios, barras de progreso...).
5. **Layouts**: Elementos extremadamente importantes en el desarrollo de interfaces, puesto que permiten definir la posición del resto de elementos, su dimensión y la distribución de estos.
6. **Containers**: Aunque **similares a los layouts**, estos son un tipo de View utilizado para realizar ciertas **distribuciones dinámicas**.
7. **Google**: Elementos relativos a Google, como **mapas o anuncios**.
8. **Legacy**: (No viene explicado en el tema ni aparece listado, solo en la imagen...)

A continuación, vamos a analizar algunos de los elementos más importantes para el desarrollo de interfaces en dispositivos móviles, atendiendo a su definición y al código que generan.

Cada uno de los **elementos** contenidos en estos bloques permite **alterar el valor de sus atributos**. Para ello, tras colocar un elemento en la zona de diseño, será posible **consultar** sus atributos desplegando la pestaña que aparece en la derecha y pone **Attributes**.



## Elementos: Button, TextField y TextView

### Button

Se trata de elementos de tipo botón. Estos pueden aparecer en forma de botones de texto, imagen o icono.

Suelen presentar una **acción asociada** al ser pulsados.

El código de creación que se genera al colocar un elemento de este tipo sobre el lienzo de diseño es el siguiente:

Definición de un elemento Button

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text" />
```

### TextField

Los elementos **Multiline Text** permiten al usuario introducir cualquier tipo de texto de una línea o más en una caja. Suele utilizarse bastante en el diseño de **formularios**. La etiqueta utilizada para la creación de este tipo de elementos es **<EditText>**.

Definición de un elemento TextField

```
<EditText
    android:id="@+id/editTextTextMultiLine"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:gravity="start|top"
    android:inputType="textMultiLine" />
```

## TextView

Este elemento permite mostrar un mensaje en forma de cadena de texto o similar a un usuario. Se trata de una etiqueta de texto.

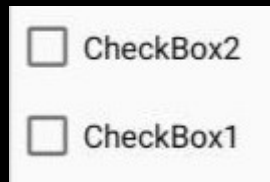
Definición de un elemento TextView

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text_view" />
```

## Otros elementos: CheckBox, ToggleButton y Spinner

### CheckBox

Este tipo de elementos permiten a los usuarios **seleccionar una o más opciones** de entre las que son mostradas por la aplicación. Los elementos checkBox se encuentran en la Palette dentro del grupo denominado **Buttons**.



La sintaxis de cada uno de los elementos que forman un grupo de checkBox es la siguiente:

Definición de un elemento CheckBox

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onCheckClicked"
    android:text="CheckBox1" />
```

## ToggleButton

Los elementos de tipo ToggleButton también se encuentran localizados en la carpeta de tipo Button. Este tipo de elementos permite al usuario **conmutar** entre dos estados.

Definición de un elemento ToggleButton

```
<ToggleButton
    android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:textOff="@string/text_off"
    android:textOn="@string/text_on" />
```

## Spinner

Este elemento es utilizado para modelar los **menús desplegables** en una aplicación con Android. Se encuentra disponible en la carpeta de elementos **Containers**.

Definición de un elemento Spinner

```
<Spinner
    android:id="@+id/spinner2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

## Layouts: RelativeLayout

En el menú de Layout de la paleta de elementos, podemos observar que existen múltiples opciones. Para añadir al diseño una de ellas, basta con seleccionar la que más se adecúa a las especificaciones del desarrollo y colocarla en el visor del lienzo central.

Existen diferentes opciones: **RelativeLayout**, **LinearLayout**, **GridLayout** o **TableLayout**, entre otras.

Cuando un elemento va a ser insertado en un layout, es necesario prestar atención a los **atributos característicos del diseño** de estas capas:

- **android:layout\_width**: determina el valor del **ancho** del elemento.
- **android:layout\_height**: determina el valor de la **altura** del elemento.
- **android:layout\_gravity**: determina la **posición** en la que debe situarse un elemento **dentro del contenedor** en el que se encuentra.
- **android:layout\_margin** (top, left, right...): determina la **distancia** con respecto a los **márgenes** del contenedor y la **alineación**.

Los atributos que determinan las **dimensiones del ancho y alto** de un elemento pueden tomar dos valores a través del menú de atributos (Attributes):

- **match\_parent**: se asigna como dimensión el **tamaño** completo del **elemento padre**.
- **wrap\_content**: se utiliza para indicar que la dimensión del elemento se **ajusta al contenido** del **mismo**.

### RelativeLayout

La distribución de elementos RelativeLayout permite colocar los elementos de manera **relativa al elemento contenedor padre**. Este layout es el que permite **mayor libertad** en cuanto a la **distribución** de los elementos.

## Más Layouts: TableLayout y GridLayout

### TableLayout:

En la distribución de tipo TableLayout, los elementos quedan distribuidos en forma de tabla, por lo que se indicarán las filas y las columnas que lo componen. Las **filas** utilizan un objeto de tipo **TableRow para ser definidas**. Se trata de una de las distribuciones más utilizadas, puesto que de una forma relativamente sencilla permite realizar una distribución de los elementos de la aplicación que **facilita la navegación del usuario** por la interfaz.

Por ejemplo, en el código de creación de este tipo de distribución que se muestra a continuación, se estaría creando una [tabla con tres filas](#):

#### Definición TableLayout

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</TableLayout>
```

### GridLayout:

La distribución GridLayout crea una rejilla formada por un conjunto de líneas verticales y horizontales en cuyos “huecos” se colocarán los elementos. La **posición de inserción** queda referenciada a través de un **índice**.

## Otro Layout: LinearLayout

El layout LinearLayout permite alinear los elementos uno tras otro en una dirección concreta (vertical u horizontal). Las **dimensiones** de cada uno de los elementos serán determinadas utilizando las propiedades destinadas a tal fin (`layout_height` y `layout_width`). El diseño genérico, si no se modifican las dimensiones, será el siguiente: Distribución LinearLayout horizontal.

En el menú Layouts, es posible encontrar dos elementos de tipo lineal, el primero de tipo horizontal, y otro de tipo vertical donde la distribución se hace en forma de filas.

## La herramienta AppBarLayout

Una herramienta clave para la programación y personalización de cualquier aplicación de desarrollo para un dispositivo móvil como Android es AppBarLayout.

Se trata de una barra que se encuentra normalmente en la **parte superior** de las aplicaciones en Android y que suele contener elementos tales como el **nombre de la aplicación**, el acceso a la configuración —normalmente representado con **tres puntos verticales**, una **caja de búsqueda**, entre otras—, y cuando se desea insertar elementos de tipo widget, la etiqueta inicial de inserción será distinta, siendo necesario indicar la ruta exacta de acceso al elemento.

## **Análisis de la herramienta e inserción de los primeros elementos**

Para comenzar con el desarrollo de nuestro proyecto, nos dirigiremos al archivo principal. En este archivo, podremos enfocarnos en la implementación de la interfaz de la aplicación. ¿Dónde podemos acceder a este archivo? Navegaremos hacia la carpeta 'App', que es la principal de nuestro proyecto. Luego, ingresaremos a la carpeta 'res', seguido de 'layout' y seleccionaremos el archivo 'Activity-Main.xml', que se desplegará a continuación.

Este archivo estará compuesto por diferentes elementos. Recordemos que previamente habíamos seleccionado una plantilla, lo que ha agregado varios elementos. Por ejemplo, elegimos una plantilla que incluye la AppBarLayout, la cual representa la zona superior con las características definidas para cada elemento, como el color de fondo, dimensiones en ancho y altura, identificador y otros atributos.

Para diseñar nuestra interfaz, añadimos elementos desde la paleta de herramientas. Al seleccionar cada tipo de elemento en la paleta, éstos aparecen, y podemos explorarlos para su inserción. Además, es crucial saber cómo visualizar tanto el código como el diseño de la interfaz simultáneamente. En la parte superior, podemos alternar entre estas vistas para trabajar de manera eficiente.

Si seleccionamos un elemento, como un botón, podemos observar cómo cambian sus atributos. Al cambiar el texto de un botón, por ejemplo, podemos ver la actualización tanto en el código como en el diseño visual.

Para personalizar aún más el diseño de nuestra interfaz, podemos especificar diferentes tipos de layout. Estos permiten distribuir los elementos en posiciones específicas. Por ejemplo, arrastramos un layout y colocamos en su interior los elementos deseados para organizarlos eficazmente.

En el código, a medida que colocamos elementos, se actualizan sus atributos, como ancho, alto y contenido. Existen diversas opciones para definir dimensiones, como usar la dimensión exacta o 'wrap-content' para ocupar solo el ancho del elemento actual.

En resumen, al comprender cómo utilizar las plantillas, paletas, layouts y visualizar simultáneamente el código y el diseño, podemos personalizar eficientemente la interfaz de nuestra aplicación de acuerdo con nuestras necesidades.



## Implementación de fragment con Maps

Vamos a implementar una pantalla de aplicación que contenga un mapa de actividad de Google Maps, indicando el código necesario en XML para el desarrollo de la misma.

Al crear un nuevo proyecto, una de las plantillas existentes permite crear una versión previa de este tipo de acciones de forma directa.

Pero si queremos crearlo desde cero, es decir, en un documento en blanco, debemos tener claro que el componente esencial será MapView, que está situado en la carpeta de la paleta llamada Google.

El código que permite crear una pantalla para incorporar elementos de actividad a través de Google Maps es el siguiente:

### Código XML Google Maps Activity

```
<fragment
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.SupportMapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MapsActivity" />
```

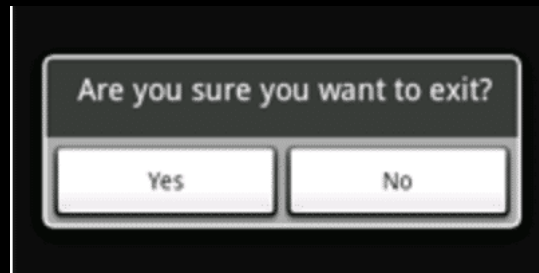
## Diseño de ventanas de diálogo para interfaces móviles con Android

El uso de ventanas emergentes en el desarrollo de interfaces móviles puede resultar un elemento esencial para mantener una comunicación activa con los usuarios, y, por tanto, mejorar la interacción y aumentar el nivel de satisfacción de los clientes de una aplicación.

Por esta razón, en este caso práctico, se pide diseñar el código Android necesario para producir una **ventana emergente** de este tipo.

**Nudo:** El cuadro de diálogo mostrará dos opciones Yes o No, como el que se muestra en la siguiente imagen.

**Desenlace:** El resultado final de implementar el código es una **interfaz** en la que aparece un cuadro de diálogo como el mostrado en la imagen:



Código Android de creación de un cuadro de diálogo

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
    .setCancelable(false)
    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            MainActivity.this.finish();
        }
    }).setNegativeButton("No", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });

AlertDialog alert = builder.create();
```

## DESARROLLO DE INTERFACES

### TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

#### Interfaces en Android

Eventos

Menús

Navigation

Animaciones

Barra de app

Imagen Asset

## Eventos

Hasta ahora, hemos analizado aspectos propios del diseño de la interfaz en cuanto al aspecto. Al igual que ocurre en el caso de las interfaces de aplicaciones de otro tipo de proyectos, la implementación relativa a la detección de eventos son determinantes para garantizar el correcto funcionamiento de una interfaz y, por tanto, la satisfacción de un usuario sobre una aplicación.

De nada sirve que el funcionamiento interno de una aplicación sea extraordinario si el diseño tanto en aspecto como en interacción con la interfaz de la aplicación no es óptimo.

La ocurrencia y posterior tratamiento de un evento se basa en la detección del mismo, habitualmente, acontecida por la pulsación del usuario sobre uno de los elementos mostrados en la pantalla del dispositivo.

La implementación para la **detección de eventos se sustenta en los siguientes pasos**, utilizando como ejemplo la detección de la **pulsación de un botón** como evento:

1. Implementar el **método `setOnClickListener`** que recibe por parámetro una **nueva instancia del objeto `ClickListener`**. Este método queda **asociado al elemento** sobre el que se focaliza la escucha, por ejemplo, un botón.
2. Se **implementa el código** que va a ser ejecutado cuando se detecta el evento. Para ello, se crea un nuevo **método que será lanzado** ante la ocurrencia del **evento**.

### Método y escuchadores

Para implementar el código de eventos, en primer lugar, se debe incluir un **elemento de escucha vinculado al método** que va a tratar la acción realizada.

Por **ejemplo**:

si el **evento** que se desea tratar es relativo a hacer **click** sobre un elemento, necesitaremos utilizar el **escuchador `OnClickListener`** y su **método `onClick`**.

De esta forma, cuando se detecta la pulsación, se **ejecutará el código** contenido en el **método `onClick()`**.

## Método tratamiento de eventos

```
private View.OnClickListener Listener1 = new View.OnClickListener() {  
    public void onClick(View v) {  
        // código a ejecutar...  
    }  
};
```

Como se puede ver en la tabla 1, existen multitud de métodos para el tratamiento de eventos y son específicos de cada caso concreto. Asimismo, cada uno de estos métodos presentará su propio escuchador.

En la siguiente tabla, se exponen algunos de los métodos de detección de eventos más comunes, y el elemento escuchador asociado:

Métodos asociados a eventos		
Método	Descripción	Evento (escuchador)
onClick()	Este método se invoca cuando se pulsa sobre un elemento de la interfaz.	onClickListener
onLongClick()	Se invoca cuando se mantiene pulsado un elemento de la interfaz.	onLongClickListener
onFocusChange()	Se invoca cuando el <b>usuario se coloca sobre el elemento</b> donde está realizando la escucha.	OnFocusChangeListener
onTouch()	Elementos “extra” que permiten <b>personalizar la interfaz de desarrollo</b> (calendarios, barras de progreso...).	OnTouchListener
onKey()	Este método se invoca cuando se pulsa sobre tecla en un teclado <b>hardware</b> físico <b>y el foco está situado sobre un elemento</b> que implementa este método en su escucha.	OnKeyListener

## Navigation

Se trata de un componente que permite implementar cualquier tipo de diseño de **navegación** a través una aplicación, aportando un alto grado de **personalización** al diseño de la interfaz. Es decir, este componente se utiliza para diseñar la secuencia de pasos en la navegación entre pantallas **de una misma aplicación**.

El funcionamiento se basa en **indicarle a NavController** la **ruta** concreta a la que se desea ir, de las recogidas en el **gráfico de navegación** u otro destino concreto. Este **destino** será mostrado en el **contenedor** llamado **NavHost**.

La implementación de este componente está **basada en tres elementos**:

- **Gráfico de navegación**: es el código **XML** en el cual queda reflejada **toda la información** de navegación.
- **NavHost**: contenedor vacío utilizado para colocar los **destinos** hacia los que apunta el **gráfico de navegación**.

Este elemento permite que los **destinos vayan modificándose** según el usuario navegue a través de la aplicación.

- **NavController**: este elemento se encarga de la **administración** de la navegación del **NavHost**.

Para la implementación de Navigation, será necesario añadir el siguiente código de **dependencias** en el fichero **build.gradle**:

### Descripción de dependencias

```
dependencies {  
    def nav_version = "2.3.0"  
    // Java language implementation  
    implementation "androidx.navigation:navigation-fragment:$nav_version"  
    implementation "androidx.navigation:navigation-ui:$nav_version"  
    // Kotlin  
    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"  
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"  
    // Feature module Support  
    implementation "androidx.navigation:navigation-dynamic-features-fragment:$nav_version"  
    // Testing Navigation  
    androidTestImplementation "androidx.navigation:navigation-testing:$nav_version"  
}
```

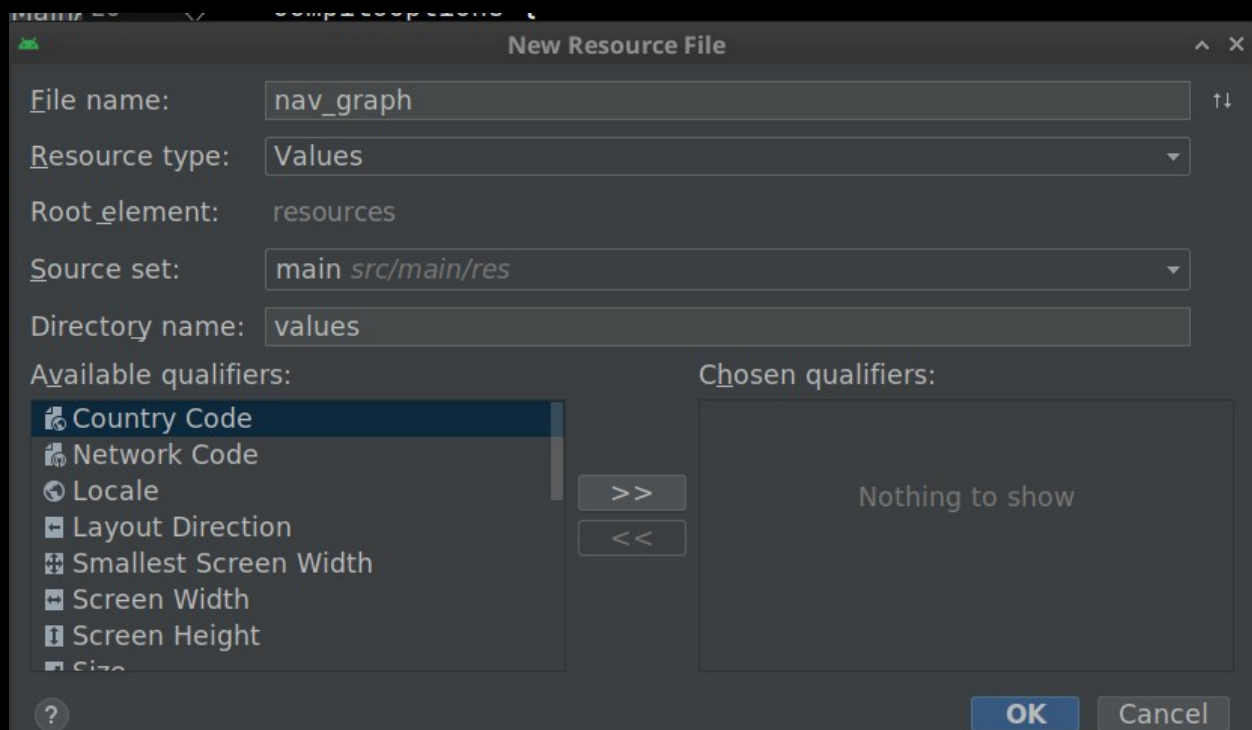
## Creación Navigation

Un gráfico de navegación es una **representación visual del conjunto de “pantallas”** que forman una aplicación y de las **acciones asociadas** a la ocurrencia de cada una de esas pantallas, que quedan representadas con una **flecha**. Ahora bien, ¿cómo se añade un gráfico de navegación?

La creación de un nuevo gráfico de navegación se lleva a cabo ubicándonos en la **carpeta res**. A continuación, se pulsa con el botón derecho sobre esta carpeta y se selecciona la opción New y New Android Resource File.

Finalmente, se introduce en la caja de texto File name **“nav\_graph”** y pulsamos OK.

### Creación de gráfico de navegación



Tras la creación, se abre por defecto el fichero **nav\_graph.xml** de la **carpeta values**, pero para poder **acceder al editor** que se analizará en el siguiente apartado, se debe localizar el fichero **nav\_graph** de la **carpeta Navigation** y ejecutarlo, es decir, hacer doble clic sobre él para que se abra el **gráfico de navegación**.

## Análisis de la interfaz

Tras la creación del gráfico de navegación, se accede al **editor de Navigation**. Como se puede ver en la siguiente imagen, la interfaz queda dividida en **tres secciones** claramente diferenciadas:

- **Panel de destinos (zona izquierda):**  
similar a los exploradores de proyectos en los que aparecen todos los ficheros que forman parte de una misma carpeta. En este caso, se muestran todos los **destinos accesibles**.
- **Graph Editor (zona central):**  
es la zona en la que se realiza la representación visual, bien en forma de **código (XML)** desde la pestaña Text, o bien de forma **visual** desde la pestaña **Design**.
- **Atributos (zona derecha):** se recogen todos los atributos del elemento que se está analizando en cada momento.

En este caso, también será posible personalizar la vista de Android Studio, **conmutando** entre las diferentes opciones que aparecen en la parte **superior derecha** de la herramienta.

En función de las **pantallas que se creen** y de las **relaciones** entre las mismas de forma gráfica, se **creará el código XML** asociado. Para acceder a este código, se selecciona la opción Code.

Los principales **elementos** que se observan en el **código generado** son:

- **navigation:** es el **elemento principal** de un fichero XML bajo el que aparecen el resto de elementos (pantallas como fragment y relaciones como action).
- **fragment:** representa cada una de las **pantallas** colocadas en el editor.
- **action:** indica el flujo de relaciones que existen entre las pantallas.

En el siguiente código, se muestra una primera pantalla y la relación de esta con la segunda:

### XML Pantalla + acción

```
<fragment
  android:id="@+id/FirstFragment"
  android:name="com.example.myapplication."
  tools:layout="@layout/fragment_first"
  android:label="@string/first_fragment_label">
  <action
    android:id="@+id/action_FirstFragment_to_SecondFragment"
    app:destination="@id/SecondFragment" />
</fragment>
```



## Creación de un nuevo esquema de pantallas en Navigation

Cuando se crea un **nuevo gráfico de navegación**, este aparece **configurado por defecto** como se muestra en la figura del apartado anterior. Habitualmente con dos elementos de tipo fragment y la relación que une a ambos. Ahora bien, un desarrollador podrá **modificar este esquema** todo lo necesario, creando así una **interfaz propia de la aplicación** y mejorando la experiencia de navegación de los usuarios de la misma.

Para **añadir nuevas pantallas**, es posible hacerlo a través del **código XML** o del **editor gráfico**.

Si se realiza de la primera forma, bastará con insertar el siguiente código donde se indica el nombre de la nueva pantalla.

Nueva pantalla-fragment

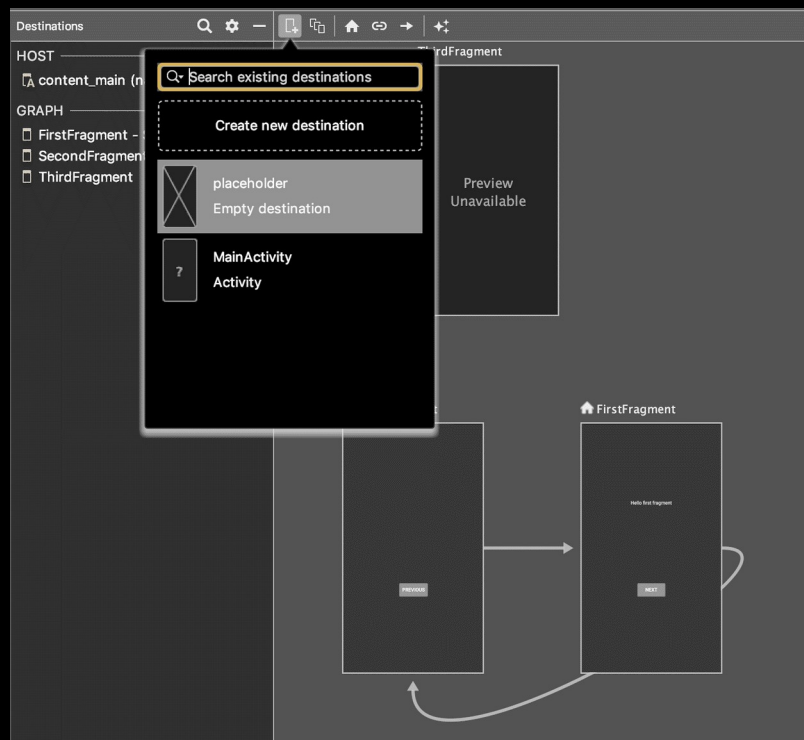
```
<fragment
    android:id="@+id/ThirdFragment"
    android:name="com.example.myapplication.SecondFragment">

</fragment>
```

Si se desea hacer de **forma gráfica**, una vez **seleccionado el tipo de pantalla** de entre las que se muestran, se pulsa el botón con un rectángulo y un símbolo + de color verde.

En cualquiera de los casos, el resultado será como el que se muestra en la siguiente imagen:

Creación de pantallas



En este segundo caso, el código XML generado solo incluye un atributo id.

Código inicial de creación de pantalla

```
fragment android:id="@+id/placeholder" />
```

Para añadir un nombre específico, basta con añadir el atributo name:

```
android:name="com.example.myapplication.NombreDelFragment".
```

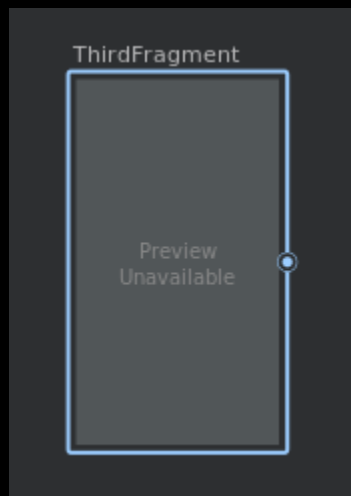
## Creación de conexiones entre pantallas

Como se ha visto anteriormente, las pantallas quedan conectadas mediante **relaciones**, es decir, la acción efectuada sobre una determinada interfaz tiene **como resultado una nueva**, ya sea de tipo específico o general (**como el regreso a la pantalla de inicio**).

Por lo tanto, las **acciones representan las rutas** que se van a seguir en la aplicación producto de la acción del usuario sobre la misma. A continuación, se indican varias formas para crear estas conexiones, ahora bien, debemos recordar que se está **creando la conexión**, pero el **código** que **implementa la acción del cambio** de una pantalla a la otra se realiza a través de los **ficheros .java**:

1. La **primera opción** consiste en seleccionar el punto que aparece en el borde de la pantalla y arrastrar la **flecha hasta el destino**. Esta opción resulta adecuada cuando el **número de pantallas es relativamente pequeño** y quedan todas a la vista.

Pantalla + elemento de conexión



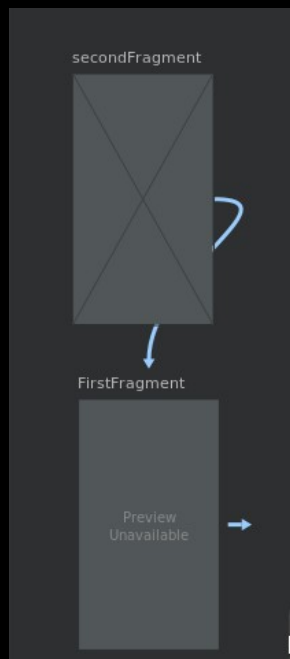
2. La **segunda opción** es pulsando con el **botón derecho sobre la pantalla origen** que va a ser conectada y se selecciona **Add Action**. En el menú, se indica el **nombre de la conexión** y **Destination** muestra en una lista de valores todos los **posibles destinos**.

### Add Action configuración

ID	SecondFragment
From	ThirdFragment
Destination	None
Transition	None
Enter	← Source
Exit	ThirdFragment (Self)
Pop Enter	Root
Pop Exit	FirstFragment placeholder2
Pop Exit	None
Pop Behavior	
Pop To	None
Inclusive	<input type="checkbox"/>
Launch Options	
Single Top	<input type="checkbox"/>
<div>Add Cancel</div>	

Cualquiera de las dos opciones tiene como resultado la **conexión de pantalla ThirdFragment con SecondFragment**:

### Add Action configuración



## Transacciones entre pantallas con animación

El manejo de Navigation requiere de práctica, por lo que se aconseja visitar el sitio oficial de desarrollo de Android para seguir ampliando y actualizando el contenido sobre sus componentes y el resto de los utilizados por Android.

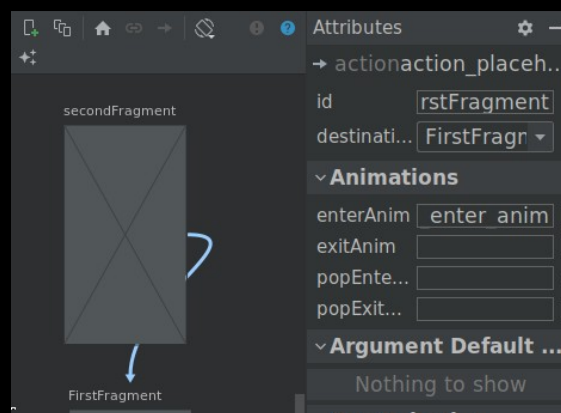
Para concluir este bloque, veremos algunas propiedades interesantes que dotan de cierta animación a la transición entre pantallas:

Propiedades de animación	
<b>app:enterAnim</b>	Animación asociada a la <b>entrada en una pantalla</b>
<b>app:exitAnim</b>	Animación asociada a la <b>salida de una pantalla</b>
<b>app:popEnterAnim</b>	Animación asociada a la <b>entrada</b> en una pantalla a través de una <b>acción emergente</b>
<b>app:popExitAnim</b>	Animación asociada a la <b>salida</b> de una pantalla a través de una <b>acción emergente</b>

Para indicar el tipo de animación que va a presentar cada relación, se debe seleccionar y, a continuación, en el **menú Atributos**, escoger la acción sobre la que se va a indicar un nuevo tipo de animación para, después, pulsar el **botón que aparece a la derecha**.

En el cuadro de diálogo que aparecerá, se debe **seleccionar el tipo de animación** que va a presentar la acción.

Atributo de animación



Desde la pestaña de Text (Code), se **actualizará automáticamente el código** añadiendo los nuevos atributos:

Código de animación de relación

```
<action  
    android:id="@+id/secondFragment"  
    app:destination="@id/FirstFragment"  
    app:enterAnim="@anim/nav_default_enter_anim" />
```

## Creación de un componente Navigation. Análisis de la interfaz y análisis del código

Navigation es un componente que nos va a permitir implementar cualquier diseño de navegación por una aplicación personalizándolo hasta el nivel en el que nosotros así lo queramos y por lo tanto nos estamos centrando en diseñar las interfaces relativas a la programación, no tanto la funcionalidad que eso se ocuparía en otro momento, en otros módulos, sino que aquí lo que nos centramos es en el **diseño de la propia interfaz** que al final va a ser un elemento clave para que el usuario quiera seguir utilizando nuestra aplicación y que además esa experiencia de navegación sea óptima.

En primer lugar, vamos a saber cómo llegar a crearnos una interfaz en la que nos aparezcan las pantallas, donde nos aparezca el sistema de Navigation.

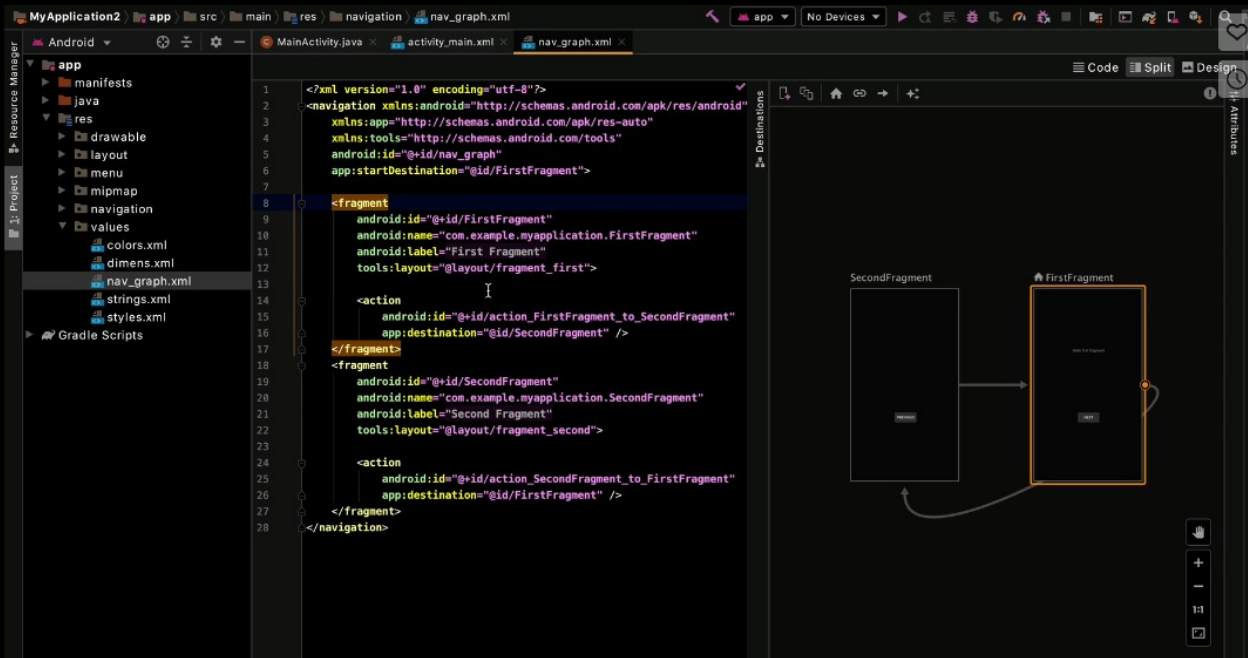
Lo que tenemos que hacer es desde la **carpeta del proyecto**, desde RES, pulsamos el botón derecho, ahora New y a continuación lo que vamos a seleccionar es Android Resource File. Vemos que nos va a aparecer esta pantalla que tenemos por aquí, lo que tenemos que pulsar es lo siguiente nav\_graph y pulsamos OK. Y como vemos, pues nos habría abierto el programa.

Ya lo tendríamos implementado de antes...

Lo que nos aparece por defecto siempre es esta pantalla que tenemos por aquí. Muy importante, puede que al abrirlo la primera vez solo nos aparezca el código y al igual que ocurría, veíamos en el capítulo anterior, si pulsamos a Design nos va a aparecer solamente la parte más gráfica y si queremos ver código y parte gráfica lo que hacemos es dividir la pantalla. A este lado nos quedaría el código en XML que define la interfaz y aquí lo que vamos a tener colocados son las diferentes pantallas.

Estas serían las partes clave. Vamos a distinguir también para que nos quede claro cuáles son los elementos. Aquí equivale un elemento de la derecha con su respectivo elemento de la izquierda. ¿Qué hacemos? Lo pulsamos cuando este queda marcado de color azul. Es el código que además se nos resalta al mismo tiempo en XML. Este es fragment. **Cada una de las pantallas aparecen identificadas inequívocamente** con un fragment y vemos que dentro de este fragmento de código vamos a tener diferentes elementos.

Por ejemplo, vamos a tener una ID, una forma de identificar como si fuese el nombre de una variable, identificar cada una de las pantallas. En este caso es secondfragment. Si nos venimos aquí marcaría el primer código. Este sería el firstfragment.



¿Qué más tenemos? La etiqueta. Esto es lo que nos aparece por aquí. Lo que nos aparece por aquí sería este nombre. Al pulsar dos veces lo que nos indica es que en este caso es una etiqueta que recibe el nombre de secondfragment y que es de tipo string. ¿Cuál es el contenido? El que nos aparece aquí arriba. Este es el nombre que aparece, el nombre que se le asigna a cada una de las etiquetas de las pantallas.

¿Qué nos quedan? Las **acciones**, las relaciones. ¿Cómo queda unida una ventana con la otra? Con este sistema de navegación lo que vamos a ir haciendo es definiendo esa **ruta que va a seguir un usuario en capas**. Imaginamos que accedemos a una aplicación, encontramos una primera pantalla de inicio. Imaginamos que es una tienda de ropa. Pulsaríamos la primera opción y escogeríamos la tipología de la ropa faldas. Entraríamos un nivel hacia abajo. Nos llevaría a otra pantalla. A continuación seleccionamos dentro de las faldas si son largas o son cortas. ¿Qué es lo que hace esto? Cuando lo pulsamos nos va a llevar a otra pantalla.

De esta forma lo que estamos construyendo son **sistemas en profundidad** que van a dar al usuario la sensación de estar el mismo diseñando su paseo por la aplicación. En este caso, equiparable a un paseo por una tienda física.

¿Cómo representamos la **unión entre cada una de las pantallas**? Aquí lo tenemos, **action**. Es lo que nos va a definir cómo conectar una con otra. Por ejemplo, en el caso de SecondFragment vendríamos a este fragmento de aquí y lo que nos dice es que se define una relación, que es esta horizontal, que cuando se realiza determinada acción sobre la pantalla número 2 nos va a llevar a la primera. En este caso, que es el que se nos crea por defecto, de nuevo si escogiéramos la FirstFragment vemos que tenemos una acción que nos relaciona, nos vincula directamente esta ventana, esta pantalla, con la segunda.



## Animaciones

Las animaciones, al igual que ocurre con el diseño web, aportan un alto grado de **dinamismo** a la funcionalidad de la aplicación y, además, permiten informar al usuario del **estado de la petición**, por ejemplo, para indicar que una petición se está **procesando** o que se está **autenticando** a un usuario en la aplicación.

En primer lugar, es necesario crear la animación, y para ello se utiliza [AnimationDrawable](#), en concreto, se implementa utilizando un elemento [animation-list](#), que permite cargar todos los **fotogramas** que formarán parte de la animación.

### Código de elemento de animación

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item
        android:drawable="@drawable/rocket_thrust1"
        android:duration="200" />
    <item
        android:drawable="@drawable/rocket_thrust2"
        android:duration="200" />
    <item
        android:drawable="@drawable/rocket_thrust3"
        android:duration="200" />
</animation-list>
```

El atributo [oneshot](#) permite indicar si la animación se reproduce **una sola vez (true)** o de forma **indefinida (false)**.

Los **códigos de creación de animaciones** se colocan en la ruta **res/drawable/**.

Para ser **utilizados**, serán llamados desde el **código Java** correspondiente.

Código de animación ejecutada al pulsar la pantalla

```
AnimationDrawable rocketAnimation;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ImageView rocketImage = (ImageView) findViewById(R.drawable.rocket_image);
    rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
    rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
    rocketImage.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            rocketAnimation.start();
        }
    });
}
```

## El sistema de color de Material Design

Para crear una **marca de seña de identidad** es importante tener un sistema de color que lo identifique. Con el sistema de color de Material Design se pueden utilizar **paletas de colores** para la creación de **temas de color**, así como herramientas específicas para la **selección del color adecuado**.

El sistema de color de Material Design permite aplicar color a la interfaz de usuario de una manera muy intuitiva. La clave de este sistema consiste en seleccionar un **color primario** y uno **secundario** para representar la marca. Además, existen **variantes oscuras o claras** que se pueden aplicar sobre el color seleccionado.

Los temas de color están diseñados para presentar un **aspecto armónico**, sin grandes excentricidades, garantizando que el **texto sea accesible** y se pueda **distinguir elementos y superficies** de la interfaz de usuario entre sí.

## Agregar un barra de app

Uno de los elementos presentes habitualmente en cualquier aplicación es la **barra de acciones o barra de app**. En el capítulo anterior ya se expuso cómo colocar un elemento de este tipo sobre la interfaz, pero ahora analizaremos en profundidad su comportamiento y la personalización posible de la misma.

Las barras de acciones tienen un **espacio limitado**, por lo que solo se podrán colocar algunos **accesos rápidos**. La elección de estos resultará clave para mejorar la **experiencia de navegación** del usuario.

Para **incorporar nuevos botones a esta barra**, se debe crear un **nuevo archivo** en la ruta **res/menu** y, a continuación, crear un **nuevo elemento ítem** para **cada uno de los componentes** que se quieran incluir en la barra.

Se debe prestar especial atención al *atributo* **showAsAction**, puesto que indica **cómo quedaría el icono** de acción representado:

- **ifRoom**: queda representada **con un icono si hay espacio** suficiente y en el menú ampliado si no lo hay.
- **never**: por el contrario, si se indica directamente el valor never, **solo será mostrado el elemento en el menú ampliado** (se accede a través de los **tres puntos en vertical**).

En el siguiente código, es posible ver un elemento con cada uno de los valores indicados.

### Creación y configuración de la barra de app

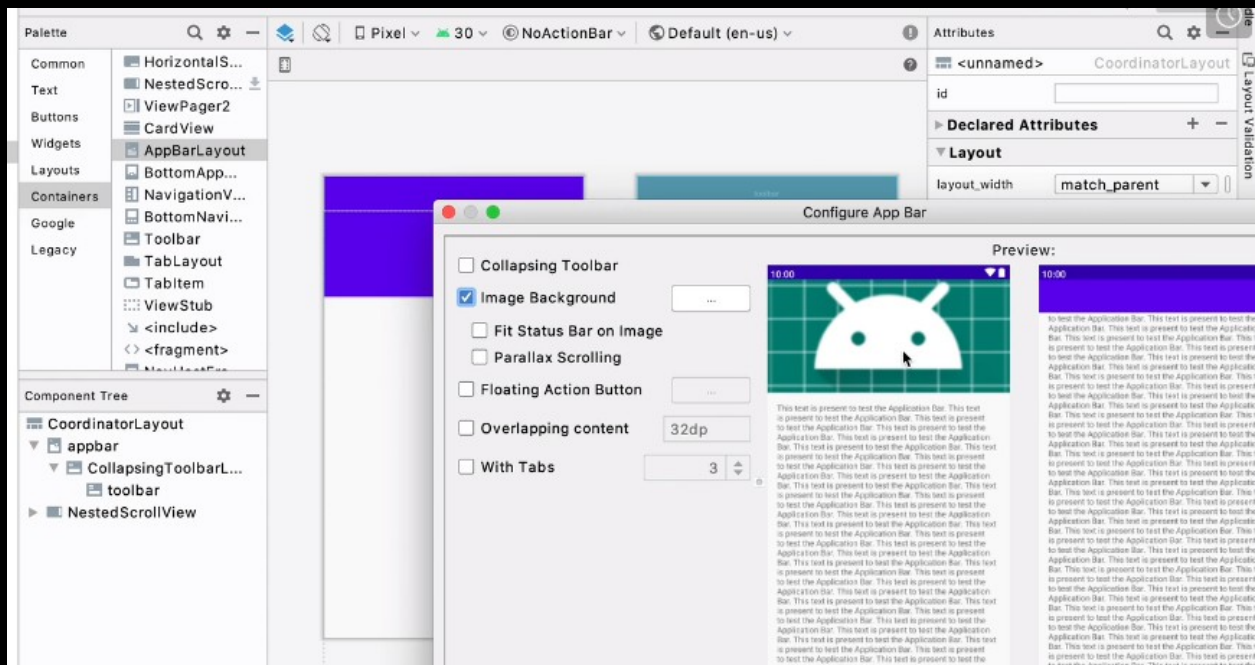
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context="com.example.myapplication.MainActivity">

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```

Los elementos colocados en la barra presentarán una **acción asociada** a través de eventos.

## Funcionamiento de la barra de app

Cuando desarrollamos una aplicación para dispositivos móviles con el sistema operativo Android, es importante tener en cuenta que aparecerá una **barra en la parte superior** de estas aplicaciones. Esta barra puede ser llamada de diversas formas, como barra de aplicaciones, barra de app o barra de acciones. En esta barra suelen mostrarse **accesos directos a diferentes funcionalidades** de la herramienta. Además, a menudo encontramos **tres puntos verticales** en el extremo derecho, conocidos como **menú ampliado**, donde se incorporan acciones adicionales que no caben en la barra de acciones principal.



Vamos a analizar cómo se inserta y diseña una interfaz que incluya este elemento. Uno de los atributos principales que vamos a examinar es 'showAsAction'.

Dentro de Android Studio, en la vista de diseño, podemos insertar esta barra seleccionando '**Container**' y luego '**appBarLayout**'. A continuación, podemos elegir entre diferentes opciones de diseño, como 'SmartKey', que proporciona un diseño más amplio, o 'WithTabs', que incluye pestañas dentro de la barra de acciones. Una vez seleccionada la opción deseada, simplemente confirmamos y la barra se insertará en la interfaz.

El código XML es crucial para definir los componentes de la interfaz. En este caso, el 'toolbar' se define con atributos como dimensiones de ancho y alto. Para modificar atributos específicos, como 'showAsAction', es necesario acceder al archivo 'menu\_main.xml' dentro de la carpeta 'res/menu' del proyecto.

El atributo '[showAsAction](#)' determina si los botones de la barra de herramientas se mostrarán como iconos directamente en la barra de acciones o en el menú ampliado.

Las opciones incluyen '[ifRoom](#)', que muestra el icono solo si hay suficiente espacio en la barra de acciones, y '[never](#)', que siempre coloca el elemento en el menú ampliado.

Es importante colocar los elementos **directamente en el menú ampliado por defecto** para que los desarrolladores puedan **priorizar** qué elementos deben aparecer en la barra de acciones. Esto evita que la barra de acciones se sature con elementos innecesarios.

Para continuar trabajando en el diseño de la interfaz, volvemos siempre al archivo 'activity\_main.xml', que es donde se desarrolla la mayor parte de la interfaz.

## Ejemplo de Navigation con dos pantallas

Utilizando el componente Navigation, vamos a implementar un sistema compuesto por, al menos, dos pantallas que deben cumplir los siguientes criterios:

- La primera ventana recibirá el nombre FirstFragment y será principal, desde donde se inicia la ejecución de la aplicación.
- La segunda ventana y siguientes recibirán los nombres SecondFragment, ThirdFragment...

Para la implementación de este caso, debemos crear un **nuevo componente Navigation** desde el menú **New Android Resource File**.

A continuación, se colocarán las pantallas incluyendo un nuevo fragmento, o desde el menú superior se selecciona el botón con el signo + de color verde y se añaden nuevas pantallas.

El código muestra la implementación de la descripción anterior:

Creación de un componente Navigation con dos pantallas y acciones

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/FirstFragment">

    <fragment
        android:id="@+id/FirstFragment"
        android:name="com.example.myapplication.FirstFragment"
        android:label="fragment_first"
        tools:layout="@layout/fragment_first">

        <action
            android:id="@+id/action_FirstFragment_to_SecondFragment"
            app:destination="@id/SecondFragment" />

    </fragment>

    <fragment
        android:id="@+id/SecondFragment"
        android:name="com.example.myapplication.SecondFragment"
        android:label="fragment_second"
        tools:layout="@layout/fragment_second">

        <action
            android:id="@+id/action_SecondFragment_to_FirstFragment"
            app:destination="@id/FirstFragment" />

    </fragment>

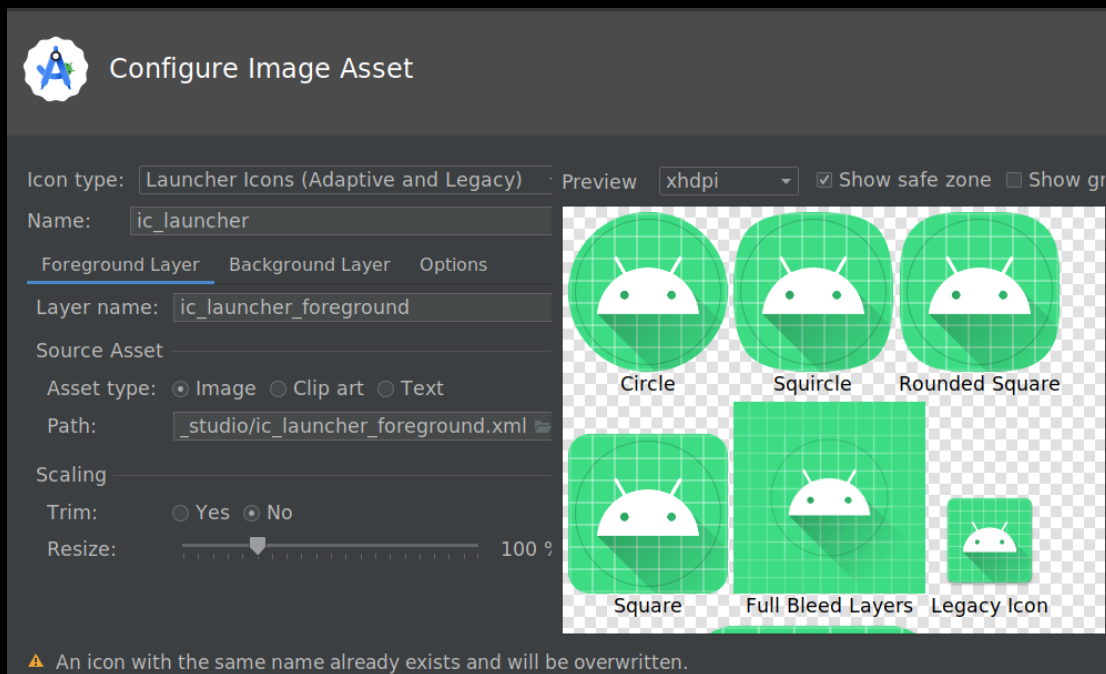
</navigation>
```

## Crear iconos y logotipos personalizados para la aplicación

El desarrollo de iconos personalizados para cada aplicación es un elemento muy importante para dotar al sitio de una marca propia de identidad. Android Studio incluye herramientas que permite utilizar imágenes y adaptarlas con distintas plantillas a los diferentes elementos visuales que pueden ser utilizados en el desarrollo de una interfaz.

En este caso, se va a desarrollar un conjunto de iconos para una aplicación que permite realizar llamadas telefónicas que van a tener una duración máxima de 10 minutos. Es decir, la funcionalidad interna de la aplicación finaliza la llamada al pasar el período de tiempo indicado en el menú de configuración.

### Configuración Imagen Asset



Para la creación de estos iconos personalizados, se debe acceder a la **ventana de configuración**, desde el res, pulsamos **New** y, a continuación, **Imagen Asset**.

En el menú de la herramienta, es posible modificar todos los atributos de diseño. Por ejemplo, imaginemos que los colores característicos de la aplicación son el naranja y el azul, por lo que accediendo a la paleta de colores, tomaremos el azul para el fondo del icono y el naranja para el símbolo central.

Ahora bien, ¿qué imagen es conveniente utilizar para el icono? Escogemos una imagen que sea fácil de asociar al propósito de la aplicación y que, además, presente líneas sencillas que no resulten difíciles de identificar en dispositivos de menor tamaño.

<https://developer.android.com>

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Desarrollo de interfaces para iOS



## Descarga e instalación del entorno Xcode

En este último capítulo, nos vamos a iniciar en el desarrollo para **iOS**, el **sistema operativo** de los **dispositivos** móviles de **Apple**. Se llevará a cabo una revisión de los aspectos clave relativos a la construcción del entorno necesario de desarrollo, se creará una primera aplicación, y, finalmente, analizaremos algunas pautas para el diseño de la interfaz de usuario para las aplicaciones iOS.

En primer lugar, desde el siguiente [enlace](#) se accede a la web de desarrollo de Apple en la que se encuentra toda la información oficial necesaria.

La **herramienta esencial** que se necesita para desarrollar en este sistema operativo es **Xcode**, el Entorno de Desarrollo Integrado (**IDE**) que nos proporciona Apple. Desde este IDE es posible **diseñar la interfaz** gráfica, **implementar** la aplicación, **probarla** de forma adecuada y, finalmente, **publicarla** en la App Store. Gracias a sus características, resulta un entorno de desarrollo excelente, puesto que desde una única herramienta será posible abordar todo el ciclo de vida de una aplicación.

Además de Xcode, también será necesario conocer el lenguaje de programación utilizado para el desarrollo de las aplicaciones en iOS, **Swift**.

Por lo tanto, en primer lugar, nos dispondremos a realizar la descarga de Xcode desde la App Store. Es imprescindible tener un equipo con macOS X para poder realizar el desarrollo de una aplicación para iOS.

## Xcode

### Primeros pasos

Tras realizar la instalación de Xcode en nuestro equipo, iniciamos la aplicación y, en la pantalla de inicio, nos aparecen varias opciones:

- abrir un proyecto anterior,
- crear un playground (permite al desarrollador **implementar** ciertos bloques de código **que no son aplicaciones** como tal),
- crear un nuevo proyecto en Xcode (es decir, crear una aplicación)
- y, finalmente, descargar un proyecto almacenado en un repositorio.

Para crear un nuevo proyecto desde cero, seleccionamos “**Create a new Xcode project**” .

A continuación, la herramienta permite configurar varios aspectos del desarrollo como el **sistema operativo** que se va a utilizar (**iOS, watchOS, tvOS, macOS**) o la **plantilla de diseño** para la aplicación. Como se puede deducir, Xcode permite el desarrollo de aplicaciones para todos los sistemas operativos de Apple, **para cada uno de los dispositivos** de la marca.

Tras la selección del sistema operativo (en nuestro caso iOS) para el desarrollo de aplicaciones móviles y una de las plantillas (plantilla en blanco con una sola pantalla), se pulsa el botón Next.

### Configuración de un nuevo proyecto

A continuación, aparece la ventana de configuración del proyecto en la que indicaremos el **nombre** del mismo. En la lista de valores **Team**, es posible seleccionar una **cuenta de desarrollador necesaria** para poder **publicar** en la App Store las aplicaciones implementadas.

Otro de los datos clave que se indican en la ventana de configuración es **Language**, donde se seleccionará **Swift**, el lenguaje más moderno y potente para el desarrollo en Apple. **Objective-C** es el primer lenguaje que se utilizó.

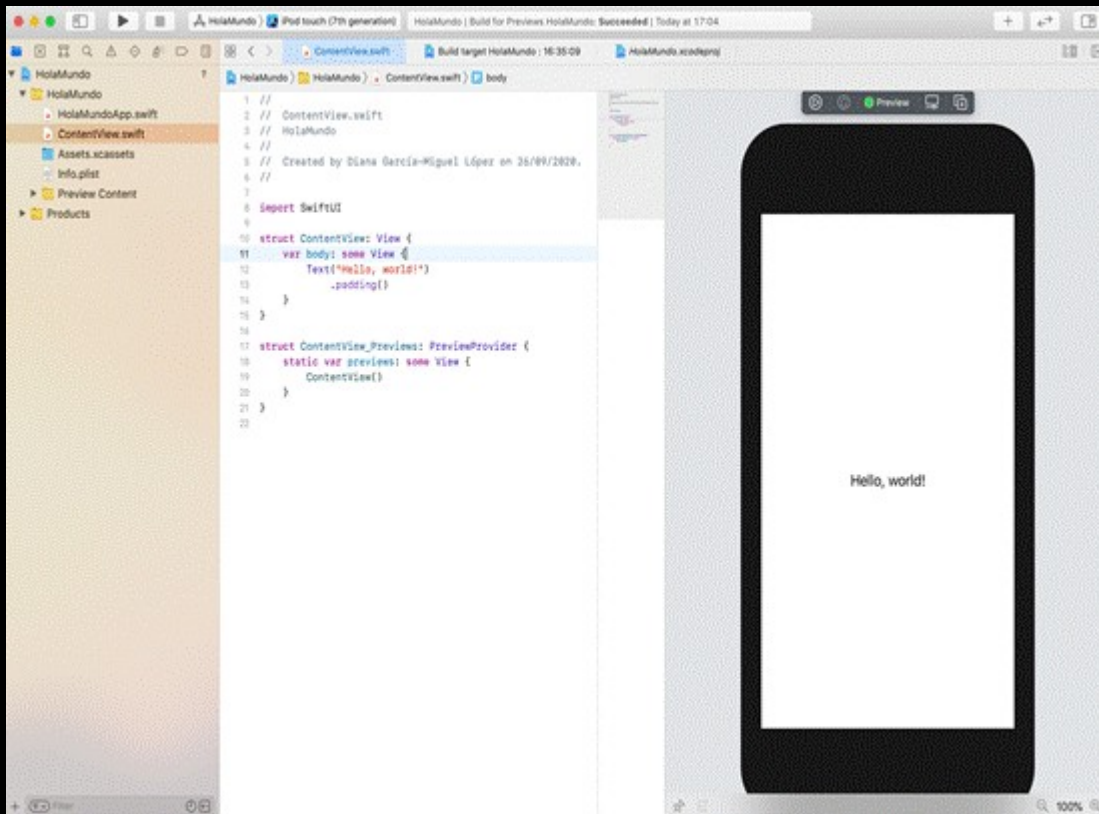
Finalmente, será necesario definir la **interfaz de usuario** (User Interface), que puede tomar los siguientes valores en función del tipo de aplicación desarrollada:

- **Storyboard**: tipo de **interfaz** utilizada **hasta 2019**.
- **SwiftUI**: **nueva librería** que incluye mejoras para el desarrollo de las **interfaces web**, y es la **más utilizada** en la actualidad para nuevos desarrollos.

Tras pulsar el botón Next, seleccionamos el **directorio** en el que va a estar ubicado el proyecto y, finalmente, se pulsa Create. Por defecto, se creará un nuevo “Hola Mundo”.

## Análisis del entorno

### Menú de configuración (zona superior de Xcode)



Este entorno de diseño permite la creación de aplicaciones de forma sencilla, analizamos cada uno de los apartados de esta herramienta.

En primer lugar, en la siguiente figura, se muestra la zona de la herramienta que permite **ejecutar y detener** la aplicación, así como el **nombre del proyecto** y el tipo de **simulador** escogido para probar la aplicación que se está desarrollando.

En este último menú, mediante una lista de valores, también es posible escoger un **dispositivo físico** para probar la aplicación.

En la **barra superior** de la herramienta, se indica el **estado actual** de la aplicación, por ejemplo, *Running HelloWorldCode on iPhone 11*, es decir, nos indica que se está ejecutando el proyecto de nombre HelloWorldCode en un simulador de tipo iPhone 11.

Finalmente, en la parte superior, podemos observar el siguiente menú:

Otras funcionalidades + librería



Este menú es uno de los más **importantes**, puesto que será el que nos permita seleccionar **nuevos componentes** para ser colocados sobre la interfaz de la aplicación. El **acceso a la paleta** de componentes se realiza a través del botón con el **símbolo “+”**. Es el **botón de librería**.

La ventana de **selección de elementos** queda distribuida en varios tipos:

- **Componentes** gráficos.
- **Bloques** de código.
- **Imágenes**.
- Paleta de **colores** configurados para el proyecto.

El segundo botón nos permite **comparar el código** desarrollado con **versiones previas** que el desarrollador hubiese subido a algún repositorio. Finalmente, el resto de opciones de este menú permiten **personalizar la vista de la interfaz**, al igual que ocurría con Android Studio a través del botón Split.

## Zona central de la aplicación

Como se puede observar en la imagen anterior de la interfaz, se muestra la interfaz de la aplicación Xcode que está estructurada en **tres grandes zonas de acción**.

Zona izquierda:

en esta zona, encontramos un completo catálogo de funcionalidades, como son:

- el acceso a los **repositorios**,
- un **buscador**,
- la zona en la que se muestran **advertencias**,
- análisis de **rendimiento**,
- **breakpoints** o puntos de ruptura,
- entre otras.

Una de las funciones más importantes de esta zona es [Navigator](#), accesible a través del icono de carpeta, donde se muestran los **ficheros y directorios** de los que queda compuesto cada proyecto. Si pulsamos sobre el archivo principal de nuestro proyecto (que aparece en primer lugar), se muestra la **configuración principal** del proyecto donde aparecen algunos de los datos principales del mismo.

#### Zona de desarrollo:

Es la **zona central** de la interfaz de la herramienta y nos muestra el **código** de implementación.

Cuando se selecciona un fichero de un proyecto en desarrollo, esta zona queda subdividida en **dos partes**: una parte para el **código** y otra llamada **Preview**. En esta última, al pulsar el botón **Resume**, se muestra una **previsualización** de la aplicación, tomando como referencia el **simulador** previamente seleccionado.

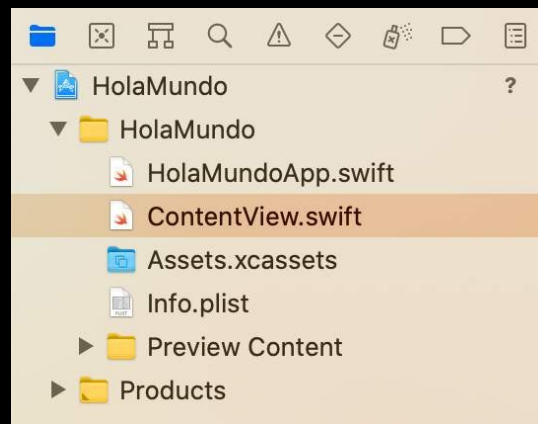
#### Zona derecha:

Como ocurre en otros entornos de desarrollo, esta zona solo se activa cuando **algún elemento** está **seleccionado**. En función del elemento del diseño escogido, se muestra un menú u otro, permitiendo **modificar las características** de los atributos. Por ejemplo, si se selecciona una etiqueta de texto, será posible modificar el String que se muestra, la fuente, o el tamaño.

## Creación de la primera aplicación

En primer lugar, para la **creación del código** de la aplicación, debemos acceder al fichero ContentView.swift.

Ficheros de desarrollo de un proyecto



Para la creación de una primera aplicación, el código utilizado es el siguiente:

1. Se añade la siguiente instrucción que permite [importar la librería SwiftUI](#).

Instrucción librería SwiftUI

```
import SwiftUI
```

2. La función ContentView define el **contenido** de la pantalla. En este caso, se incluye un componente de texto con el contenido “Hola mundo”.

Función ContentView. Define contenido de la pantalla

```
struct ContentView: View {  
    var body: some View  
    {  
        Text("Hola Mundo")  
    }  
}
```

3. Finalmente, es necesario introducir el siguiente fragmento de código para que se genere la previsualización que aparece a la derecha de la pantalla en el IDE.

Código lanzador de la previsualización que aparece a la derecha

```
struct ContentView_Previews: PreviewProvider {  
    static var previews: some View {  
        ContentView()  
    }  
}
```

Por último, para añadir alguno de los elementos y componentes del menú Librería, accesible a través del **botón ‘+’** que aparece en el menú superior en la derecha, basta con seleccionar uno y colocarlo en la posición deseada en la interfaz gráfica o en la zona de implementación.

## Ampliación de contenido Swift. Manual de uso

<https://www.apple.com/la/swift/>

libro de Swift:

<https://books.apple.com/us/book-series/swift-programming-series/id888896989>

Guía del desarrollador de Swift

<https://developer.apple.com/swift/>

Apple, desde su sitio web, incorpora un amplio catálogo tanto de aplicaciones como de información sobre el lenguaje de programación que se utiliza para el desarrollo de aplicaciones tanto para iOS, iWatch o macOS, entre otras. Como vemos, aquí está la página donde tenemos todo lo relativo a Switch. Si vamos bajando hacia la parte de abajo, tenemos herramientas que han sido desarrolladas utilizando este nuevo lenguaje de programación que, como nos cuentan, se trata de una potente herramienta de desarrollo. Además, nos indica que los docentes están incluyendo Switch en sus planes de estudios.

Podemos ver que esto está desarrollado íntegramente utilizando Switch y el entorno de desarrollo propio de Apple, que ya hemos visto, Xcode. La parte de aquí abajo, que es la que nos queríamos centrar, incorpora los tres elementos principales para comenzar a trabajar con Switch. En este tema, en este capítulo, hemos creado unos primeros pasos para acercarnos en qué consiste, qué entornos de desarrollo necesitamos, qué lenguajes de programación para desarrollar con iOS.

Evidentemente, la creación de este tipo de aplicaciones requiere de tiempo. Aun así, si nos convertimos en expertos desarrolladores de este tipo de aplicaciones, dentro del entorno profesional, eso va a ser un punto a nuestro favor. Para empezar, centrándonos en estos tres elementos que van a ser, en primer lugar, la obra de referencia de Switch Programming Language. Esta obra, este iBooks, permite descargarse gratis desde este enlace. Vamos a necesitar de un dispositivo iOS para poder leer, pero como digo, este libro, vamos a pinchar aquí y vamos a ver la descarga.

Se trata de un libro muy completo, está en inglés, pero es muy completo y nos habla de todas las estructuras de control, todo lo que necesitamos saber para desarrollar utilizando Switch. Como vemos, en este caso, se abriría la aplicación de Switch y si accedemos aquí, vemos una vista previa. En este caso, como lo tengo descargado en el equipo, pulsaríamos en leer y ya nos aparecería todo el libro. De forma gratuita, tenemos un manual completo de un lenguaje de desarrollo. Es muy aconsejable revisar esta aplicación.

Además, tenemos en el siguiente punto [Xcode](#). Como hemos visto, Xcode es el entorno de desarrollo. Para descargarlo, basta con pulsar a esta opción de aquí, descargar Xcode desde la App Store y accederíamos a ello. Finalmente, aquí tenemos el icono que nos habla de Switch. En este caso, si seleccionamos el enlace, nos describe las diferentes pautas. Es un resumen del libro que hablábamos en el primero de los enlaces.

Os aconsejamos revisar todo esto si de verdad queréis comenzar el desarrollo en interfaces multiplataforma, porque se trata de una aplicación de todo lo que se ha visto a lo largo de esta asignatura con las particularidades propias de cada lenguaje de programación y cada entorno de desarrollo, pero la base principal va a ser la misma, la creación de eventos, el tratamiento y captura de estos y la inserción de diferentes componentes empaquetados que nos permite la reutilización de los mismos para crear aplicaciones e interfaces adaptadas a cualquier tipo de entorno y de casuística.



## Swift. Primeros pasos

Swift es el lenguaje de programación de Apple para crear aplicaciones para sus dispositivos (iOS, watchOS...). En estos apartados, veremos algunos de los puntos claves de desarrollo de este lenguaje que, al igual que en el resto de lenguajes, presenta su propia sintaxis y reglas.

### Creación de variables

Una variable es un elemento utilizado para almacenar un valor que podrá ser modificado a lo largo de la ejecución, es decir, podrá tomar otros valores.

Pero se debe tener presente que si en primer lugar se **crea una variable** con una cadena de texto y luego se modifica su valor a un entero, el sistema nos devolverá un error, puesto que la variable será del tipo que se declaró la primera vez, y ya **no será posible cambiar el tipo de dato interno**. Por tanto, será necesario crear una nueva variable.

Para crear variables se utiliza la palabra var y, a continuación, se indica el nombre de la misma. Para la construcción del nombre, cada vez es más común observar ciertas convenciones sobre cómo deben ser definidos los nombres. Uno de los más utilizados es **CamelCast**, este indica que se tiene que diferenciar cada una de las palabras con una letra mayúscula, **empezando siempre con una letra minúscula**. Por ejemplo, un nombre válido sería primeraVariable. A continuación, se indica el valor de dicha variable.

#### Creación de variables

```
var primeraVariable="Hola Mundo"
```

Ahora bien, la creación de las **instancias** en Swift se realiza a través de **struct**, que equivalente a una **class de JavaScript**.

Por ejemplo, si queremos definir un **nuevo objeto** que contiene una cadena de texto, quedaría de la forma:

#### Creación de clases

```
struct nuevoDato {  
    var primeraVariable: String  
}
```

Para la posterior **instanciación de un objeto** de la clase nuevo dato, se utilizará:

```
var nuevoDato1 = nuevoDato(primerVariable:"Hola")
```

## Comentarios

El uso de comentarios en programación no es un hecho trivial, es clave para construir programas que resulten más fáciles de mantener en el futuro.

```
/* */
```

(Barra inclinada adelante, asterisco de comienzo, asterisco de fin y barra inclinada adelante): Estos permiten insertar entre los valores de apertura y cierre todas las **líneas** de comentarios que sean necesarias. Este tipo de delimitadores son útiles cuando existen varias líneas de comentarios o se quiere comentar algún fragmento de código.

```
/*  
Aquí  
hay  
varias líneas  
*/
```

```
//
```

Este tipo de delimitadores se utilizan, habitualmente, para comentar **una línea** completa.

```
// Esto es un comentario de una línea
```

## Botones

Los botones son elementos muy importantes en el desarrollo de interfaces, ya que permiten al usuario tener una interacción directa con la aplicación. Para añadir uno de estos elementos, basta con introducir el siguiente **fragmento de código** o seleccionarlo de la paleta de componentes.

Creación de un botón

```
Button(action: signIn) {  
    Text("Sign In")  
}
```

También es posible **desplegar la librería** a través del botón '+' y desde componentes gráficos, seleccionar el elemento, en este caso, un Button. Si pulsamos sobre él y lo **arrastramos** hasta la zona de desarrollo, se **añade el código**.

- En **Action** se indica qué va a hacer el botón cuando este sea pulsado,
- y en **Content**, el **texto** que aparecerá en el botón.

Insertión de botón en código

```
struct ContentView: View {  
    var body: some View {  
        Button(action:/*Action*/){  
            // Content  
        }  
    }  
}
```

## Pautas para la creación de una interfaz en iOS

De nada sirve que el funcionamiento interno de una aplicación sea extraordinario, si el diseño tanto en aspecto como interacción de la interfaz de la aplicación no es bueno.

El desarrollo de las interfaces para aplicaciones en iOS se caracteriza por presentar diseños limpios y **minimalistas** que incluyen todo lo necesario, pero **sin sobrecargar** la interfaz.

Las **características de diseño** que diferencia este sistema de otras plataformas son:

- **Claridad:**
  - implica que el **texto** contenido en la aplicación debe ser **legible** en todos los tamaños,
  - los **iconos** deben ser **precisos**, es decir, no utilizan imágenes recargadas.
  - Además, la **distribución** en el espacio de los componentes, la **paleta de color** seleccionada, las **fuentes y los gráficos**, entre otros, resaltan el **contenido importante** de la aplicación y aportan interactividad al usuario.
- **Adaptación al contenido:**

la **navegación** en la aplicación debe realizarse a través de un **movimiento fluido** que permite a los usuarios comprender e interactuar con el sistema con movimientos naturales. Por otro lado, el **contenido** debe ocupar **toda la pantalla** del dispositivo.
- **Profundidad:**

la aplicación se implementa a través de varias **capas visuales** cuya transición entre capas aporta **sensación de movimiento** sobre la jerarquía de la aplicación, navegando “en profundidad” por el diseño.

En cuanto a los **criterios de diseño** establecidos por Apple para el diseño de las aplicaciones y garantizar de esta forma el impacto de sus desarrollos en el mercado, son:

- **Integridad estética.**
- **Consistencia.**
- **Manipulación directa.**
- **Retroalimentación.**
- **Metáforas.**
- **Control de usuario.**

## Aspectos importantes de las interfaces en iOS

El nuevo sistema operativo **iOS 14** trae novedades muy importantes respecto al diseño de interfaces móviles. Desde septiembre de 2020 el diseño de la **pantalla de inicio**, los **mapas** y **mensajes** han sufrido un rediseño.

A partir de esta nueva versión la **pantalla de inicio puede personalizarse** con un mayor grado de detalle:

- se podrán **agrupar las aplicaciones** de distintas pantallas,
- o **paginar en App Library**.
- En la parte **superior izquierda** estará una agrupación de **sugerencias**,
- y en la **parte superior** derecha estarán las de **uso más reciente**.

Además, cualquier aplicación podrá ocupar más espacio en la pantalla mediante la creación de **Widgets del tamaño deseado** por el usuario.

Y por último otra de las novedades más importantes y que se deberá de tener en cuenta a la hora de diseñar interfaces para iOS 14, es que:

desde ahora se puede utilizar el **modo “Picture in picture”**:

es decir, la posibilidad de tener **abierta una pantalla**, por ejemplo, reproducir un vídeo, mientras que se está **utilizando otra aplicación**.

## Colores del sistema

iOS proporciona una gama de colores que se adaptan al diseño garantizando aspectos claves relativos a la **accesibilidad**, como son el **aumento del contraste** y la **disminución de la transparencia**. La elección de color de iOS permite que estos aseguren la satisfacción en el usuario, tanto cuando son mostrados de forma individual como combinándolos.

El **cuidado diseño en las interfaces** de las aplicaciones de Apple que garantizan la usabilidad de la misma son una de sus características de identidad. Por ello, desde el sitio oficial, indican los siguientes **pilares clave para la elección del color**:

- Usar el **color con prudencia**. Es aconsejable utilizarlo **solo** para resaltar **determinados elementos** sobre los que queremos llamar la atención del usuario.
- Utilizar **colores complementarios** en el diseño completo de la aplicación. Es decir, los colores escogidos deben funcionar bien de forma **independiente y combinados**.
- Seleccionar una **paleta de colores concreta y limitada**. Además, es aconsejable que esta selección vaya en consonancia con el diseño del logo de la aplicación.
- Escoger un **color característico** para toda la aplicación. Aunque se seleccione más de un color para el diseño de la aplicación, es conveniente que siempre exista **uno principal** que identifique a la aplicación. Por ejemplo, la app del BBVA basa su paleta de color en el azul, o la aplicación general Notes, escoge el color amarillo como central.
- Proporcionar **dos colores** para garantizar que la aplicación se vea bien en el **entorno oscuro y el claro**, que son los dos modos disponibles en iOS.

Los dispositivos iOS permiten **seleccionar dos tipos de interfaces**: una **oscura** y otra **clara**, por lo tanto, en función de cual se seleccione, será conveniente escoger unos colores u otros.

Por ejemplo, en la siguiente imagen, se muestran algunos colores en su versión clara y oscura:

Default		Accessible	
Light	Dark	Name	API
<div><div>R 0</div><div>G 64</div><div>B 221</div></div>	<div><div>R 64</div><div>G 156</div><div>B 255</div></div>	Blue	<a href="#">systemBlue</a>
<div><div>R 36</div><div>G 138</div><div>B 61</div></div>	<div><div>R 48</div><div>G 219</div><div>B 91</div></div>	Green	<a href="#">systemGreen</a>
<div><div>R 54</div><div>G 52</div><div>B 163</div></div>	<div><div>R 125</div><div>G 122</div><div>B 255</div></div>	Indigo	<a href="#">systemIndigo</a>
<div><div>R 201</div><div>G 52</div><div>B 0</div></div>	<div><div>R 255</div><div>G 179</div><div>B 64</div></div>	Orange	<a href="#">systemOrange</a>
<div><div>R 211</div><div>G 15</div><div>B 69</div></div>	<div><div>R 255</div><div>G 100</div><div>B 130</div></div>	Pink	<a href="#">systemPink</a>
<div><div>R 137</div><div>G 68</div><div>B 171</div></div>	<div><div>R 218</div><div>G 143</div><div>B 255</div></div>	Purple	<a href="#">systemPurple</a>

## Ejemplo de creación de un proyecto desde cero con Xcode

Se va a desarrollar una nueva aplicación básica utilizando la IDE Xcode y el lenguaje de programación Swift. Se muestra una cadena de texto con la frase "Soy tu primera aplicación". La previsualización será como se muestra en la siguiente imagen:

En primer lugar, se va a crear un proyecto nuevo desde File, New, Project.

Ahora se selecciona la opción deseada para crear una nueva aplicación. Recordamos que sea en **iOS**, puesto que estamos desarrollando una **aplicación móvil**.

A continuación, en la ventana de configuración, se seleccionan los diferentes parámetros en base al diseño de la nueva aplicación. Ya hemos creado el proyecto, así que para **comenzar a programar**, basta con acceder al fichero **ContentView.swift**.

En el fichero **Content.View.swift** se añade la siguiente instrucción para importar la librería SwiftUI.

```
import SwiftUI
```

La **función clave** de todo desarrollo es **ContentView**, puesto que será esta la que **defina el comportamiento y los elementos** mostrados en la pantalla.

Si se desean **implementar otras funciones**, se pueden realizar externamente y ser **invocadas desde esta función**:

```
struct ContentView
```

Tal y como se indica en el enunciado, se añade una nueva etiqueta de texto.

```
Text ("Soy tu primera aplicación").padding();
```



El código completo se muestra a continuación:

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Soy tu primera aplicación").padding()
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

## A tener en cuenta al diseñar una interfaz para una aplicación iOS

Hemos introducido brevemente el funcionamiento de Swift y el entorno de desarrollo Xcode. Existe mucha información en la red, pero se aconseja, sobre todo, acceder al siguiente enlace, a través de un sistema de MAC OS X. Se trata de un **manual completo de uso para Swift** desarrollado por Apple.

<https://books.apple.com/es/book-series/swift-programming-series/id888896989>

Se pide diseñar la interfaz para una nueva aplicación iOS que cumpla los criterios de diseño utilizando una herramienta de **prototipado** o diseño gráfico. En este caso, se solicita el diseño de una aplicación para visualizar una galería de imágenes.

La programación de la funcionalidad es clave, pero el diseño de la interfaz también juega un papel fundamental. En el caso de este tipo de dispositivos, se aconseja seguir los criterios establecidos por Apple para el diseño de las aplicaciones y garantizar, de esta forma, el impacto de sus aplicaciones en el mercado.

En esta interfaz, se muestra el resultado de una aplicación para iOS creada con Xcode y lenguaje de programación Swift. Contiene las imágenes publicadas por los usuarios de la aplicación que han asistido a un determinado evento.

Gracias al uso de una interfaz sencilla e intuitiva, el resultado es lo suficientemente accesible para garantizar su éxito en el mercado y cumplir los requisitos de prototipo de las aplicaciones iOS.

## Conclusión

Hemos realizado una breve introducción al desarrollo en iOS, centrándonos en el entorno de desarrollo y en los primeros pasos a utilizar con el lenguaje de programación Swift.

La herramienta utilizada para el desarrollo de este tipo de aplicaciones es **Xcode**, el Entorno de Desarrollo Integrado (IDE) que nos proporciona Apple. Desde este IDE, es posible diseñar la interfaz gráfica, implementar la aplicación, probarla de forma adecuada y, finalmente, publicarla en la App Store. Gracias a sus características, resulta un entorno de desarrollo excelente, puesto que desde una única herramienta será posible abordar todo el ciclo de vida de una aplicación.

También, hemos aprendido que el desarrollo de las interfaces de las aplicaciones en iOS se caracteriza por presentar **diseños limpios y minimalistas** que incluyen todo lo necesario, pero sin sobrecargar la interfaz. Las **características de diseño que diferencia** este sistema de otras plataformas son:

- **claridad,**
- **adaptación al contenido**
- **y profundidad.**

(CAP nemotécnica)

En cuanto a los **criterios de diseño** establecidos por Apple para el diseño de las aplicaciones y garantizar el impacto de sus desarrollos en el mercado son:

- integridad estética,
- consistencia,
- manipulación directa,
- retroalimentación,
- metáforas
- y control de usuario.

<https://www.apple.com/es/swift/>

<https://developer.apple.com/swift/>