

TEMA 2: Entornos de Desarrollo

Módulo

Entornos de Desarrollo

para los ciclos

Desarrollo de Aplicaciones Multiplataforma

Desarrollo de Aplicaciones Web



ED FP-GS; Tema2:Entornos de Desarrollo

© Gerardo Martín Esquivel, Noviembre de 2022

Algunos derechos reservados.

Este trabajo se distribuye bajo la Licencia "Reconocimiento-No comercial-
Compartir igual 3.0 Unported" de Creative Commons disponible en
<http://creativecommons.org/licenses/by-nc-sa/3.0/>

2.1 Introducción.....	4
2.2 Eclipse.....	6
2.2.1 Descarga de Eclipse.....	6
2.2.2 Instalación en Ubuntu.....	6
2.2.3 Instalación en Windows 10.....	8
2.2.4 Cambiar el idioma.....	8
2.2.5 Espacios de trabajo.....	9
2.2.6 Proyectos.....	9
Para crear un proyecto:.....	9
Para importar un proyecto consolidado:.....	10
2.2.7 Conjuntos de trabajo.....	10
Para crear un conjunto de trabajo:.....	10
Para mover proyectos entre conjuntos de trabajo:.....	10
2.2.8 Ficheros.....	11
Para crear un fichero en blanco:.....	12
Para crear un fichero a partir de una plantilla:.....	12
Para importar un fichero a nuestro proyecto.....	12
Para eliminar un fichero o un proyecto.....	12
Cambiar el nombre de un fichero o un proyecto.....	12
2.2.9 Perspectivas.....	13
2.2.10 Configuraciones de ejecución.....	13
2.2.11 Plugins.....	14
2.2.12 WindowBuilder.....	14
Para comenzar a usar WindowBuilder.....	15
Propiedades.....	15
Un pequeño ejemplo.....	16
2.2.13 ER Master.....	17
Para crear un modelo:.....	17
Notación UML para Entidad/Relación.....	17
Crear una tabla.....	17
Crear relaciones.....	18
Exportar esquema a otros formatos.....	19
Generar la base de datos.....	19
Importar esquema desde una BD.....	20
2.2.14 Otras ayudas interesantes.....	20
2.3 Netbeans.....	21
2.3.1 Descarga.....	21
2.3.2 Instalación.....	22
2.3.3 Uso básico de Netbeans.....	23
Plugins para Java.....	23
Plugins para C.....	23
Abrir un proyecto.....	23
Crear un proyecto nuevo Java.....	24

Crear un proyecto nuevo C.....	24
Crear una clase Java.....	24
Crear un fichero fuente C.....	24
Ejecutar un proyecto.....	24
Borrar un proyecto.....	24
Usar parámetros de línea de comandos.....	24
Importar un proyecto creado con Eclipse.....	24
Cambiar el nombre de un fichero o proyecto.....	25
Sugerencias Netbeans.....	25
2.3.4 GUI Builder.....	25
2.4 Android Studio.....	27
2.4.1 Descarga en Windows.....	27
2.4.2 Descarga en Ubuntu.....	27
2.4.3 Instalación (Windows y Ubuntu).....	28
2.4.4 Versiones Android Vs niveles API.....	29
2.4.5 Crear un proyecto.....	30
2.4.6 Estructura de un proyecto: ficheros y carpetas que lo componen.....	31
AndroidManifest.xml.....	31
Carpeta java.....	32
Carpeta res.....	32
2.4.7 Ejecutar un proyecto en un terminal virtual.....	35
2.4.8 Ejecutar un proyecto en un terminal real.....	36
2.4.9 Ejemplo: Crear una app Calculadora.....	37
Ejemplo: Código Java para el método del botón Suma.....	38
Varios idiomas.....	39
Varios Layouts.....	40

2.1 Introducción

Un **IDE** (Integrated Development Environment - Entorno Integrado de Desarrollo) es una aplicación que incluye las herramientas que necesita un programador para hacer su tarea. Un **IDE** puede estar enfocado a un único lenguaje de programación o puede dar soporte para múltiples lenguajes.

Los elementos que incluye el **IDE** son:

- **Editor de textos:** para escribir el **código fuente** del programa. Contiene todas las utilidades que cualquier editor de textos del mercado (**bloc de notas**, **gedit**, ...) pero además hace una gran ayuda coloreando de forma distinta los variados elementos que existen en un lenguaje de programación: palabras reservadas, instrucciones, comentarios. Ayuda también con recordatorios de símbolos de puntuación que se suelen olvidar, con autocompletados, etc.
- **Compilador:** para traducir el **código fuente** a **código objeto**. En el tema anterior vimos como hacerlo desde un terminal de **Windows** o de **Linux**. Dentro del **IDE** tendremos una herramienta que nos permite hacer el compilado. Si encuentra errores en el **código fuente** avisará de dónde se encuentran y ofrece ayudas muy detalladas para resolverlos.
- **Depurador (debugger):** para depurar y limpiar los errores. Nos permite analizar lo que ocurre paso a paso en la ejecución de nuestro programa y analizar los valores de las variables en cada momento.
- **Diseño de interfaz gráfica:** para dibujar con facilidad las pantallas que incluye nuestro programa. En lugar de escribir todo el código que dibuja los elementos de una pantalla (botones, etc) podremos, con esta herramienta, arrastrarlos, dimensionarlos y decorarlos hasta lograr el aspecto deseado. Posteriormente y de forma automática, se genera el código necesario para esa pantalla.
- **Control de versiones:** para mantener versiones distintas de un programa. En ocasiones, los programadores tenemos que modificar los programas hacia nuevas versiones sin desechar totalmente las antiguas (por ejemplo, porque es la que siguen usando los clientes mientras terminamos la nueva). Cuando nos encontremos en esta situación valoraremos mucho esta herramienta.

Existen multitud de **IDE**. La elección de uno de ellos dependerá del proyecto a realizar, del lenguaje que usamos, del tipo de licencia (libre o propietaria), etc. Algunos de los más extendidos son:

- **Eclipse:** Orientado al desarrollo de programas. Es gratuito y puede descargarse de la web oficial <http://download.eclipse.org>. Una vez instalada la aplicación básica podemos añadir **plugins** que nos permiten trabajar con distintos lenguajes de programación, lenguajes de marcas o modelos de datos. Por ejemplo, si queremos usar **Eclipse** para programar con el lenguaje **Java** tendremos que instalar el **JDK** (que ya es conocido por todos vosotros) e indicarle a **Eclipse** donde se encuentra.
- **SQL Developer:** Orientada a Bases de Datos. Es gratuito y puede descargarse de la web de Oracle <http://www.oracle.com/us/downloads/index.html> después de registrarse y aceptar las condiciones.

- **NetBeans**: Orientado al desarrollo de programas. Dispone de módulos para extender sus funciones. Es libre y gratuito sin limitaciones de uso aunque es un producto de **Sun Microsystems/Oracle Corporation**. Descargable en <https://netbeans.apache.org/download>.
- **Microsoft Visual Studio**: Orientado al desarrollo para **Windows**. Soporta múltiples lenguajes de programación como **C++**, **C#**, **Visual Basic**, **.NET**, etc). Es propietario de **Microsoft** que proporciona versiones de prueba de 90 días.
- **Android Studio**: Desde <https://developer.android.com/studio/index.html> lo puedes descargar gratuitamente. Es un entorno creado por **Google** orientado a la programación para **Android**. No obstante, es posible programar para **Android** usando otros **IDE**, como **Eclipse**.

2.2 Eclipse

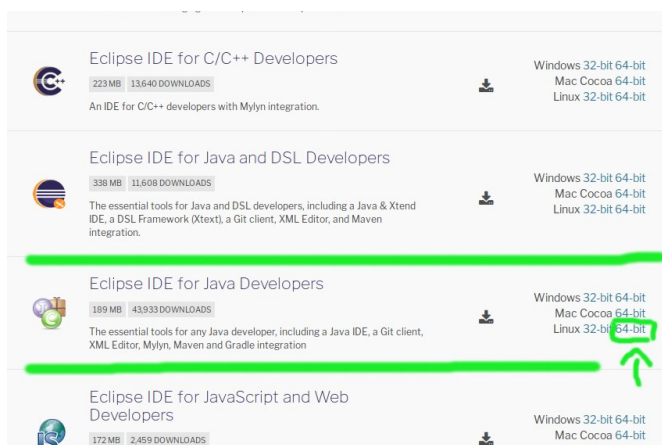
Eclipse es uno de los **IDE** más utilizados debido, probablemente, a que es software gratuito y a su flexibilidad para trabajar con distintos lenguajes de programación a través de **plugins**.

Las sucesivas versiones de **Eclipse** tienen nombres relacionados con la astronomía siguiendo orden alfabético en su primera letra, una versión por año que aparecía el 4º miércoles de junio de cada año. Así, las versiones han sido **Helios**, **Indigo**, **Juno**, **Kepler**, **Luna**, **Mars**, **Neon**, **Oxygen**, y, en Junio de 2018, **Photon**. Desde 2018 hay una revisión cada tres meses que se numera con año y mes: **2018-09**, **2018-12**, **2019-03**, etc.

2.2.1 Descarga de Eclipse

Puedes descargarlo desde <https://www.eclipse.org/downloads/packages> donde encontrarás el instalador de la última versión, que, en el momento de redactar este tema, es **2022-09**. Fíjate en la imagen y asegúrate de seleccionar el modelo básico para desarrollo de **Java** (no confundir con desarrollo **Java EE**) y válido para tu sistema operativo, teniendo en cuenta si dispones de la versión de 32 o la de 64 bits.

En la imagen está señalado el enlace de descarga de la versión para **Ubuntu**, pero como puedes ver también existen enlaces de las versiones de **MacOS** y **Windows**.



2.2.2 Instalación en Ubuntu

El fichero descargado según estas instrucciones tiene la extensión **.tar.gz** (comprimido y empaquetado). Sólo será necesario desempaquetarlo en el lugar adecuado y se generará una carpeta llamada **eclipse** que incluye entre otras cosas, un ejecutable.

El comando para desempaquetar desde la línea de comandos:

```
cd /opt
sudo tar -xzf ruta/nombreFichero.tar.gz
```

Nota: Fíjate que hemos instalado la aplicación en la carpeta **/opt** que es la adecuada para aplicaciones de terceros. También podríamos haber usado la carpeta **/usr**, aunque está pensada para aplicaciones que siguen de forma más estricta la filosofía de subcarpetas de **Linux**. En cualquier caso es buena idea tener bien organizada la información y las aplicaciones porque acabarás usando muchas.

Tras las acciones anteriores tenemos una nueva carpeta en **/opt/eclipse** y podremos ejecutar la aplicación con el comando:

```
/opt/eclipse/eclipse
```

Nota: Recuerda que para poder ejecutar **eclipse** será necesario tener instalado el **JRE** de **Java**.

Si quieres ejecutarlo desde cualquier carpeta sin escribir la ruta, tienes que añadir la ruta a la variable de entorno **PATH**, en el fichero **/etc/environment**.

La variable de entorno **PATH**

La variable de entorno **PATH** contiene una cadena con todas las rutas donde hay que buscar los ficheros ejecutables. Esta variable existe tanto en los sistemas **Linux** (donde las rutas se separan con el símbolo dos puntos :) como en los sistemas **Windows** (donde las rutas se separan con el símbolo punto y coma ;).

Cada vez que, desde el terminal, pedimos la ejecución de un programa sin indicar la ruta, el sistema busca el programa en todas las rutas que estén especificadas en la variable de entorno **PATH**.

Si el contenido de **PATH** finaliza con el símbolo separador (dos puntos en **Linux**, punto y coma en **Windows**), hace referencia a la carpeta actual, es decir, buscará el programa también en la ruta desde la que pedimos la ejecución. Si el contenido de **PATH** incluye el símbolo separador dos veces seguidas también hace referencia a la carpeta actual.

Por ejemplo, si queremos ejecutar la aplicación **eclipse** desde la propia carpeta donde se encuentra y la variable **PATH** incluye la carpeta actual, bastará con escribir:

```
eclipse
```

Pero si la variable **PATH** no incluye a la carpeta actual, habrá que indicar la ruta:

```
./eclipse
```

Nota: Si quieres que te aparezca en el listado de programas deberás hacerlo manualmente. Esa operación depende del sistema operativo que estés usando.

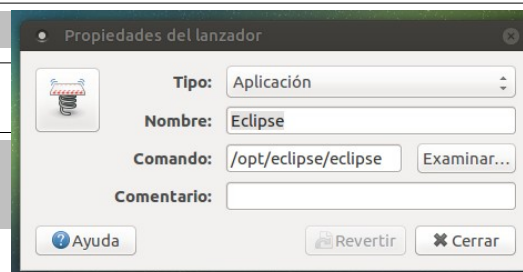
En el caso de **Ubuntu** con **Unity** bastará con pulsar (botón derecho) sobre el icono de **Eclipse** del lanzador y seleccionar “**Mantener en el lanzador**” o “**Añadir a favoritos**”. Si no permite esta acción es porque quizá te falte crear el acceso directo para **Eclipse** en el lanzador. Esto último se consigue creando un fichero de texto en **/usr/share/applications/eclipse.desktop** (mejor en **~/local/share/applications**) con el siguiente contenido:

```
[Desktop Entry]
Name=Eclipse
Comment=Eclipse
Exec=/opt/eclipse/eclipse
Icon=/opt/eclipse/icon.xpm
Terminal=false
Type=Application
```

En el caso de Ubuntu **Mate** debes ir a:

**SISTEMA/PREFERENCIAS/VISUALIZACIÓN Y
COMPORTAMIENTO/MENÚ PRINCIPAL**

Y en el apartado de programación crear un **elemento nuevo** con el comando necesario.



2.2.3 Instalación en Windows 10

El fichero descargado según las instrucciones tiene extensión **.zip** (comprimido) y deberíamos extraerlo en **C:/Archivos de programa**. En esta ubicación se creará una carpeta llamada **eclipse** que incluye un ejecutable **eclipse.exe**. Con doble-clic sobre este último fichero iniciamos **Eclipse** y ya podemos anclar el icono en la barra de tareas para tenerlo localizado en futuras ejecuciones.

2.2.4 Cambiar el idioma

La instalación inicial estará en inglés que es válida. Pero seguramente habrá gente interesada en trabajar en castellano. Podemos descargar los paquetes necesarios desde el propio **Eclipse**:

Help/Install new software...

Y aparece un cuadro de diálogo con un combo en la zona superior. Pinchamos en la flechita para desplegar la lista por si tuviésemos la suerte de tener un repositorio con los idiomas. Si es así lo señalamos. Si no es así, pulsamos el botón añadir para indicar uno:

- en nombre escribimos: **Eclipse idiomas**
- En ubicación: <https://download.eclipse.org/technology/babel/update-site/R0.18.0/2020-06/>

Nota: La ubicación de los repositorios puede cambiar en función de la versión de **Eclipse** y/o la versión de los paquetes. Podemos encontrar una lista actualizada de repositorios con idiomas de los distintas versiones de eclipse en <http://www.eclipse.org/babel/downloads.php>

Las versiones antiguas suelen quedar en una ubicación diferente que puedes obtener cambiando la palabra "**download**" por "**archive**", esto es, cuando la versión 0.8.1 de **Helios** era la más actual los idiomas se descargaban de:

<http://download.eclipse.org/technology/babel/update-site/R0.8.1/helios>

Desde que la versión 0.8.1 de **Helios** fue sustituida por otra más actualizada, aún podremos encontrar los idiomas de la versión antigua en la ubicación:

<http://archive.eclipse.org/technology/babel/update-site/R0.8.1/helios>

Tanto si ya teníamos el repositorio de idiomas, como si lo hemos tenido que añadir, cuando todo va bien, en el área grande del cuadro de diálogo aparecerá el rótulo "**pending...**" que indica que está buscando los paquetes (hay que tener un poco de paciencia). Si en lugar de ese rótulo aparece "**No repository found**" tendremos que revisar la dirección introducida o probar nuevamente después de un tiempo prudencial por si se debe a problemas de red.

En cualquier caso el rótulo "**pending...**" indica que el proceso va bien (aunque tarde un poco) y al final aparecerán todos los paquetes de idiomas para que elijamos (**Babel Language Packs in Spanish**) y ordenemos la descarga, al final de la cual y siguiendo las instrucciones todos los menús estarán traducidos. Bueno, no todos.

Nota: Se ha observado que después de descargar los paquetes de idioma español, la versión para **Windows** de **Eclipse** deja de detectar las entradas por teclado, de modo que queda inutilizado. Si te ocurre esto debes desinstalar el paquete de idioma español **rt.rap**. Búscalo en:

AYUDA/ACERCA DE ECLIPSE/DETALLES DE LA INSTALACIÓN/INSTALLED SOFTWARE

Si tienes problemas con la conexión a los repositorios de **Eclipse**, puedes descargar el archivo de idiomas desde <http://www.eclipse.org/babel/downloads.php> (**Zippped p2 repository for 2021-12**), descomprimirlo en una carpeta temporal y desde **Install new software**, añadir como repositorio la ruta absoluta a **.../2021-12/content.jar** que se ha generado.

2.2.5 Espacios de trabajo

Eclipse necesita un espacio de trabajo o **workspace** que será la carpeta en la que guarde todos los ficheros que generemos con él. Por defecto nos ofrece una carpeta con ese mismo nombre y, salvo que tengas otras preferencias, recomendamos aceptarla (será la primera pregunta cada vez que iniciamos el programa, aunque lo podemos evitar marcando la casilla de "no volver a preguntar").

Nota: el alumnado que use **Eclipse** para otros módulos, como el de **Programación**, el de **Bases de Datos** o el de **Lenguajes de Marcas**, puede estar interesado en crear varios espacios de trabajo. En ese caso podrían crear carpetas hijas de **workspace**, por ejemplo: **workspace/Programacion**, **workspace/BasesDatos** y **workspace/Marcas**. Cada vez que cambien de módulo tendrían que cambiar de espacio de trabajo con

ARCHIVO/CAMBIAR ESPACIO DE TRABAJO...

La verdad es que el cambio de espacio de trabajo es un poco molesto porque la aplicación se cierra y se vuelve a abrir, pero por otro lado podremos ordenar mucho mejor los trabajos de uno y otro módulo. También podremos indicar que nos pregunte al iniciar **Eclipse** en cual de los espacios de trabajo queremos comenzar. Para configurar el comportamiento de **Eclipse** respecto a los espacios de trabajo debemos ir a:

VENTANA/PREFERENCIAS/GENERAL/INICIO Y APAGADO/ESPACIOS DE TRABAJO

En esa pantalla podremos establecer si queremos que pregunte por el espacio de trabajo cada vez que iniciamos **Eclipse** y también añadir o eliminar espacios de trabajo. (Desde aquí no creamos ni eliminamos carpetas, sólo indicamos si esas carpetas se consideran como espacios de trabajo).

Muy importante: No manipules el espacio de trabajo directamente desde el sistema operativo, siempre desde el propio Eclipse.

2.2.6 Proyectos

Un **proyecto Eclipse** es el conjunto de todos los ficheros que operan juntos, por ejemplo todos los ficheros con código fuente que componen un programa **Java** o todos los ficheros **HTML** de una web. Crea un **proyecto Eclipse** distinto para cada trabajo, no hagas la chapuza de mezclar trabajos distintos en el mismo proyecto porque los problemas que tendrás serán superiores al ahorro de tiempo que puedas creer que tienes.

Podemos ver todos los proyectos del espacio de trabajo que tengamos activo en el **explorador de paquetes**. Es una ventana de **Eclipse** que suele encontrarse en la izquierda de la pantalla, en el caso de que no veas esa ventana:

VENTANA/MOSTRAR VISTA/EXPLORADOR DE PAQUETES

PARA CREAR UN PROYECTO:

ARCHIVO/NUEVO/PROYECTO . . .

Se abre un cuadro de diálogo con multitud de tipos de proyectos. Recuerda que **Eclipse** se puede usar con muchas finalidades. Debes elegir el tipo de proyecto más adecuado a tu tarea, por ejemplo: **Proyecto Java**.

Nota: Cada vez que generamos un proyecto nuevo, se creará una nueva carpeta dentro del espacio de trabajo y en ella se guardarán todos los ficheros que lo componen: el código fuente en la subcarpeta **src** y el código objeto en la subcarpeta **bin**. **Volvemos a insistir en que el espacio de trabajo no se debe manipular desde fuera de Eclipse.**

PARA IMPORTAR UN PROYECTO CONSOLIDADO:

Si tienes un proyecto ya creado con **Eclipse** (por ejemplo, que lo traes desde otro equipo) puedes añadirlo a tu ventana de proyectos. Ojo!! hablamos de proyectos creados con **Eclipse** que tienen sus propios datos de configuración, no ficheros sueltos creados con otras herramientas, para esos otros casos lee los siguientes apartados sobre importación de ficheros.

Lo primero que tienes que hacer es copiar la carpeta completa del proyecto en tu espacio de trabajo (esta tarea la hacemos desde el sistema operativo) y, a continuación:

ARCHIVO/IMPORTAR/GENERAL/PROYECTOS EXISTENTES EN EL ESPACIO DE TRABAJO/SIGUIENTE

a continuación marca “**seleccione el directorio raíz**”, clic en **Examinar**

seleccionamos la carpeta correspondiente al proyecto que queremos añadir y clic en **Finalizar**

Nota: Al importar el proyecto **NO SE HACE** copia, sino que trabaja con la carpeta original en su ubicación original. Si no copiamos la carpeta previamente en el espacio de trabajo, tendremos el espacio de trabajo diseminado por todo el disco duro. Llama la atención que cuando importamos un fichero el comportamiento es justo el contrario (véase “**importar un fichero**”).

2.2.7 Conjuntos de trabajo

Los proyectos se pueden agrupar en **conjuntos de trabajo**. Por ejemplo, imaginemos que en el módulo de programación te encargan una práctica con 10 ejercicios de **Java**. Cada ejercicio tendrá su propio proyecto, pero además puedes agrupar los 10 proyectos en un **conjunto de trabajo** llamado **Practica1**. Si, posteriormente, te encargan una segunda práctica, podrás crear otro conjunto de trabajo.

Para que los **conjuntos de trabajo** sean visibles en el **explorador de paquetes**:

- Abrimos el menú del **explorador de paquetes**

ELEMENTOS DE NIVEL SUPERIOR/CONJUNTOS DE TRABAJO

PARA CREAR UN CONJUNTO DE TRABAJO:

Desde el menú del Explorador de paquetes:

CONFIGURAR CONJUNTOS DE TRABAJO/NUEVO...

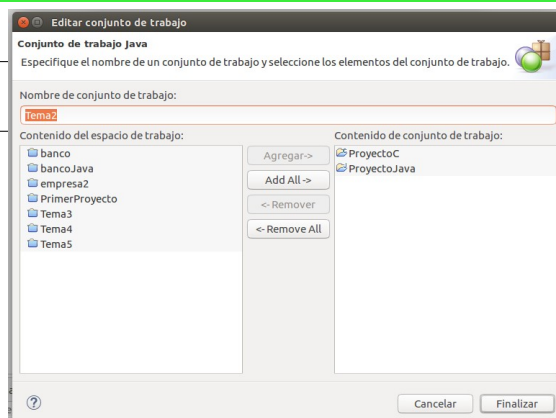
Nota: cuando se crean conjuntos de trabajo sólo son visibles desde **Eclipse**. En la estructura de directorios todos los proyectos aparecen juntos en el **workspace**.

PARA MOVER PROYECTOS ENTRE CONJUNTOS DE TRABAJO:

Abrimos el menú del **explorador de paquetes**:

CONFIGURAR CONJUNTOS DE TRABAJO.../(ELEGIMOS CONJUNTO)/EDITAR

Y pasamos los proyectos hacia dentro o hacia fuera del conjunto.



A modo de resumen:

Un **espacio de trabajo** hace referencia a una carpeta del sistema donde se almacenan todos los ficheros generados y/o manipulados desde **Eclipse**. Será suficiente un espacio de trabajo y solamente tienes que usar varios si utilizas **Eclipse** en actividades muy variadas.

En cualquier caso, en una sesión de trabajo se usa exclusivamente un espacio de trabajo.

Un **proyecto** es el conjunto de todos los ficheros implicados en determinado programa o ejercicio. Nunca se deberían incluir ficheros de dos programas distintos en un mismo proyecto. El proyecto estará asociado a una subcarpeta del espacio de trabajo y tendrá una estructura interna que dependerá del tipo de proyecto. Por ejemplo, un proyecto **Java** tendrá una subcarpeta **src** (source) con los ficheros que contienen el código fuente (**.java**) y una subcarpeta **bin** (binaries) con los ficheros que contienen el código compilado (**.class**).

Un **conjunto de trabajo** es una agrupación de proyectos que sólo existe a los ojos de **Eclipse**. La estructura de carpetas no se altera al agrupar los proyectos en un conjunto o en otro.

	Eclipse	Sistema de archivos
Proyecto Java	Carpeta azul con subcarpeta src para los fuentes (ficheros .java)	Carpeta en el workspace con subcarpetas src (fuentes .java) y bin (binarios .class)
Conjunto de trabajo Java	Carpeta que agrupa proyectos. Solo son visibles si usamos la perspectiva " Java "	No se refleja de ningún modo
Espacio de trabajo (workspace)	Conjunto de proyectos con los que trabajamos en cada sesión de Eclipse. Puede haber más de un Espacio de trabajo, pero en cada sesión trabajamos solamente con uno.	Carpeta donde se guarda todo lo que genera Eclipse
Perspectiva	Distribución de ventanas (configurable) que usamos para cada tarea, por ejemplo: Programación con Java, Depuración de código, Programación web....	

2.2.8 Ficheros

Antes de trabajar con un proyecto debemos asegurarnos de que lo tenemos seleccionado (en el explorador de paquetes) porque **Eclipse** permite tener varios proyectos abiertos a la vez y eso puede ser una fuente de problemas. Podemos saber si cada proyecto está abierto o no por el icono que lo acompaña. Lo mejor es cerrar todos los proyectos con los que no estemos trabajando, pero para los "atrevidos" bastará simplemente con asegurarse de que el proyecto con el que queremos trabajar es el seleccionado, aunque haya otros abiertos.

Podemos crear un fichero en blanco sin más o podemos crear un fichero indicando el uso que le vamos a dar para que **Eclipse** nos permita comenzar a partir de las plantillas.

PARA CREAR UN FICHERO EN BLANCO:

(con el proyecto abierto y seleccionado)

ARCHIVO/NUEVO/ARCHIVO

escribimos el nombre (la carpeta ya estará escrita si lo hicimos bien) y clic en **Finalizar**

PARA CREAR UN FICHERO A PARTIR DE UNA PLANTILLA:

(con el proyecto abierto y seleccionado)

ARCHIVO/NUEVO/OTRAS

y seleccionamos el tipo de fichero según nuestra necesidad. Según los **plugins** que tengamos instalados nos ofrecerá más o menos opciones. Podremos elegir, por ejemplo, una **clase** o un **paquete** para un proyecto **Java**, o bien, un fichero **XML**, **HTML** o **CSS** si estamos trabajando con lenguajes de marcas, etc. En realidad todos ellos los podemos crear a partir de un fichero en blanco. Si elegimos un fichero concreto es para aprovechar las ayudas que **Eclipse** proporciona presentando una plantilla correspondiente al tipo de fichero y que nos evita escribir una y otra vez las mismas cosas.

clic en **Siguiente**

escribimos el nombre respetando la extensión que nos ofrece (la carpeta ya estará escrita si lo hicimos bien), seleccionamos las ayudas que necesitemos y clic en **Finalizar**

PARA IMPORTAR UN FICHERO A NUESTRO PROYECTO

Si queremos incluir en un proyecto un fichero que ha sido creado fuera del proyecto (con una herramienta distinta de **Eclipse** o en otro proyecto distinto):

ARCHIVO/IMPORTAR/GENERAL/SISTEMA DE ARCHIVOS/SIGUIENTE/

clic en **Examinar** (*del directorio*, cuidado que hay dos botones **examinar**)

Seleccionamos la carpeta, seleccionamos el(los) fichero(s), clic en **Finalizar**.

Nota: Al importar un fichero, **Eclipse** se comporta de forma totalmente distinta a cuando importamos un proyecto. Ahora será necesario que el fichero se encuentre fuera del espacio de trabajo y al importar **SI SE HACE** una copia del fichero dentro del espacio de trabajo, de modo que el fichero original no se verá afectado. (véase "**importar un proyecto**").

PARA ELIMINAR UN FICHERO O UN PROYECTO

Atención: Eliminar un fichero o un proyecto desde **Eclipse**, significa eliminar también el fichero o carpeta respectivamente, **sin posibilidad de recuperarlo porque no va a la papelera**.

click derecho sobre el fichero o proyecto en la ventana de proyectos / **Suprimir**

Cuando borramos un proyecto será necesario marcar la casilla "**borrar los contenidos del disco**". Si no lo hacemos así estaremos borrando la referencia al proyecto dentro de eclipse, pero no la carpeta que contiene el proyecto.

CAMBIAR EL NOMBRE DE UN FICHERO O UN PROYECTO

Para cambiar el nombre de un fichero o un proyecto, seleccionamos el fichero o proyecto y:

ARCHIVO/REDENOMINAR (TAMBIÉN PUEDES PULSAR **F2**)

2.2.9 Perspectivas

Una **perspectiva** es una distribución de la pantalla de **Eclipse** en ventanas, cada una de las cuales ofrece una funcionalidad.

Eclipse dispone de varias perspectivas que adaptan el entorno a la tarea que se está realizando. Así para programar con **Java** existe la perspectiva **Java**, para trabajar con lenguajes de marcas tenemos la perspectiva **Web**, y para depurar un programa tenemos la perspectiva **depurar**.

Si en algún momento necesitamos cambiar la perspectiva podemos dirigirnos a:

VENTANA/PERSPECTIVA/ABRIR PERSPECTIVA

Nota: También puedes cambiar de perspectiva con los iconos de la esquina superior derecha.

La perspectiva **Java** distribuye las ventanas de la siguiente manera:

- **Explorador de paquetes** (a la izquierda): te muestra todo lo que tienes alojado en el espacio de trabajo que estás usando.
- **Zona de edición** (editor de textos, zona central): es la parte donde escribes el código.
- **Esquema de la clase** (a la derecha).
- **Consola Java** (abajo): para mensajes de salida, peticiones de entrada, errores, resultado de ejecución. Simula un terminal durante la ejecución del programa.

Nota: Hay muchas más ventanas. Si necesitas alguna búscala en **Ventana/Mostrar Vista...**

2.2.10 Configuraciones de ejecución

Cuando queremos ejecutar una aplicación desde **Eclipse** necesitamos una configuración de ejecución que es algo así como el conjunto de características que definen el entorno en el que queremos simular la ejecución. Algunas de las cosas que tenemos que definir en ese entorno son:

- El proyecto que queremos ejecutar: seleccionamos uno de los que hemos creado.
- El tipo de aplicación que es: aplicación **Java**, aplicación **C**, aplicación **Android**,...
- Los argumentos con los que se lanza el programa. Cuando un programa está terminado se ejecuta desde el terminal del sistema operativo escribiendo su nombre:

```
C:\> java Suma
```

Pero a veces el programa necesita datos, por ejemplo, el programa **Suma** podría estar diseñado para que le aportes directamente los números a sumar:

```
C:\> java Suma 7 4
```

Esos datos que ponemos tras el nombre del programa se llaman argumentos, en este caso hay dos argumentos **7** y **4**. Para simular desde **Eclipse** la ejecución de un programa de terminal, aportamos los argumentos en la configuración de ejecución.

Podemos definir varias configuraciones de ejecución para una misma aplicación (por ejemplo con argumentos distintos). Una vez definida, cada vez que queramos ejecutar la aplicación sólo tenemos que seleccionar la configuración de ejecución adecuada.

2.2.11 Plugins

Para que **Eclipse** pueda ofrecer ayudas para un lenguaje determinado necesitamos instalar los **plugins** correspondientes (un **plugin** es un añadido, una extensión que aporta nuevas funcionalidades a una aplicación):

- **Java**: la instalación de **Eclipse** que hemos hecho ya reconoce este lenguaje de programación. No necesitamos ningún plugin más.
- **C**: necesitamos instalar el **plugin** para **C/C++**. Para ello, en:

AYUDA/INSTALL NEW SOFTWARE...

seleccionamos:

2022-09 - <https://download.eclipse.org/releases/2022-09/>

y nos vamos a:

Programming Languages/C/C++ Development Tools

Nota1: El plugin **CDT** (C Development Tools) incluye todas las ayudas que necesitamos para desarrollar en **C**, pero no el compilador, que es necesario para poder compilar y ejecutar el programa. Si trabajas con algún SO derivado de **UNIX** ya tienes el compilador de **C**. Si estás trabajando con **Windows** tendrás que instalarlo.

Nota2: En las últimas distribuciones de **Ubuntu** no viene instalado el compilador de **C**. Habrá que instalar los paquetes **gcc** y **make** para disponer de él:

sudo apt get install gcc make

- **Lenguajes de marcas**: también hay plugins para ayudas con lenguajes de marcas.
- **WindowBuilder** es un **plugin** para **Eclipse** desarrollado por **Google** para diseñar pantallas (**GUI** Graphic User Interface - Interfaz Gráfica de Usuario).
- **ER Master** es un **plugin** para crear modelos **E/R**. Lo veremos en este tema.

2.2.12 WindowBuilder

Las aplicaciones destinadas a ejecutarse en un entorno gráfico necesitan una Interfaz Gráfica de Usuario (**GUI**) que es el conjunto de ventanas y controles con los que el usuario interacciona con la aplicación. El diseño de esas pantallas se puede hacer escribiendo código pero, cada vez más, se usan ayudas gráficas para colocar los elementos, dimensionarlos, decorarlos, etc. Esas herramientas se encargarán de traducir el dibujo a código, que después podemos retocar.

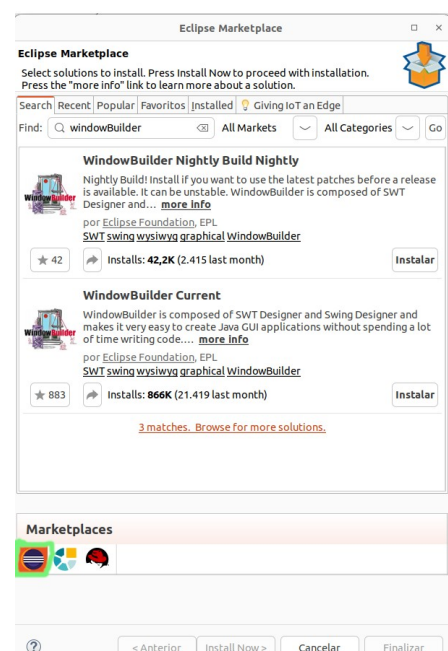
WindowBuilder es un plugin para **Eclipse** que ofrece esta ayuda. Para descargarlo:

AYUDA/ECLIPSE MARKETPLACE

Seleccionamos el icono correspondiente al **Marketplace de Eclipse** (marcado en verde en la imagen) y escribimos:

WindowBuilder

e instalamos.



PARA COMENZAR A USAR WINDOWBUILDER

Sobre un proyecto **Java** abierto (al que queremos añadir una pantalla) seleccionamos:

ARCHIVO/NUEVO/OTRO/WINDOWBUILDER/SWING DESIGNER/APPLICATION WINDOW

Aportamos un nombre y aceptamos.

Observamos que se ha creado una clase **Java** con algún código ya escrito. Esa es la forma en la que se ve la ventana en **Java**. En la parte inferior del editor de textos hay dos pestañas que nos permiten alternar las vistas **Design** (gráfica)/ **Source** (Java).

Si cambiamos a la vista de diseño vemos cuatro ventanas:

- El **formulario**: es la ventana principal donde se dibujará nuestra pantalla. Inicialmente estará vacía a la espera de que vayamos colocando elementos.
- **Paleta**: es un muestrario con todos los elementos disponibles que podemos usar.
- **Estructura**: muestra de forma jerárquica todos los elementos que tenemos colocados.
- **Propiedades**: muestra las propiedades (características) color, tamaño, tipo de letra, etc, del elemento que tengamos seleccionado.

Una pantalla está formada por contenedores y controles:

- Los **contenedores** son cajas que agrupan elementos. Dentro de un contenedor podemos incluir controles y otros contenedores que a su vez podrán tener contenido.
- Los **layouts** son las distintas formas de distribuir los elementos en un contenedor (en una fila, en una columna, en una parrilla, ...). Por ejemplo, **GridLayout** los distribuye en una cuadrícula de filas y columnas, y **AbsoluteLayout** te permite colocar los elementos libremente en el espacio del contenedor.
- Los **controles** son los elementos que interactúan con el usuario y son muy variados: etiquetas, cajas de texto, botones, checkbox, radiobutton, listas desplegables, ... De momento nos vamos a fijar en los siguientes:
 - ◆ **JLabel** es una etiqueta, es decir, contiene un texto para mostrarse al usuario y normalmente el usuario no podrá hacer nada con él (salvo leerlo).
 - ◆ **JButton** es un botón que al ser pulsado desencadenará una acción.
 - ◆ **JTextField** es una caja de texto, un espacio donde el usuario podrá escribir.

PROPIEDADES

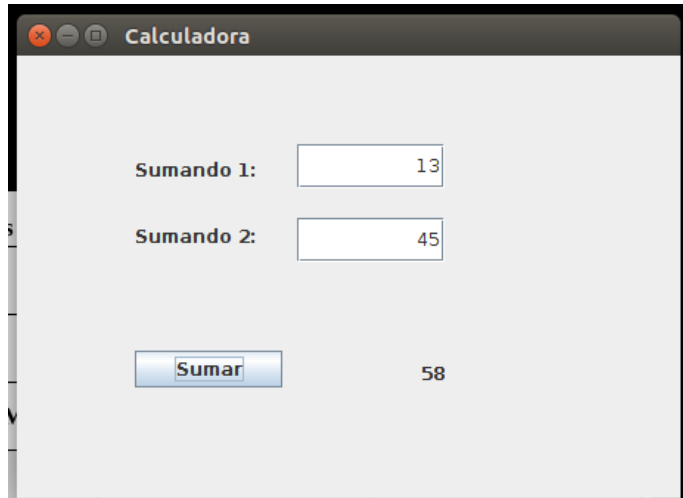
Cada vez que seleccionamos un elemento de la ventana (desde la zona gráfica o desde la estructura), en la ventana de propiedades aparecen todas las características que podemos modificar de ese elemento. Por ejemplo, para un botón se puede modificar su situación, tamaño o rótulo y para un **gridlayout** podemos modificar la cantidad y tamaño de columnas y filas.

De ese modo, para modificar el título de la ventana, seleccionamos el primer elemento de la estructura que es un **frame** que representa a la ventana completa. Una vez seleccionado, buscamos **title** en propiedades y escribimos el título.

UN PEQUEÑO EJEMPLO

Para crear la “**Calculadora**” de la imagen hemos usado una distribución ***absoluteLayout*** sobre el frame principal y hemos colocado:

- dos ***JText*** para recoger el valor de los dos sumandos.
- tres ***JLabel*** para los letreros “Sumando 1:”, “Sumando 2:” y el resultado.
- un ***JButton*** para realizar la suma.



Para dar por finalizada esta aplicación es necesario programar la acción a realizar cuando el usuario pulse el botón. A continuación se muestra el código incluido:

```
// código a ejecutar al pulsar el botón
btnSumar.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
        // Aquí tú código
        int sumando1 = Integer.parseInt(txtSumando1.getText());
        int sumando2 = Integer.parseInt(txtSumando2.getText());
        int resultado = sumando1 + sumando2;
        etqResultado.setText(String.valueOf(resultado));
    }
});
// fin del código
```

En este ejemplo hemos usado los siguientes métodos:

- ***getText()***: Devuelve el ***String*** del contenido de la caja de texto (lo que haya introducido el usuario).
- ***SetText(valor)***: Modifica el contenido de una etiqueta al valor (***String***) indicado.
- ***Integer.parseInt(valor)***: Convierte a entero el ***String*** recibido.
- ***String.valueOf(valor)***: Convierte a ***String*** el entero recibido.

Nota: Ten en cuenta que los conocimientos de programación son parte de otro módulo. Aquí sólo intentamos dar unas pinceladas para poder finalizar un ejemplo completo.

2.2.13 ER Master

ER Master es un plugin para **Eclipse** para modelar esquemas **Entidad/Relación (E/R)**. En este tema veremos unas nociones básicas sobre **ER Master**, si quieres la información completa puedes dirigirte a la página <http://ermaster.sourceforge.net>.

Nota: Es posible que la instalación de **ER Master** en **Eclipse** sobre **Ubuntu** no funcione correctamente. Es buena idea instalarlo en **Eclipse** sobre **Windows**.

Para instalarlo, desde el menú **Ayuda/Install New Software...**, pulsa **Añadir...** para añadir un repositorio con los siguientes datos:

- Nombre: **ER Master**
- Ubicación: <http://ermaster.sourceforge.net/update-site/>

y sigue las instrucciones hasta completar la instalación que terminará reiniciando **Eclipse**.

PARA CREAR UN MODELO:


Situados sobre un proyecto **Java**, **Archivo/Nuevo/Otros.../ER Master/ER Master**, aportar un nombre para nuestro modelo y seleccionar el SGBD **MySQL** (u otra que prefieras).


Verás que la pantalla de **ER Master** tiene 3 zonas:

- **Vista de edición:** la zona donde dibujamos.
- **Barra de botones:** (arriba) aparecen botones nuevos para su uso con este plugin.
- **Paleta de diseño:** (izquierda) te ofrece todos los elementos que puedes incluir.


NOTACIÓN UML PARA ENTIDAD/RELACIÓN

ER Master no usa la notación tradicional para representar el diagrama **Entidad/Relación**, sino la notación **UML (Unified Modeling Language)**. En esta notación las tablas se representan en rectángulos con el nombre de la tabla en la parte superior y debajo un atributo por línea. Las relaciones se muestran con líneas de tabla a tabla que expresan las **participaciones** de la siguiente forma:

➤ (0,1): 

➤ (1,1): 

➤ (0,n): 

➤ (1,n): 

CREAR UNA TABLA

Seleccionamos "**Table**" en la paleta y arrastramos el ratón dentro de la vista de edición, con lo que saldrá dibujada la tabla. Con doble clic sobre la tabla podremos acceder a la ventana donde describimos la tabla "**Table Information**". Tendrás que indicar:

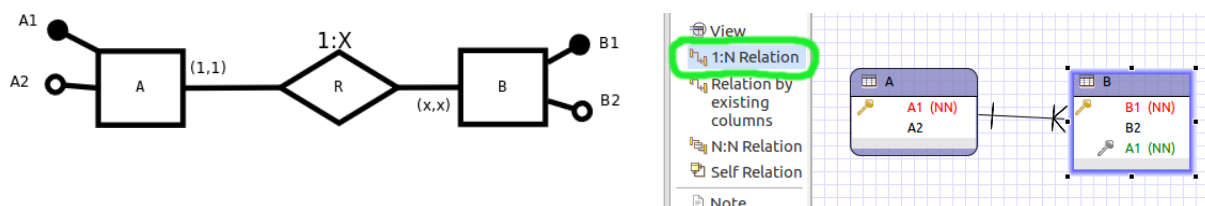
- **Nombre** de la tabla.
- Cada uno de los **campos** (al pulsar el botón **add** te permite detallar todas las características del campo: nombre, tipo, tamaño, si es clave primaria,...).

CREAR RELACIONES

Una vez creadas todas las entidades que aparecen en tu modelo clásico, para crear las relaciones distinguimos dos casos:

- **Aquellas relaciones en las que la transformación al modelo relacional se propaga la clave:** (son todas las relaciones que tienen una participación (1,1) en alguno de los lados).

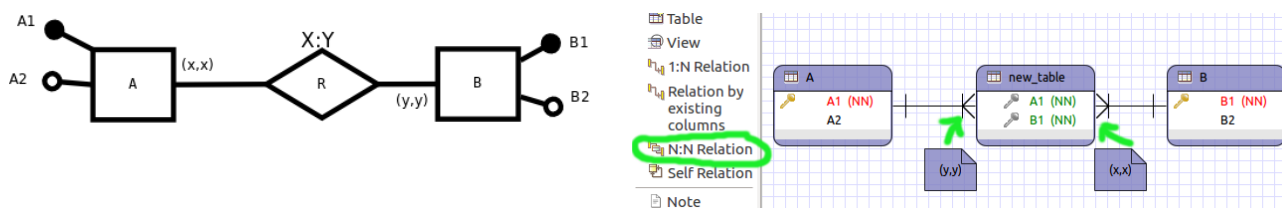
Para estas relaciones seleccionamos “1:N Relation” y arrastramos el ratón desde la tabla con participación (1,1) hasta la otra¹. Al hacer eso, en la segunda tabla aparecerá la clave de la primera, ya tratada como clave ajena. Quizá sea conveniente modificar el nombre del campo que ha aparecido.



También habrá que indicar como se reacciona ante modificaciones (**UPDATE**) y borrados (**DELETE**) y la participación de la segunda tabla. Para eso, hacemos doble click en la línea de la relación.

- **Aquellas relaciones en las que la transformación al modelo relacional genera una nueva tabla:** (todas las que no admiten la propagación de clave).

Para estas relaciones seleccionamos “N:N Relation” y arrastramos el ratón desde una tabla a la otra. Al hacer eso, aparece una nueva tabla que incluye las claves de ambas, ya tratadas como claves ajenas.



1 Si la participación en ambos lados es (1,1), se puede elegir cual de las claves se propaga. Una de ellas, pero no las dos.

Sería conveniente cambiar el nombre de esos campos y tendremos que indicar para la nueva tabla: la **clave primaria**, el **nombre de la tabla** y el **resto de atributos** si faltaran.

Además en la línea de la relación, con doble click podemos indicar como reaccionar **ON UPDATE** y **ON DELETE** y las **participaciones** (la participación que sale en las tablas originales es 1 y eso no hay que modificarlo. Sin embargo en la parte de la tabla nueva hay que cruzar lo que tengamos en el diseño: observa las posiciones de (x,x) e (y,y) en las dos imágenes).

La opción “**Self Relation**” (autorelación) permite crear una relación reflexiva que se resuelva con propagación de clave. Para las reflexivas que generan nueva tabla, creamos la nueva tabla manualmente y relacionamos las columnas existentes. Las líneas salen superpuestas, tendrás que mover una de ellas para diferenciarlas.

Con “**Relation by existing columns**” (relación entre columnas existentes) podemos crear las relaciones en el caso de que ya tengamos todos los campos y claves ajenas en sus tablas. Con esta opción no se añaden campos nuevos, sólo permite relacionarlos arrastrando desde la tabla referenciada hasta la tabla que contiene la clave ajena. Habrá que indicar cual es esa clave ajena y detallar **UPDATE**, **DELETE** y las participaciones.

EXPORTAR ESQUEMA A OTROS FORMATOS

Una vez creado el esquema **E/R** podemos exportarlo a montones de formatos. Para ello, hacemos **clic-derecho sobre la zona de edición/Exportar** y elegimos el formato:

- **DDL**: Lenguaje de Definición de Datos. Directamente obtiene las sentencias **SQL** para crear la **BD** correspondiente al esquema diseñado.
- **Hoja de cálculo**: Crea una hoja de cálculo con información muy completa sobre la **BD**.
- **HTML**: La misma información que en la anterior opción, pero en formato **HTML**.
- **Imagen**: Una imagen del esquema **E/R**.
- **Diccionario**: Información de las tablas en formato **csv**.
- **Java**: Crea las clases **Java** correspondientes al esquema **E/R**.

GENERAR LA BASE DE DATOS

También podemos generar la propia base de datos usando uno de los botones (ver imagen de la derecha) de la barra superior (**Export to BD**).



Nota: Para exportarlo a un **SGBD** tendremos que elegir el que deseamos (por ejemplo, si al crear el esquema señalamos **MySQL** podremos exportar a **MySQL**). Para poder exportar a otros **SGBD** (como **Oracle**) tendremos que cambiarlo con **clic-derecho sobre la zona de edición/Option**

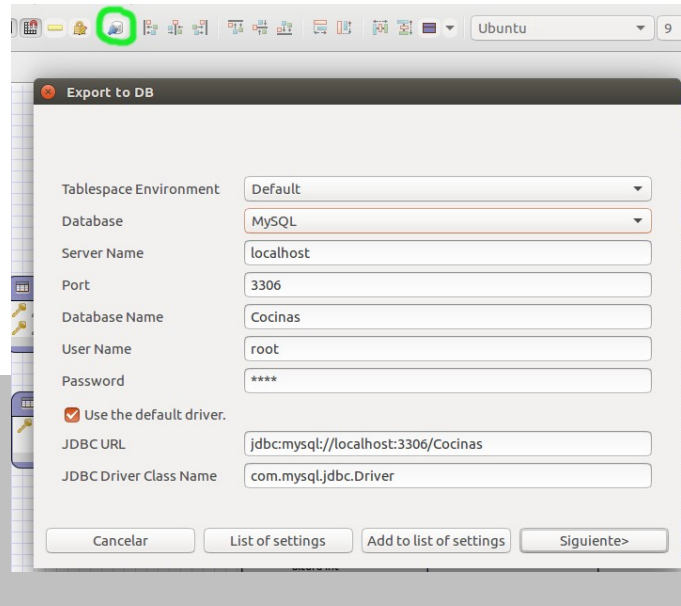
La exportación a **MySQL** requiere que la base de datos esté previamente creada desde **MySQL** y habrá que indicar el nombre de usuario y contraseña.

También será necesario contar con una librería Java (**mysql-connector-j-8.0.31.jar**) que contiene el conector para **MySQL**. Desde www.mysql.com/products/connector/ se puede descargar y hay que guardarlo en un lugar conocido, por ejemplo **/lib/mysql**, si estamos bajo **Ubuntu**.

Si estamos bajo el Sistema Operativo **Windows**, debemos incluir el conector (**JDBC** – Java DataBase Connector) durante la instalación de **MySQL**. Cuando se nos pregunte por su ubicación podemos buscar por **C:\Program Files (x86)\MySQL\Connector J 8.0** o ruta similar.

Nota: Al trabajar con **Java** será frecuente que necesites usar librerías. Es una buena práctica guardarlas siempre en una carpeta con el nombre adecuado en **/lib**.

Nota: La misma librería se usa en **Linux** y en **Windows**.



IMPORTAR ESQUEMA DESDE UNA BD

Otra de las posibilidades que ofrece **ER Master** es importar una **BD** ya creada. Para usar esta opción necesitas tener instalado un **SGBD** y conocer su usuario y contraseña, de modo que se pueda hacer la conexión a la **BD**.

Si tienes todo esto, **clic-derecho sobre la zona de edición/Importar/BD**.

Nota: También para importar la base de datos se necesita la librería con el conector para **MySQL**.

2.2.14 Otras ayudas interesantes

Eclipse ofrece indentación automática pulsando simultáneamente las teclas **CTRL** + **SHIFT** + **F**, o **CTRL** + **I** para indentar sólo la parte de código seleccionado.

Cuando veas que empiezas a escribir una etiqueta y no aparecen ayudas para terminarla prueba a pulsar **CTRL** + **espacio**. Si aún así, no aparecen las ayudas lo más probable es que haya un error previo (sintáctico o semántico) que le impida reconocer las etiquetas adecuadas al punto en el que estás.

Para viajar rápidamente hasta el código de un método basta con hacer **CTRL** -**clic** sobre el nombre del método en cualquier parte del código. Esto es especialmente útil para ver el código fuente de métodos o clases que pertenecen a librerías externas.

CTRL + **7** para comentar (o quitar comentario) en el trozo de código seleccionado.

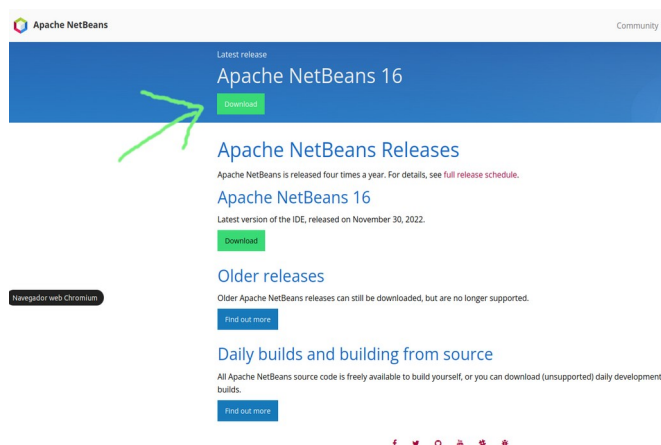
Puedes visualizar una lista completa de atajos con **CTRL** + **SHIFT** + **L** y puedes encontrar listados muy útiles si haces una búsqueda en la web con “**Eclipse atajos teclado**”.

2.3 Netbeans

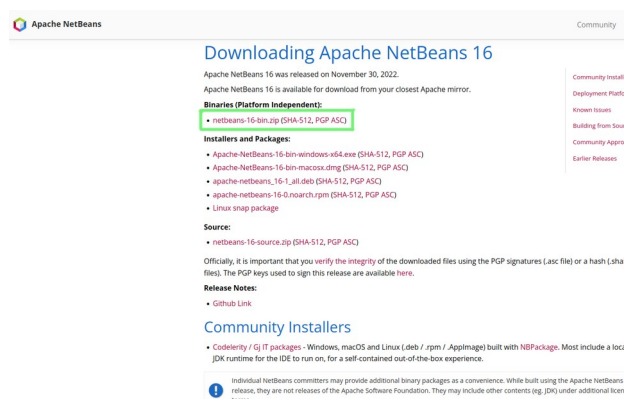
Netbeans es un entorno integrado de uso general. Como **Eclipse**, es gratuito.

2.3.1 Descarga

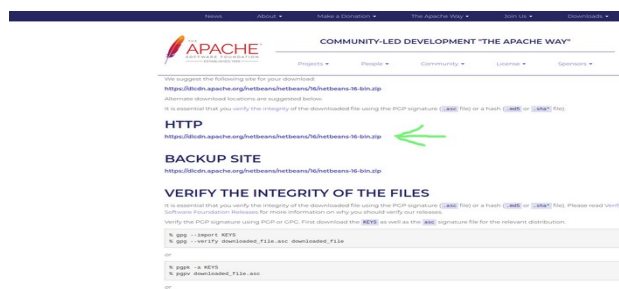
Puedes acceder a la página oficial en <https://netbeans.org/downloads/>. Hasta hace un tiempo, cada año se liberaban cuatro versiones de **Netbeans**: una **LTS** abril (.0) y otra cada tres meses. Con la versión **12** se perdió esa regularidad y nunca más se han publicado versiones **LTS** (o al menos, sus nombres no indican que sean **LTS**). En el momento de escribir este documento la última versión es la de noviembre de 2022, la **16**. Esa es la versión recomendada porque es la única soportada. (También la tienes disponible en la **Moodle**).



En la primera pantalla elegimos **Download** de la versión **16**.



En esta segunda pantalla seleccionamos el fichero **zip** que tiene los **binarios**.



En esta última pantalla puedes elegir cualquiera de los servidores para iniciar la descarga.

2.3.2 Instalación

El fichero descargado nos sirve para **Linux** y para **Windows**. Lo único que hay que hacer para instalar **Netbeans 16** es descomprimir el fichero **zip** descargado y llamar al ejecutable. En el caso de **Ubuntu**, lo vamos a situar en la carpeta **/opt** que es el lugar adecuado para las aplicaciones no integradas en los repositorios.

Extrae el contenido del fichero **zip** en algún sitio, por ejemplo: el escritorio (el lugar no es importante porque al finalizar lo podremos eliminar). Aparecerá una carpeta llamada **netbeans-16-bin** que aloja a otra carpeta llamada **netbeans**. Vamos a mover esta última hasta **/opt**:

```
sudo mv /home/administrador/Escritorio/netbeans-16-bin/netbeans /opt
```

Nota: en amarillo la ruta donde has extraído el fichero **zip** que, en tu caso, puede ser otra.

Para ejecutar el **IDE** hay que escribir la orden (requiere que tengamos instalado un **JDK**):


```
/opt/netbeans/bin/netbeans
```

Si añades la ruta **/opt/netbeans/bin** a la variable de entorno **PATH** editando el fichero **environment** (`sudo gedit /etc/environment`), podrás ejecutarlo con solo escribir:

```
netbeans
```

Si quieres que aparezca en el lanzador de aplicaciones debes crear un fichero llamado **netbeans.desktop**, y situarlo en la carpeta **~/.local/share/applications**, con el siguiente contenido:

```
[Desktop Entry]
Name=Netbeans
Comment=Netbeans
Exec=/opt/netbeans/bin/netbeans
Icon=/opt/netbeans/nb/netbeans.icns
Terminal=false
Type=Application
```

Si no aparece en el lanzador de aplicaciones, búscalo en el **menú del lanzador de aplicaciones** (). Si además lo quieres mantener allí permanentemente, desde el propio menú del lanzador de aplicaciones:

```
CLIC DERECHO/AÑADIR A FAVORITOS
```

Nota: no es posible trabajar en castellano con **Netbeans** en versiones posteriores a la **8.2**.

Nota: Si tienes problemas para crear proyectos **Java** es posible que la ruta donde busca el **JDK** no sea correcta. Puedes modificarla en el archivo de configuración que se encuentra en

```
/ruta de instalación de Netbeans/etc/netbeans.conf
```

Edita ese fichero y busca **netbeans_jdkhome** para indicar la ruta correcta:

```
netbeans_jdkhome="rutaCorrectaDelJDK"
```

2.3.3 Uso básico de Netbeans

La pantalla inicial de **Netbeans** tiene un panel de navegación (**Projects**) a la izquierda, la zona de edición a la derecha y la ventana de salida (**output**) en la inferior. Puedes localizar estas u otras ventanas en el menú **Window**.

Una diferencia importante con **Eclipse** es que en **Netbeans** no se ven los proyectos cerrados, sólo los que están abiertos. Se aconseja localizar todos los proyectos **Netbeans** en una misma carpeta para organizarlos.

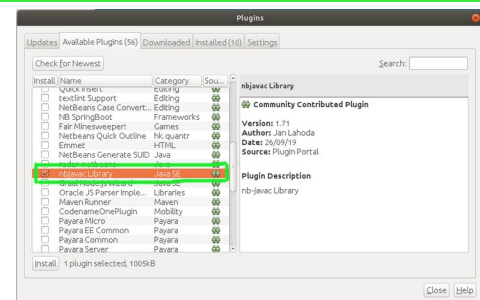
El navegador de proyectos, muestra la estructura de archivos, tal y como la mantiene el SO.

PLUGINS PARA JAVA

Para trabajar con el lenguaje **Java** **no necesitamos añadir nada**. No obstante, si tuvieras problemas, añade el **plugin**:

TOOLS/PLUGINS/AVAILABLE PLUGINS

Buscamos un plugin llamado **nbjavac** y lo marcamos para instalar. A continuación pulsa en **Install** y sigue las instrucciones.

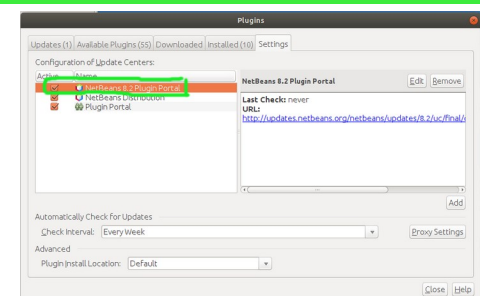


PLUGINS PARA C

De entrada no aparece el soporte para programar con los lenguajes **C** o **C++**. Tenemos que seguir los siguientes pasos:

Activar el **portal de plugins de Netbeans 8.2** desde:

TOOLS/PLUGINS/SETTINGS

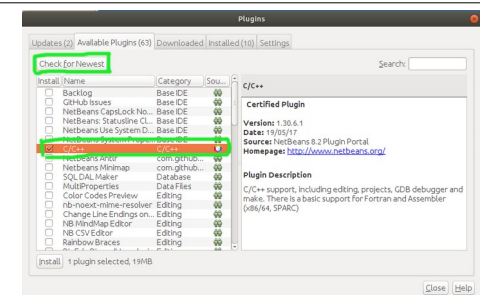


Actualizar la lista de plugins desde:

TOOLS/PLUGINS/AVAILABLE PLUGIN/CHECK FOR NEWEST

Instalar el plugin para **C** y **C++** desde:

**TOOLS/PLUGINS/AVAILABLE PLUGIN/
C/C++ /INSTALL**



ABRIR UN PROYECTO

FILE/OPEN PROJECT...

y navegamos por las carpetas del sistema.

Nota: Cuando navegamos por las carpetas del sistema de archivos debemos prestar atención al icono que aparece a la izquierda de cada carpeta. Si **Netbeans** reconoce esa carpeta como un proyecto le asignará el icono adecuado (una taza para **Java**, una cajita con una rueda dentada para **C**, etc). Si aparece un icono de una carpeta, significa que **Netbeans** no lo reconoce como proyecto.

CREAR UN PROYECTO NUEVO JAVA**FILE/NEW PROJECT/JAVA WITH ANT/JAVA APPLICATION**

escribimos el nombre del proyecto y la localización y nos fijamos en la última línea de la ventana donde podremos decidir que cree una clase y, en ese caso, el nombre de la clase.

CREAR UN PROYECTO NUEVO C**FILE/NEW PROJECT.../ C/C++ / C/C++ APPLICATION/NEXT**

escribimos el nombre del proyecto y ubicación y también tenemos la posibilidad de solicitar que se genere la función principal **main** y el nombre del fichero fuente. Presta atención a la selección **C** o **C++**.

Nota: Los proyectos, por defecto, se almacenan en carpetas del mismo nombre ubicados en una carpeta llamada **NetbeansProjects**. No obstante, y aunque en general no se recomienda, puedes guardar los proyectos en cualquier otro sitio.

CREAR UNA CLASE JAVA

Nos situamos, dentro de la jerarquía del proyecto **Java**, en "**Source packages**" y hacemos:

CLIC-DERECHO/NEW/JAVA CLASS.../

escribimos el nombre de la clase, recordando que debe iniciarse con una letra mayúscula.

CREAR UN FICHERO FUENTE C

Nos situamos, dentro de la jerarquía del proyecto **C**, en "**Source Files**" y hacemos:

CLIC-DERECHO/NUEVO/C SOURCE FILE

y escribimos el nombre y extensión del fichero. La extensión normalmente será **.c**.

EJECUTAR UN PROYECTO**RUN/RUN PROJECT****BORRAR UN PROYECTO****CLIC-DERECHO SOBRE EL PROYECTO/DELETE**

Si quieres eliminar los ficheros del proyecto definitivamente debes marcar la casilla que aparece a continuación. Si no la marcas, el proyecto deja de estar visible en **Netbeans**, pero no se eliminan los ficheros.

USAR PARÁMETROS DE LÍNEA DE COMANDOS

Cuando nuestro programa necesite que incluyamos argumentos desde la línea de comandos, podemos simularlo desde:

RUN/SET PROJECT CONFIGURATION/CUSTOMIZE.../RUN

En la casilla **arguments** escribimos separados por espacios todos los argumentos.

IMPORTAR UN PROYECTO CREADO CON ECLIPSE

Cuando deseemos incorporar un proyecto **Netbeans** que traemos desde otro equipo, no es necesario importar, solamente habrá que hacer una copia del proyecto en nuestro equipo, lo ideal es que sea en la carpeta **NetbeansProjects** y a continuación, abrirlo.

En **Netbeans** sólo se usa el concepto **importar** para traer proyectos que han sido generados con otras herramientas, como **Eclipse**.

Para importar un proyecto generado con **Eclipse**:

FILE/IMPORT PROJECT/ECLIPSE PROJECT.../

seleccionamos "**Importar proyectos desde el espacio de trabajo**" ("**Import projects from workspace**") y en "**Ubicación del espacio de trabajo**" ("**Workspace Location**") señalamos la ruta del **workspace** que contiene el proyecto (podemos ayudarnos con el botón **Browse...**), pulsamos en **Next** y elegimos el proyecto deseado.

Presta mucha atención a la selección que puedes hacer en la parte inferior de la pantalla:

- **Almacena información sobre el proyecto Eclipse (store Netbeans project data inside Eclipse project folders)**: con esta opción **Netbeans** trabajará directamente sobre los ficheros originales, y los cambios que se hagan desde un entorno serán visibles desde el otro. Es decir, se trabaja sobre la misma copia.
- **Crear proyecto Netbeans en ubicación separada (create imported Netbeans projects in a separate location)**: Esta opción que crea una copia nueva en **Netbeans**, pero copia todo salvo los fuentes, que siguen siendo los originales en el espacio de trabajo de **Eclipse**. Por tanto, los cambios hechos con un **IDE** se verán en el otro también.

CAMBIAR EL NOMBRE DE UN FICHERO O PROYECTO

Nos situamos, en el navegador de la izquierda, sobre el elemento a renombrar:

CLIC - DERECHO / RENAME

Presta atención cuando cambies el nombre a un proyecto: aparece la opción de renombrar también (o no) la carpeta que lo contiene. Normalmente deberías cambiar esta casilla para que los nombres de carpetas y los nombres de proyectos guarden coherencia.

SUGERENCIAS NETBEANS

Cuando tecleamos código podemos llamar al servicio de sugerencias de **Netbeans** con la combinación **Alt** + **Enter**. **Netbeans** mostrará sugerencias para resolver problemas o continuar escribiendo código.

CTRL + **espacio** completa código.

ALT + **SHIFT** + **F** para autoindentación.

CTRL + **SHIFT** + **C** para comentar/descomentar el trozo seleccionado.

Una búsqueda en la web con el texto "**atajos teclado netbeans**" te proporcionará páginas muy útiles para los atajos.

2.3.4 GUI Builder

Sin necesidad de añadir un plugin, **Netbeans** incluye una herramienta llamada **GUI Builder** para generar el interfaz gráfico. Comenzaremos creando un proyecto **Java** sin clase principal, porque la clase será la ventana que la crearemos posteriormente. Dentro de ese proyecto seleccionamos:

CLIC DERECHO / NEW / OTHER... / SWING GUI FORMS / JFRAME FORM

y damos nombre a la clase que generará la ventana.

GUI Builder dispone de las siguientes zonas:

- **Área de trabajo**: En la zona central. Donde se puede crear y modificar la ventana y sus componentes, bien de forma gráfica (**Design**) o bien con código Java (**Source**).

En la parte superior de esta zona tenemos los botones que nos permiten alternar entre ambas.

- **Paleta:** Situada en la derecha, arriba. La paleta contiene todos los elementos que podemos incorporar a nuestra ventana: botones, cuadros de texto, etiquetas, etc.
- **Propiedades:** En la derecha, abajo. La ventana de propiedades muestra las características del elemento que selecciones en cada momento y permite modificarlas. Por ejemplo, la propiedad **text** establece el texto que inicialmente tienen las etiquetas y cuadros de texto o el rótulo en los botones, y la propiedad **horizontalAlignment** permite establecer la alineación del texto.

Nota: Si no se muestra la ventana de **Propiedades**, puedes hacerla visible desde el menú contextual del elemento con el que estés trabajando.

Una vez que se ha diseñado la pantalla es conveniente revisar el código y poner nombres adecuados a todos los elementos. Por ejemplo, si tienes un botón con el rótulo “**Aceptar**”, en el código Java aparecerá como, por ejemplo, **jButton7** y mejorarás mucho si lo cambias a, por ejemplo, **btnAceptar**. Para cambiar el nombre, desde el código debes hacer:

```
CLIC DERECHO/REFACTOR/RENAME
```

Para añadir funcionalidad a los botones, desde la vista diseño:

```
CLIC DERECHO/EVENTS/ACTION/ACTIONPERFORMED
```

(o bien doble-clic sobre el botón).

Al hacer esto, en el código aparecerá el siguiente método:

```
private void nombreBotonActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

dentro de este método tendrás que codificar la acción que se ejecuta al pinchar el botón.

2.4 Android Studio

Android Studio es un entorno integrado orientado a la programación de apps para **Android**. Este IDE es propiedad de **Google**, al igual que el sistema **Android**, lo que debe suponer una ventaja de cara a otras alternativas de programación para **Android**. No obstante, también es posible hacerlo con **Eclipse** y el plugin **ADT**.

El desarrollo para **Android** requiere:

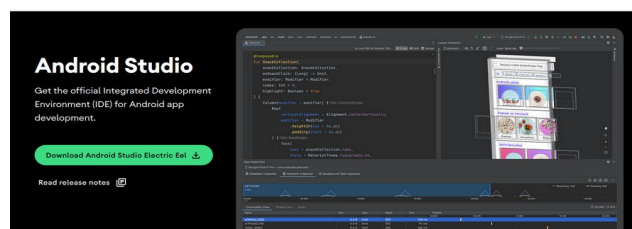
- El **JDK** (Java Development Kit – Kit de desarrollo para Java): Tenemos dos alternativas: **openjdk** y el **JDK** de **Oracle**. En el tema 1 vimos cómo instalarlo.
- El **IDE Android Studio**: Es un IDE muy completo que permite acceso directo a otras aplicaciones que nos pueden ser útiles, como el **Android SDK** o el gestor de **AVD**.
- El **SDK** de Android (Android Software Development Kit – Kit de desarrollo de Software para Android): El **Android SDK** puede ser gestionado a través de una aplicación que nos permite ver los paquetes que tenemos instalados y realizar las descargas.
- **AVD** (Android Virtual Device – Dispositivo Virtual Android): Los **AVD** son emuladores de terminales reales en los que se ejecutan las app. Se trata de smartphone, tablet, smartTV, etc concretos.

2.4.1 Descarga en Windows

Desde <https://developer.android.com/studio/install.html> se descarga en formato **.exe**, ejecutable. Las pantallas son las mismas que en la descarga para otros sistemas operativos. La página detecta tu sistema y te ofrece el fichero adecuado.

2.4.2 Descarga en Ubuntu

Desde <https://developer.android.com/studio/install.html> se descarga en formato **.tar.gz**.



Lo extraemos en el escritorio y movemos la carpeta a **/opt**, con **sudo**:

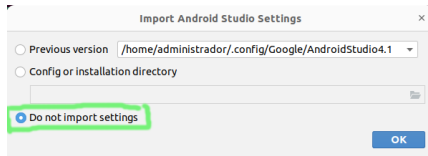
```
sudo mv ~/Escritorio/android-studio/ /opt/android-studio/
```

Con esto ya estará plenamente operativo. Para ejecutarlo, la orden será:

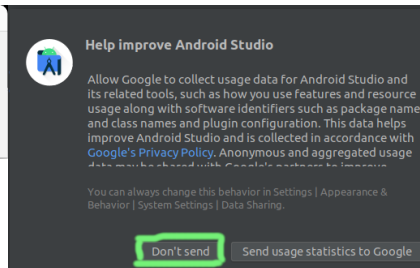
```
sh /opt/android-studio/bin/studio.sh
```

2.4.3 Instalación (Windows y Ubuntu)

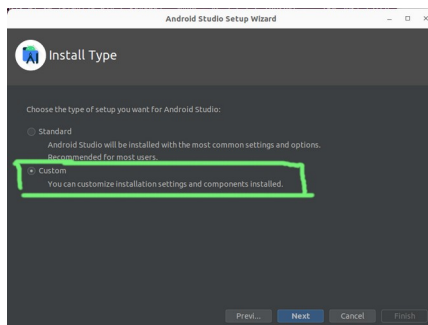
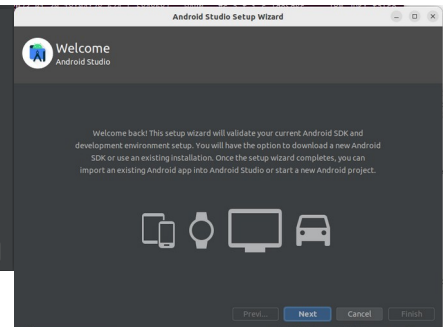
La primera vez que lo ejecutamos nos hará unas preguntas fáciles antes de iniciarse:



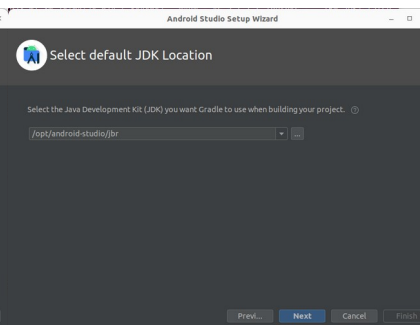
Si queremos importar configuración de una instalación anterior



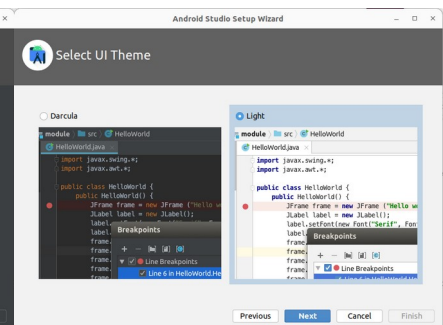
Si queremos enviar datos de uso a Google



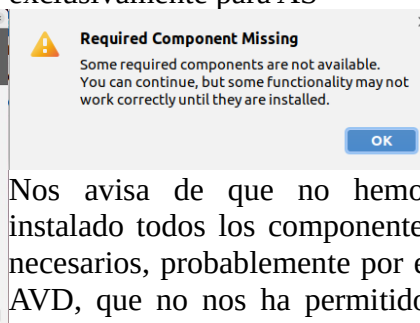
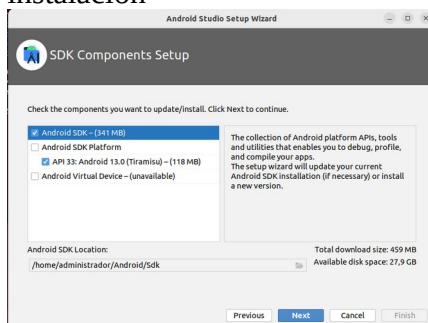
Elegimos personalizar la instalación



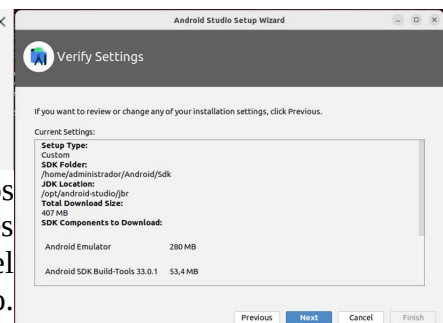
Aceptamos usar un JDK exclusivamente para AS



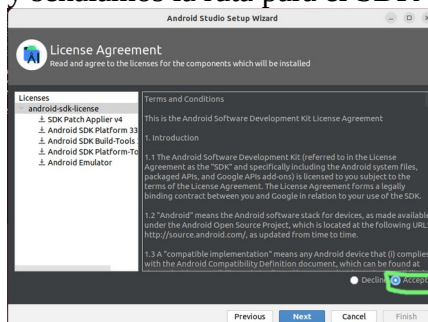
Elegimos el tema que nos guste



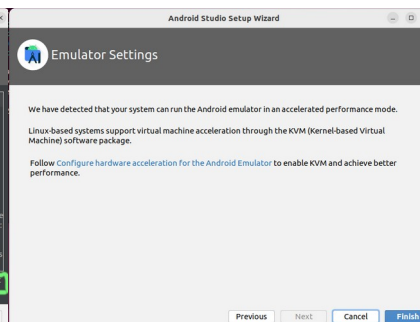
Nos avisa de que no hemos instalado todos los componentes necesarios, probablemente por el AVD, que no nos ha permitido. Más adelante veremos qué hacer



Instalamos todo lo que nos deje y señalamos la ruta para el SDK²



Aceptamos condiciones



Y finalizamos

² Tendrás que crear una carpeta dedicada en exclusiva para el **SDK**. Se recomienda que lo hagas en algún lugar del árbol que cuelga de tu carpeta personal.

Nota: La recomendación de ubicar la carpeta para el **SDK** en el lugar adecuado toma más importancia si tienes el disco con dos particiones (una para la raíz del sistema (/) y otra para los datos de usuario (/home)) ya que las descargas de **SDK** pueden ser muy voluminosas y obligarte a modificar el tamaño de la partición del sistema. Si ubicas muchas aplicaciones en /opt o incluyes allí la carpeta del **SDK** quizá te interese añadir una tercera partición para /opt.

2.4.4 Versiones Android Vs niveles API

Antes de empezar un proyecto en **Android** hay que elegir la versión del sistema para la que deseamos realizar la app. Es muy importante observar que hay clases y métodos que están disponibles sólo a partir de una determinada versión. Si las vamos a usar hemos de conocer la versión mínima necesaria.

<i>Versión de Android</i>	<i>Nivel de API</i>	<i>Nombre Comercial</i>	<i>Nombre Interno</i>	<i>Fecha</i>
Android v1.0	API 1	Apple Pie		Septiembre 2008
Android v1.1	API 2	Banana Bread	Petit Four	Febrero 2009
Android v1.5	API 3	Cupcake	Cupcake	Abril 2009
Android v1.6	API 4	Donut	Donut	Septiembre 2009
Android v2.0	API 5	Éclair	Eclair	Octubre 2009
Android v2.0.1	API 6	Éclair 0.1	Eclair	
Android v2.1.x	API 7	Éclair MR1	Eclair	Enero 2010
Android v2.2.x	API 8	Froyo	Froyo	Mayo 2010
Android v2.3 a v2.3.2	API 9	Gingerbread	Gingerbread	Diciembre 2010
Android v2.3.3 a v2.3.4	API 10	Gingerbread MR1	Gingerbread	
Android v3.0.x	API 11	Honeycomb	Panal	Febrero 2011
Android v3.1.x	API 12	Honeycomb MR1	Panal	Mayo 2011
Android v3.2	API 13	Honeycomb MR2	Panal	Julio 2011
Android v4.0 a v4.0.2	API 14	Ice Cream Sandwich	Sandwich de Helado	octubre 2011
Android v4.0.3 a v4.0.4	API 15	Ice Cream Sandwich MR1	Sandwich de Helado	Diciembre 2011
Android v4.1 a v4.1.1	API 16	Jelly Bean	Jelly Bean	Julio 2012
Android v4.2 a v4.2.2	API 17	Jelly Bean MR1	Jelly Bean	Noviembre 2012
Android v4.3	API 18	Jelly Bean MR2	Jelly Bean	Julio 2013
Android v4.4	API 19	Kitkat	Key Lime Pie	Octubre 2013
Android v4.4W (Wearables)	API 20	Kitkat_Watch	Key Lime Pie	
Android v5	API 21	Lollipop	Lemon Meringue Pie	Noviembre 2014
Android v5.1	API 22	Lollipop MR1	Lemon Meringue Pie	
Android 6.0-6.0.1	API 23	Marshmallow	Macadamia Nut Cookie	Octubre 2015
Android 7.0	API 24	Nougat	New York Cheesecake	Junio 2016
Android 7.1	API 25	Nougat	New York Cheesecake	
Android 8.0 y 8.1	API 26 y 27	Oreo	Oatmeal Cookie	Agosto 2017
Android 9.0	API 28	Pie	Pistachio Ice Cream	Agosto 2018
Android 10.0	API 29	Android 10	Quince Tart	Septiembre 2019
Android 11.0	API 30	Android 11	Red Velvet Cake	Septiembre 2020
Android 12.0 y 12L	API 31	Android 12	Snow Cone	Octubre 2021
Android 13.0	API 32	Android 13	Tiramisú	Agosto 2022

Puedes ver los detalles incorporados en cada versión de **Android** en:

http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

Cuando se ha lanzado una nueva plataforma siempre ha sido compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades y en el caso de modificar alguna funcionalidad no se elimina, se etiquetan como obsoletas pero se pueden continuar utilizando.

La anterior tabla establece la relación entre la versión de **Android** de un dispositivo y la **API** (Application Programming Interface - Interfaz de Programación de Aplicaciones) que debes usar para programar ese dispositivo. El nivel de **API** corresponde a números enteros comenzando desde 1. Para los nombres comerciales se han elegido nombre de postres en orden alfabético. También para los nombres internos que a veces coinciden, pero no siempre.

A la hora de crear el proyecto de nuestra aplicación debemos especificar:

- La **API** con la que compilar el proyecto (normalmente elegiremos la última). Sólo podremos compilar con las **APIs** que hayamos instalado desde el **SDK** de **Android**, y no conviene instalar más de una porque son muy pesadas.
- La **mínima API** necesaria para poder ejecutar la aplicación sin problemas en el terminal. Debería ser la **API** más baja que soporte las capacidades programadas. A día de hoy, si cogemos la **API 24** abarcamos el 94,4% del mercado; con la **API 33**, el 5,2%. Cuanto más baja, menos funcionalidad, cuanto más alta, menos dispositivos. En la ventana de creación del proyecto informa de estos porcentajes que, naturalmente, van cambiando.

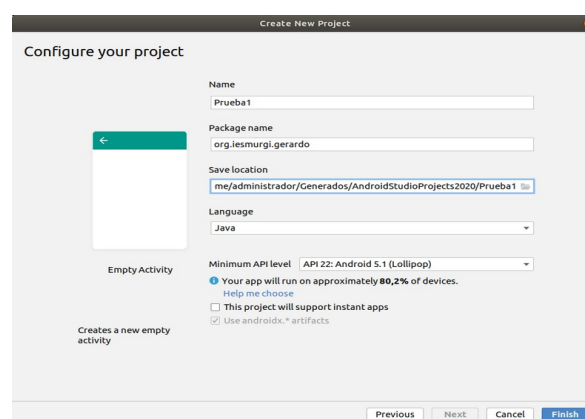
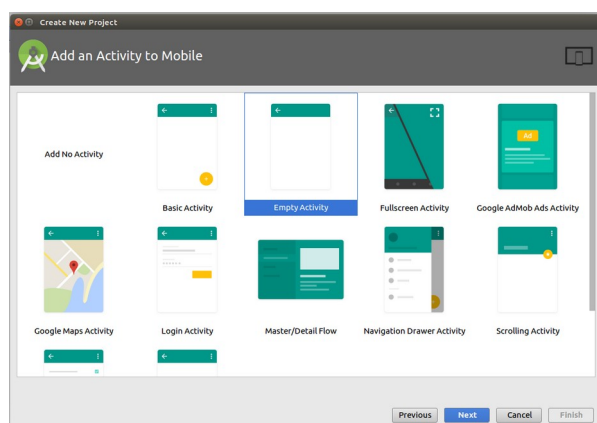
Nota: **Google** también publica los porcentajes de tamaños de pantalla. Si ya has colgado una app en **Play Store** podrás consultar los datos concretos de los usuarios de tu app.

2.4.5 Crear un proyecto

Creamos un nuevo proyecto desde la opción:

FILE/NEW/NEW PROJECT...

Elegimos el tipo de app de modo que pueda generar automáticamente parte del código. Si queremos partir de cero elegiremos el primero (sin actividad). En nuestro caso elegiremos **EmptyActivity** (con una actividad vacía).



En la siguiente pantalla nos pregunta por el nombre de la app que vamos a crear y la ubicación del proyecto en el sistema de archivos. Observa que también pregunta el dominio de la compañía para la que estamos programando. Esto es de utilidad para nombrar los paquetes de forma única, tal y como comentamos anteriormente en este tema.

Podemos elegir el lenguaje de programación: **Kotlin** o **Java** y la **API** mínima que admitirá. Observa que aparecen los porcentajes actuales de terminales que abarcamos con esa elección.

2.4.6 Estructura de un proyecto: ficheros y carpetas que lo componen

AndroidManifest.xml

Se trata del fichero principal de nuestro proyecto que, en formato **XML**, contiene cuestiones generales de la aplicación, entre ellas el **SDK** mínimo para el que funciona, el nombre del paquete **Java**, los componentes de la aplicación y sus características. He aquí un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bol01.ejercicio01">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Cada aplicación se construye con una serie de componentes, entre ellos, las llamadas actividades. Una aplicación tendrá como mínimo una actividad que será la actividad principal.

Recuerda: Aunque tengamos el código de una actividad implementado correctamente, ésta no podrá ejecutarse si no aparece en el fichero **AndroidManifest.xml**.

Para cada componente hay que indicar sus habilidades mediante los llamados **intent**. Por ejemplo, la actividad principal debe poder mostrarse en el lanzador de aplicaciones y ser ejecutada desde allí. Cuando el usuario instala una app en un dispositivo móvil, aparece la lista de permisos que tendrá esa app. Esa lista se genera a partir de la información del **manifest**. El programador no podrá ocultar los permisos de la app porque si no los declara en el **manifest** no tendrán efecto.

Recuerda: Si la actividad principal no incluye un **intent** con la acción **MAIN** y la categoría **LAUNCHER** no podrá iniciarse la aplicación en el dispositivo.

El **manifest** del ejemplo corresponde a una aplicación que contiene una actividad que, a nivel interno, será una clase **Java** llamada **MainActivity**, alojada en el paquete **com.bol01.ejercicio01**. Esa actividad tendrá el permiso para aparecer en el lanzador de aplicaciones del dispositivo (**LAUNCHER**) y de ser ejecutada desde fuera de la aplicación (**MAIN**).

Nota: El fichero **AndroidManifest.xml** tiene más información de la que puedes ver en un primer momento. A las opciones que insertas directamente se le añaden otras opciones que proceden, por ejemplo, de bibliotecas importadas. Para ver el contenido completo debes usar el botón **Merged Manifest** que aparece cuando estás viendo el fichero. Una de las cosas que podrás ver en el fichero completo es la **API** con la que se compila la app y la mínima **API** en la que funcionará. Estas características no se pueden editar.

CARPETA JAVA

Contiene el código fuente en **Java**. Las clases están agrupadas en paquetes.

Esta carpeta debe contener, al menos, un paquete con el nombre indicado en el elemento **package** del elemento **manifest** del fichero **AndroidManifest.xml**. (en el ejemplo anterior **com.bol01.ejercicio01**). En ese paquete se encontrará la actividad principal que es una clase que toma el nombre **MainActivity** por defecto. Esta clase será el punto de entrada a la aplicación tal y como hemos establecido en el **AndroidManifest**.

Si al crear el proyecto no pedimos la creación automática de la actividad principal, probablemente este paquete no exista y deberíamos crearlo con:

CLICK-DERECHO SOBRE LA CARPETA JAVA/**New/Package**

Dentro del paquete deberá existir al menos una clase **Java** que defina la actividad principal:

CLICK-DERECHO SOBRE EL PAQUETE/**New/JAVA Class**

El nombre de la clase empezará por una letra mayúscula, coincidirá con el nombre del fichero que la contiene y debe aparecer en el atributo **android:name** del elemento **activity** que la define en el fichero **AndroidManifest.xml**.

Habitualmente esa clase la llamaremos **MainActivity** a no ser que se tengan buenos motivos para llamarla de otra manera.

La clase **MainActivity**, como actividad que es, tendrá que heredar de la superclase **AppCompatActivity** definida en el paquete **androidx.appcompat.app**. (o indirectamente heredando de alguna de sus subclases)

```
package com.bol01.ejercicio01;

import androidx.appcompat.app.AppCompatActivity;
...
public class MainActivity extends AppCompatActivity {
    ...
}
```

CARPETA RES

La carpeta **res/** contiene los recursos que utiliza nuestra aplicación agrupados por tipos en las carpetas **res/drawable/**, **res/layout/** y **res/values/**, entre otras.

Cuidado: Los nombres de recursos sólo admiten letras minúsculas, dígitos y el carácter guión bajo (_). Debes tenerlo presente porque si usas otros caracteres (por ejemplo, una mayúscula) tendrás problemas generalizados difíciles de detectar.

drawable

Son recursos "dibujables", imágenes. Si en esta carpeta incluimos un fichero con una imagen llamado **flag.png**, podremos acceder a ese recurso del siguiente modo:

- Desde un documento **XML** con:

"@drawable/flag"

- Desde código **Java** con:

R.drawable.flag

Observa que en ambos los casos se omite la extensión del fichero.

layouts

Son distribuciones (diseños) de pantalla. Un **layout** es una disposición de elementos de texto, gráficos y controles en la pantalla del dispositivo. Se almacenan en formato **XML**. Tendremos que crear al menos un **layout** para la actividad principal para que ésta sea visible y podemos llamarlo **activity_main.xml**.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Posteriormente, desde la clase **Java** que define la actividad, tendremos que asociarla a este **layout** con el método **setContentView()**.

```
setContentView(R.layout.activity_main);
```

values

En la carpeta **values** de los recursos almacenamos ficheros **XML** que agrupan recursos según su tipo. Así, será habitual tener un fichero **strings.xml** que contiene la declaración de todas las cadenas de texto que usemos en la aplicación.

```
<resources>
    <string name="app_name">Prueba1</string>
    <string name="hello">Hello World!!!</string>
</resources>
```

También se pueden escribir las cadenas de texto directamente, sin definirlos como recurso, pero definirlos aquí nos aporta dos ventajas:

- Cuando una cadena de texto se usa en varios puntos de nuestro programa, tenerla como recurso nos ayuda a mantenerla. Sólo se modifica en el fichero **strings.xml**.
- Al definir las cadenas de texto como recursos, **Android** gestionará por nosotros los cambios a distintos idiomas.

Una vez que una cadena está definida como recurso podemos hacer uso de ella así:

- Desde un documento **XML** con:

```
"@string/app_name"
```

- Desde código **Java** con:

```
R.string.app_name
```

Recursos alternativos

Los recursos de una aplicación **Android** se organizan en carpetas dentro de la carpeta **res/**. Estas carpetas no se pueden anidar, es decir, no podemos crear carpetas dentro de esas carpetas.

Lo que si podemos hacer es crear varias carpetas con recursos de un mismo tipo. Por ejemplo, podemos crear varias carpetas **drawable**, cada una de las cuales terminará en un sufijo distinto, por ejemplo: **drawable-en**, **drawable-es**, **drawable-land**, etc.

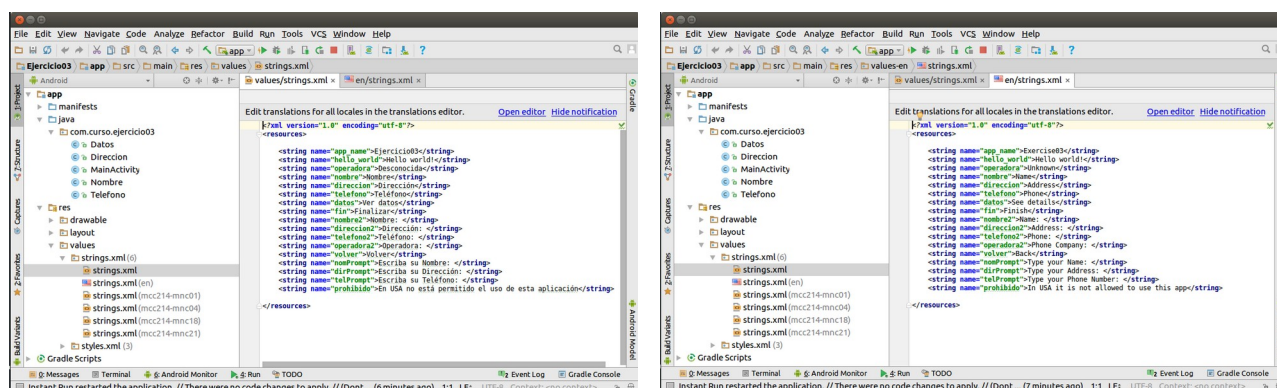
Nota: Desde el IDE **Android Studio** no se aprecia la existencia de estas carpetas. Solamente veremos varios recursos con el mismo nombre seguidos de los sufijos entre paréntesis.

Los sufijos no son aleatorios, hacen referencia a determinadas características (físicas o de configuración) del dispositivo en el que se ejecutan:

- **-en** el dispositivo está configurado para idioma inglés.
- **-es** el dispositivo está configurado para idioma castellano.
- **-land** el dispositivo se encuentra en posición horizontal

Cuando la aplicación hace referencia a un recurso, se buscará en la carpeta correspondiente a las características actuales del dispositivo y si no lo encuentra allí, entonces recurrirá al recurso que se encuentre en la carpeta por defecto (la que no tiene ningún sufijo, en nuestro ejemplo, **drawable**).

En el siguiente ejemplo se muestra el recurso **strings.xml** para castellano (**en**) y para inglés (**en**). Dependiendo del idioma seleccionado en el dispositivo, la aplicación usará uno u otro.



2.4.7 Ejecutar un proyecto en un terminal virtual

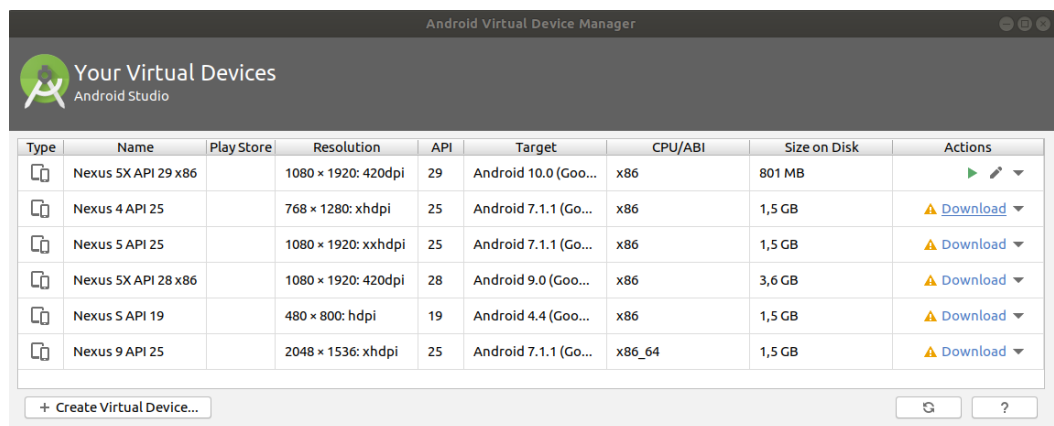
Desde el menú, seleccionamos:

RUN/RUN APP...

Aparecerá la pantalla para seleccionar el (AVD) dispositivo virtual en el que queremos ejecutar la app.

Si aún no tenemos ninguno, podrás crearlo desde:

TOOLS/AVD MANAGER

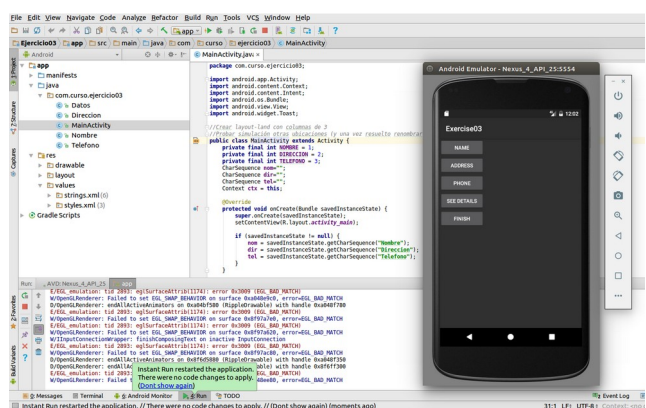


Puedes descargar uno de los dispositivos Google que aparecen directamente o seleccionar el que desees con **Create Virtual Device...**. En el asistente que sigue tendremos la posibilidad de indicar:

- **Tipo de dispositivo:** Smartphone, SmartTV, etc.
- **Marca y modelo:** Por ejemplo, Nexus 5.
- **API** del dispositivo.

Si estás interesado en un dispositivo concreto que no aparece en el listado podrás importarlo.

Quando hayas elegido uno de ellos comenzará a abrir el dispositivo virtual (esto puede tardar un poco) y ejecutará la aplicación sobre él. Se recomienda no cerrar el dispositivo para que las sucesivas ejecuciones sean más rápidas.

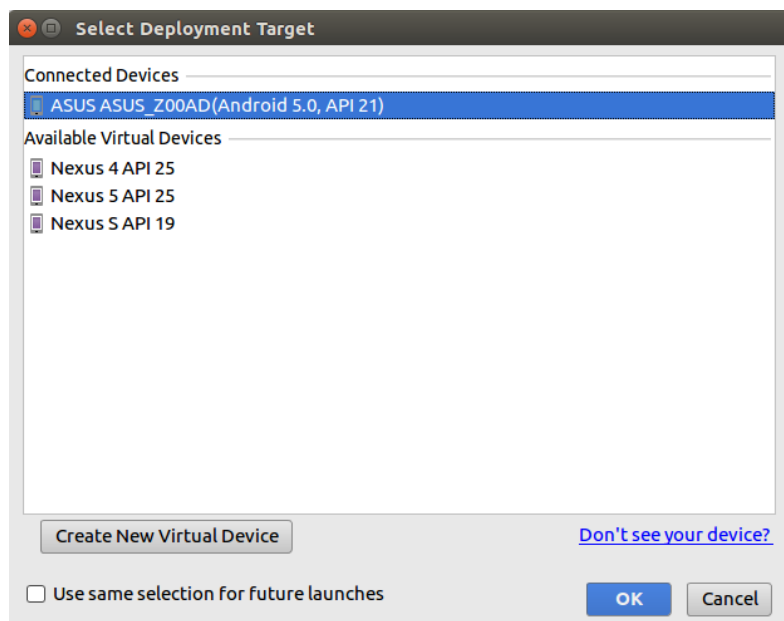


2.4.8 Ejecutar un proyecto en un terminal real

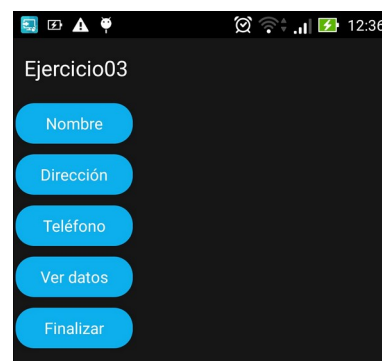
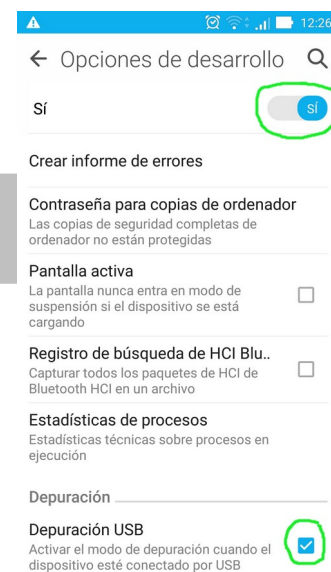
También podemos usar un smartphone real para ejecutar la app. Será necesario activar en el terminal las **Opciones de desarrollo** y el **modo depuración USB** y, naturalmente, conectar el smartphone al PC mediante un cable USB.

Importante: Una vez finalizadas las pruebas debes desactivar las opciones de desarrollo y el modo depuración **USB** en tu smartphone. Dejarlas activadas puede suponer **un agujero de seguridad en tu móvil**.

Ahora, al solicitar la ejecución de la app, entre los dispositivos conectados debe de aparecer tu smartphone:



Si lanzamos la ejecución, esta vez la app se ejecutará en el smartphone. Además quedará en el lanzador de aplicaciones de tu móvil para futuras ejecuciones.

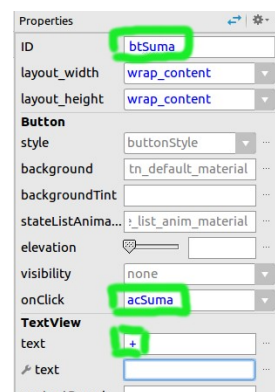


2.4.9 Ejemplo: Crear una app Calculadora

1. Crea un nuevo proyecto, bien desde la pantalla de bienvenida, bien desde **File/New/New Project**. Seguimos los pasos indicados en el apartado sobre creación de proyectos eligiendo “**EmptyActivity**” como modelo.
2. Modifica el **layout** asociado a la actividad principal. Si has seguido las instrucciones será el fichero **activity_main.xml** que se encuentra en **res/layout**.

- Arrastra los elementos desde la paleta usando **TextView** para las etiquetas no editables, **Button** para los botones y el **EditText** adecuado para los cuadros de texto. (Elijiendo los **EditText** en función del contenido que se espera, **Android** hará el control de los datos de entrada).
- Cambia la ubicación y tamaño de cada uno de los elementos incorporados arrastrando con el ratón.
- Modifica sus propiedades prestando especial atención a:

- **ID**: Es el identificador para referenciar el control. Por defecto tomará nombres como **TextView1**, **EditText2**, **Button3**. Se recomienda cambiar a un nombre que puedas asociar fácilmente con el elemento, por ejemplo el **EditText** que alojará al operando 1 podría llamarse **etOp1**, el botón para sumar podría llamarse **btSuma**, etc.
- **text**: Es la propiedad que contiene el rótulo que mostrará el elemento (en el caso de botones y etiquetas) o el texto inicial en el caso de los **EditText**.
- **textAlignment**: De forma gráfica podrás señalar la alineación del rótulo.
- **onClick**: Esta propiedad es fundamental para los botones. Aquí deberás poner el nombre del método que se ejecuta cuando el usuario pulse el botón. Se recomienda seguir una nomenclatura homogénea, por ejemplo para el botón **btSuma** se debería poner **onClick: acSuma** (de acción suma).



Nota: Todos los elementos que añadimos al **layout** serán referenciados por su **ID**. Las referencias desde **Java** serán de la forma:

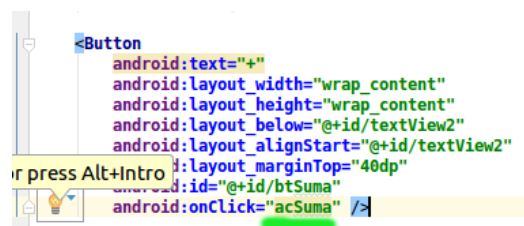
R.id.IdDelElto

Las referencias desde **XML** serán de la forma:

@+id/IdDelElto

- Crea los métodos asociados a los botones.

- Guarda el **layout** para que tengan efecto los cambios.



- Muestra el **layout** en formato **XML** (cambiando entre los botones **Design** y **Code**).
- Haz clic en la línea donde aparece el nombre que has asignado al método del botón (en el ejemplo **acSuma**).
- Usa la **bombillita** (💡) del margen para desplegar el menú de ayuda.
- Selecciona “**Create acSuma in MainActivity**”.

Nota: La combinación de teclas **Alt** - **Enter** hace aparecer las ayudas contextuales.

Al terminar este proceso, aparecerá un nuevo método en la actividad principal:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    public void acSuma(View view) {  
    }  
}
```

En ese método será donde debes incluir el código que se ejecutará al pulsar el botón.

EJEMPLO: CÓDIGO JAVA PARA EL MÉTODO DEL BOTÓN SUMA

```
public void acSuma(View view) {  
    EditText et1 = (EditText) findViewById(R.id.etOp1);  
    EditText et2 = (EditText) findViewById(R.id.etOp2);  
    TextView tvRes = (TextView) findViewById(R.id.tvResultado);  
    double op1 = Double.parseDouble(et1.getText().toString());  
    double op2 = Double.parseDouble(et2.getText().toString());  
    double resul = op1 + op2;  
    tvRes.setText(String.valueOf(resul));  
}
```

Donde:

- **et1**, **et2** y **tvRes** son objetos **Java** que hacen referencia al cuadro de texto del operando 1, el cuadro de texto del operando 2 y la etiqueta del resultado respectivamente. En las tres primeras líneas declaramos estas variables y les asignamos el objeto que se obtiene al buscar por su **id**.
- **op1**, **op2** y **resul** son variables de tipo **double** que alojan los datos numéricos. Los dos primeros los obtenemos de convertir en numéricos los **Strings** que hay en los cuadros de texto. El resultado lo calculamos sumando los operandos.
- En la última línea modificamos el rótulo de la etiqueta que aloja el resultado con el método **setText**. Se le asigna el **String** que resulta de convertir el número **resul**.

VARIOS IDIOMAS

Todos los rótulos que usa nuestra app pueden escribirse directamente en el código. Por ejemplo los rótulos de las etiquetas se escriben en el **XML** del layout:

```
android:text="Operando 1"
```

y los rótulos de mensajes o alertas se escriben en el código de la **Actividad**:

```
tvRes.setText("No se puede dividir por cero.");
```

Sin embargo, es buena práctica extraer todos los **Strings** al fichero **res/values/strings.xml**, donde se les asigna un nombre y usar una referencia a ese nombre:

```
/* fichero strings.xml */
<resources>
    <string name="stOp1">Operando 1</string>
    <string name="stDiv">No se puede dividir por cero.</string>
</resources>
```

Una vez hecho esto, en el código haremos uso de esta referencia, bien desde **XML**:

```
android:text="@string/stOp1"
```

bien desde la **Actividad Java**:

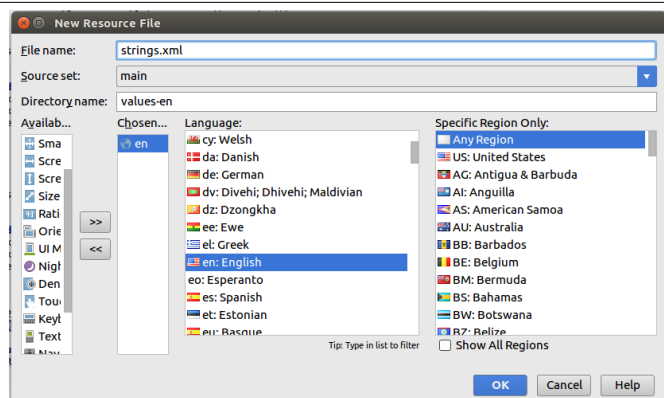
```
tvRes.setText(R.string.stDiv);
```

Nota: Si ya tenemos los textos escritos, podemos hacer la extracción automáticamente haciendo clic en el texto y usando la ayuda contextual (💡) con “**extract string resource**”.

El uso del fichero **strings.xml** para todos los rótulos de nuestra app, nos va a permitir traducir fácilmente la aplicación a otros idiomas que se aplicarán de forma automática cuando cambie la configuración del dispositivo.

Para traducir la app:

- Nombre del fichero: **strings.xml**.
- Directorio: **values** (durante el proceso se modificará con el sufijo correspondiente a nuestra elección).
- Elegimos el criterio diferenciador (en este caso **Locale**), inglés de cualquier región.



Aparecerá un nuevo fichero **strings.xml(en)** en el que podremos copiar y pegar los **Strings** del original para hacer la traducción. En la parte superior de la ventana de **Strings.xml** hay un enlace “**Open Editor**” que ayuda a hacer la traducción, incluso usando traductores de la web.

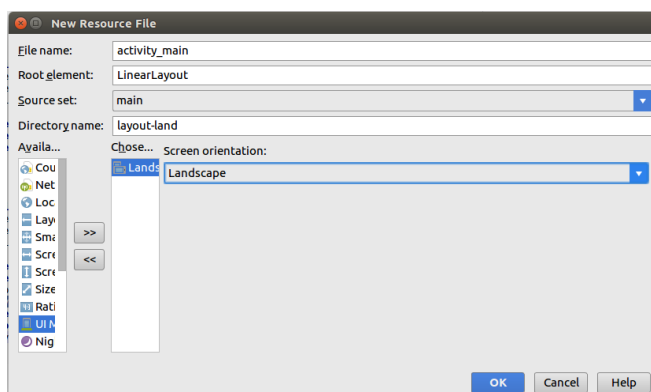
VARIOS LAYOUTS

De la misma forma que **Android** gestiona los rótulos en función del idioma activo en el terminal, puede automatizar la selección del **layout** adecuado a una configuración, por ejemplo, para diferenciar dos dispositivos de distinto tamaño o resolución. Y también para visualizar una pantalla distinta según la orientación del terminal (vertical u horizontal).

Vamos a crear un segundo layout para la actividad principal que se activará cuando el terminal se coloque en posición horizontal y tendrá, por tanto, una distribución más adecuada a esa orientación:

CLIC-DERECHO SOBRE **ACTIVITY_MAIN.XML/NEW/LAYOUT RESOURCE FILE**

- Nombre del fichero: **activity_main.xml**.
- Directorio: **layout** (durante el proceso se modificará con el sufijo correspondiente a nuestra elección).
- Elegimos el criterio diferenciador (en este caso **Orientation**), **landscape**.



Aparecerá un nuevo fichero **activity_main.xml(land)** en el que podremos copiar y pegar el código del **layout** original para después distribuirlo como nos interese.