

3.E. Librerías y paquetes.

Sitio: [VIRGEN DE LA PAZ](#)
Curso: Programación
Libro: 3.E. Librerías y paquetes.

Imprimido por: Cristian Esteban Gómez
Día: miércoles, 31 de mayo de 2023, 10:59

Tabla de contenidos

1. Librerías de objetos (paquetes).

- 1.1. Sentencia import.
- 1.2. Compilar y ejecutar clases con paquetes.
- 1.3. Jerarquía de paquetes.
- 1.4. Utilización de los paquetes.
- 1.5. Inclusión de una clase en un paquete.
- 1.6. Proceso de creación de un paquete.

2. Librerías Java.

1. Librerías de objetos (paquetes).

Conforme nuestros programas se van haciendo más grandes, el número de clases va creciendo. Meter todas las clases en único directorio no ayuda a que estén bien organizadas, lo mejor es hacer **grupos de clases**, de forma que todas las clases que estén relacionadas o traten sobre un mismo tema estén en el mismo grupo.

Un **paquete** de clases es una agrupación de clases que consideramos que están relacionadas entre sí o tratan de un tema común.



Imagen extraída de curso Programación del MECD.

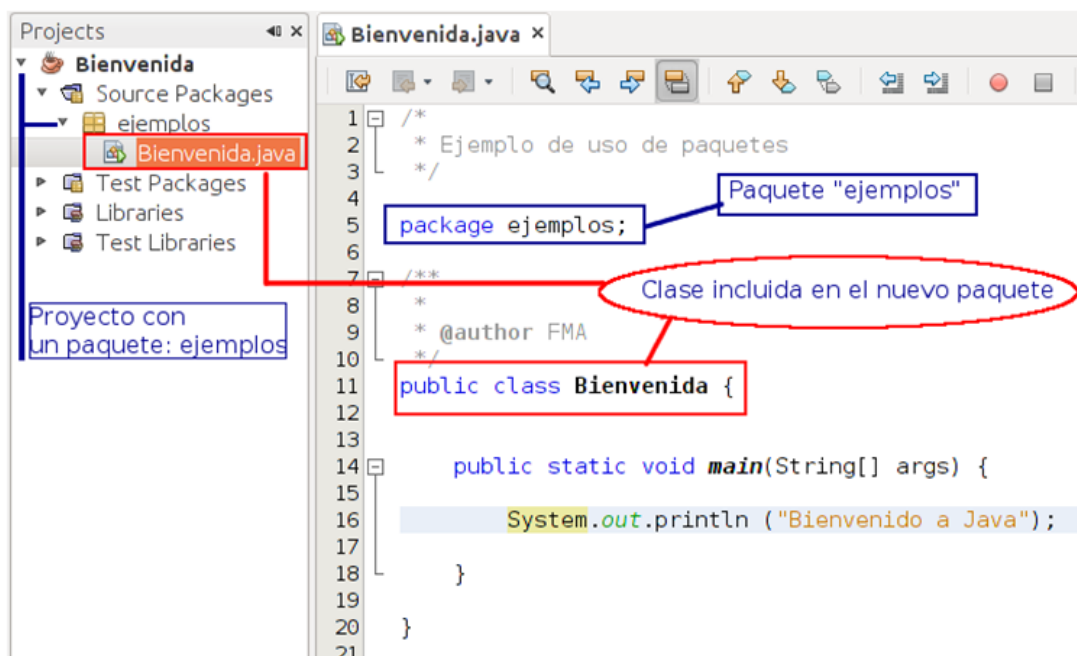
Las clases de un mismo paquete tienen un acceso privilegiado a los atributos y métodos de otras clases de dicho paquete. Es por ello por lo que se considera que los paquetes son también, en cierto modo, unidades de encapsulación y ocultación de información.

Java nos ayuda a organizar las clases en **paquetes**. En cada fichero .java que hagamos, al principio, podemos indicar a qué **paquete** pertenece la clase que hagamos en ese fichero.

Los paquetes se declaran utilizando la palabra clave **package** seguida del nombre del paquete. Para establecer el paquete al que pertenece una clase hay que poner una sentencia de declaración como la siguiente al principio de la clase:

```
package Nombre_de_Paquete;
```

Por ejemplo, si decidimos agrupar en un paquete "ejemplos" un programa llamado "Bienvenida", pondríamos en nuestro fichero **Bienvenida.java** lo siguiente:



El código es exactamente igual que como hemos venido haciendo hasta ahora, solamente hemos añadido la línea **"package ejemplos;"** al principio. En la imagen se muestra cómo aparecen los paquetes en el entorno integrado de Netbeans.

1.1. Sentencia import.

Cuando queremos utilizar una clase que está en un paquete distinto a la clase que estamos utilizando, se suele utilizar la sentencia `import`. Por ejemplo, si queremos utilizar la clase `Scanner` que está en el paquete `java.util` de la Biblioteca de Clases de Java, tendremos que utilizar esta sentencia:

```
import java.util.Scanner;
```

Se pueden importar todas las clases de un paquete, así:

```
import java.awt.*;
```

Esta sentencia debe aparecer al principio de la clase, justo después de la sentencia `package`, si ésta existiese.

También podemos utilizar la clase sin sentencia `import`, en cuyo caso cada vez que queramos usarla debemos indicar su ruta completa:

```
java.util.Scanner teclado = new java.util.Scanner (System.in);
```

Hasta aquí todo correcto. Sin embargo, al trabajar con paquetes, Java nos obliga a organizar los directorios, compilar y ejecutar de cierta forma para que todo funcione adecuadamente.

1.2. Compilar y ejecutar clases con paquetes.

Si hacemos que `Bienvenida.java` pertenezca al paquete `ejemplos`, debemos crear un subdirectorio "ejemplos" y meter dentro el archivo `Bienvenida.java`.

Por ejemplo, en Linux tendríamos esta estructura de directorios:

```
/<directorio_usuario>/Proyecto_Bienvenida/ejemplos/Bienvenida.java
```

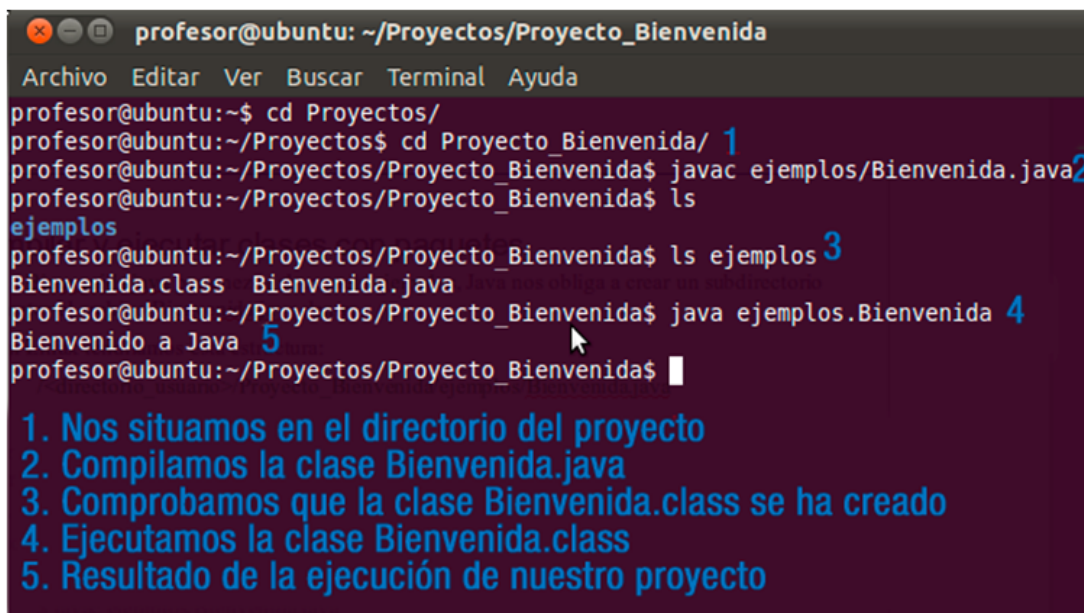
Debemos tener cuidado con las mayúsculas y las minúsculas, para evitar problemas, tenemos que poner el nombre en "package" exactamente igual que el nombre del subdirectorio.

Para **compilar** la clase `Bienvenida.java` que está en el paquete `ejemplos` debemos situarnos en el directorio padre del paquete y compilar desde ahí:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
```

```
$ javac ejemplos/Bienvenida.java
```

Si todo va bien, en el directorio `ejemplos` nos aparecerá la clase compilada `Bienvenida.class`.



```
profesor@ubuntu: ~/Proyectos/Proyecto_Bienvenida
Archivo Editar Ver Buscar Terminal Ayuda
profesor@ubuntu:~$ cd Proyectos/
profesor@ubuntu:~/Proyectos$ cd Proyecto_Bienvenida/ 1
profesor@ubuntu:~/Proyectos/Proyecto_Bienvenida$ javac ejemplos/Bienvenida.java 2
profesor@ubuntu:~/Proyectos/Proyecto_Bienvenida$ ls
ejemplos
profesor@ubuntu:~/Proyectos/Proyecto_Bienvenida$ ls ejemplos 3
Bienvenida.class Bienvenida.java
profesor@ubuntu:~/Proyectos/Proyecto_Bienvenida$ java ejemplos.Bienvenida 4
Bienvenido a Java 5
profesor@ubuntu:~/Proyectos/Proyecto_Bienvenida$
```

1. Nos situamos en el directorio del proyecto
2. Compilamos la clase `Bienvenida.java`
3. Comprobamos que la clase `Bienvenida.class` se ha creado
4. Ejecutamos la clase `Bienvenida.class`
5. Resultado de la ejecución de nuestro proyecto

Para ejecutar la clase compilada `Bienvenida.class` que está en el directorio `ejemplos`, debemos seguir situados en el directorio padre del paquete. El nombre completo de la clase es "paquete.clase", es decir "`ejemplos.Bienvenida`". Los pasos serían los siguientes:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
```

```
$ java ejemplos/Bienvenida
```

```
Bienvenido a Java
```

Si todo es correcto, debe salir el mensaje "Bienvenido a Java" por la pantalla.

Si el proyecto lo hemos creado desde un entorno integrado como Eclipse o Netbeans, la compilación y ejecución se realizará al ejecutar la opción `RunFile` (Ejecutar archivo) o hacer clic sobre el botón `Ejecutar`.

1.3. Jerarquía de paquetes.

Para organizar mejor las cosas, un **paquete**, en vez de clases, también puede contener otros paquetes. Es decir, podemos hacer **subpaquetes** de los paquetes y subpaquetes de los subpaquetes y así sucesivamente. Esto permite agrupar paquetes relacionados en un paquete más grande. Por ejemplo, si quiero dividir mis clases de ejemplos en ejemplos básicos y ejemplos avanzados, puedo poner más niveles de paquetes separando por puntos:

```
package ejemplos.basicos;
```

```
package ejemplos.avanzados;
```

A nivel de sistema operativo, tendríamos que crear los subdirectorios **basicos** y **avanzados** dentro del directorio **ejemplos**, y meter ahí las clases que correspondan.

Para compilar, en el directorio del proyecto habría que compilar poniendo todo el path hasta llegar a la clase. Es decir, el nombre de la clase va con todos los paquetes separados por puntos, por ejemplo **ejemplos.basicos.Bienvenida**.

La estructura de directorios en el sistema operativo cuando usamos subpaquetes sería:

```
/<directorio_usuario>/Proyecto_Bienvenida/ejemplos/basicos/HolaMundo.java
```

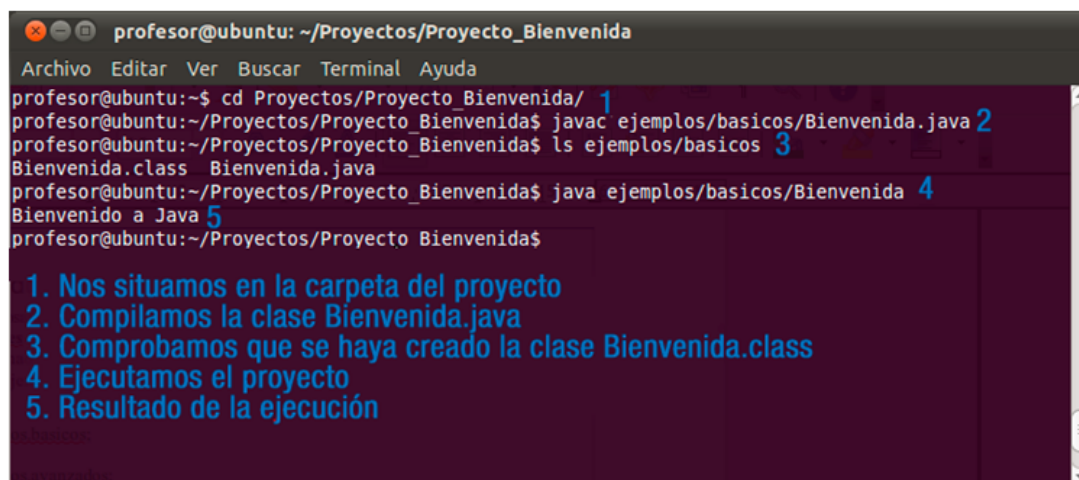
Y la compilación y ejecución sería:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
```

```
$ javac ejemplos/basicos/Bienvenida.java
```

```
$ java ejemplos/basicos/Bienvenida
```

```
Hola Mundo
```



La Biblioteca de Clases de Java se organiza haciendo uso de esta jerarquía de paquetes. Así por ejemplo, si quiero acceder a la clase **Date**, tendré que importarla indicando su ruta completa, o sea, **java.util.Date**, así:

```
import java.util.Date;
```

Los paquetes en Java pueden organizarse jerárquicamente de manera similar a lo que puedes encontrar en la estructura de carpetas en un dispositivo de almacenamiento, donde:

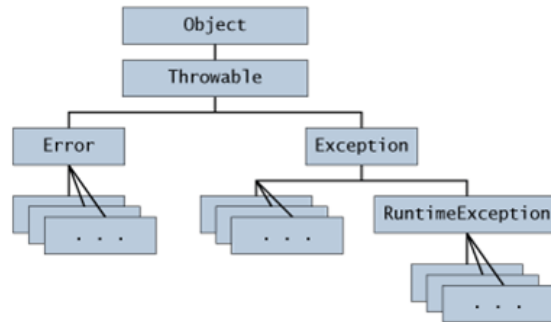
- Las clases serían como los archivos.
- Cada paquete sería como una carpeta que contiene archivos (clases).
- Cada paquete puede además contener otros paquetes (como las carpetas que contienen carpetas).
- Para poder hacer referencia a una clase dentro de una estructura de paquetes, habrá que indicar la trayectoria completa desde el paquete raíz de la jerarquía hasta el paquete en el que se encuentra la clase, indicando por último el nombre de la clase (como el path absoluto de un archivo).

La estructura de paquetes en Java permite organizar y clasificar las clases, evitando conflictos de nombres y facilitando la ubicación de una clase dentro de una estructura jerárquica.

Por otro lado, la organización en paquetes permite también el control de acceso a miembros de las clases desde otras clases que estén en el mismo paquete gracias a los modificadores de acceso (recuerda que uno de los modificadores que viste era precisamente el de paquete).

Las clases que forman parte de la jerarquía de clases de Java se encuentran organizadas en diversos paquetes.

Todas las clases proporcionadas por Java en sus bibliotecas son miembros de distintos paquetes y se encuentran organizadas jerárquicamente. Dentro de cada paquete habrá un conjunto de clases con algún tipo de relación entre ellas. Se dice que todo ese conjunto de paquetes forman la **API** de Java. Por ejemplo las clases básicas del lenguaje se encuentran en el paquete `java.lang`, las clases de entrada/salida las podrás encontrar en el paquete `java.io` y en el paquete `java.math` podrás observar algunas clases para trabajar con números grandes y de gran precisión.



Para saber más

Puedes echar un vistazo a toda la jerarquía de paquetes de la API de Java en la documentación oficial de Oracle (en inglés):

[Java Platform, Standard Edition 8 API Specification.](https://docs.oracle.com/javase/8/docs/api/)

1.4. Utilización de los paquetes.

Es posible acceder a cualquier clase de cualquier paquete (siempre que ese paquete esté disponible en nuestro sistema, obviamente) mediante la calificación completa de la clase dentro de la estructura jerárquica de paquete. Es decir indicando la trayectoria completa de paquetes desde el paquete raíz hasta la propia clase. Eso se puede hacer utilizando el operador **punto.**) para especificar cada subpaquete:

```
paquete_raiz_subpaquete1_subpaquete2_ ... _subpaquete_n_NombreClase
```

Por ejemplo:

```
java.lang.String.
```

En este caso se está haciendo referencia a la clase **String** que se encuentra dentro del paquete **java.lang**. Este paquete contiene las clases elementales para poder desarrollar una aplicación Java.

Otro ejemplo podría ser:

```
java.util.regex.Pattern.
```

En este otro caso se hace referencia a la clase **Pattern** ubicada en el paquete **java.util.regex**, que contiene clases para trabajar con expresiones regulares.

Dado que puede resultar bastante tedioso tener que escribir la trayectoria completa de una clase cada vez que se quiera utilizar, existe la posibilidad de indicar que se desea trabajar con las clases de uno o varios paquetes. De esa manera cuando se vaya a utilizar una clase que pertenezca a uno de esos paquetes no será necesario indicar toda su trayectoria. Para ello se utiliza la sentencia **import** (importar):

```
import paquete_raiz.subpaquete1.subpaquete2. ... .subpaquete_n.NombreClase;
```

De esta manera a partir de ese momento podrá utilizarse directamente **NombreClase** en lugar de toda su trayectoria completa.

Los ejemplos anteriores quedarían entonces:

```
import java.lang.String;
```

```
import java.util.regex.Pattern;
```

Si suponemos que vamos a utilizar varias clases de un mismo paquete, en lugar de hacer un **import** de cada una de ellas, podemos utilizar el **comodín** (símbolo **asterisco: "*"**) para indicar que queremos importar todas las clases de ese paquete y no sólo una determinada:

```
import java.lang.*;
```

```
import java.util.regex.*;
```

Si un paquete contiene subpaquetes, el comodín no importará las clases de los subpaquetes, tan solo las que haya en el paquete. La importación de las clases contenidas en los subpaquetes habrá que indicarla explícitamente. Por ejemplo:

```
import java.util.*;
```

```
import java.util.regex.*;
```

En este caso se importarán todas las clases del paquete **java.util** (clases **Date**, **Calendar**, **Timer**, etc.) y de su subpaquete **java.util.regex** (**Matcher** y **Pattern**), pero las de otros subpaquetes como **java.util.concurrent** o **java.util.jar**.

Por último tan solo indicar que en el caso del paquete **java.lang**, no es necesario realizar importación. El compilador, dada la importancia de este paquete, permite el uso de sus clases sin necesidad de indicar su trayectoria (es como si todo archivo Java incluyera en su primera línea la sentencia **import java.lang.***).

Autoevaluación

La sentencia **import** nos facilita las cosas a la hora de especificar las clases que queremos utilizar en nuestro archivo Java. Con el uso del comodín (asterisco) podemos importar todas las clases y subpaquetes que se encuentran en un determinado paquete a través de una sola sentencia **import**. ¿Verdadero o falso?

- ☐ Verdadero.
☐ Falso.

1.5. Inclusión de una clase en un paquete.

Al principio de cada archivo .java se puede indicar a qué paquete pertenece mediante la palabra reservada `package` seguida del nombre del paquete:

```
package nombre_paquete;
```

Por ejemplo:

```
package paqueteEjemplo;
```

```
class claseEjemplo {
```

```
...
```

```
}
```

La sentencia `package` debe ser incluida en cada archivo fuente de cada clase que quieras incluir ese paquete. Si en un archivo fuente hay definidas más de una clase, todas esas clases formarán parte del paquete indicado en la sentencia `package`.

Si al comienzo de un archivo Java no se incluyen ninguna sentencia `package`, el compilador considerará que las clases de ese archivo formarán parte del paquete por omisión (un paquete sin nombre asociado al proyecto).

Para evitar la ambigüedad, dentro de un mismo paquete no puede haber dos clases con el mismo nombre, aunque sí pueden existir clases con el mismo nombre si están en paquetes diferentes. El compilador será capaz de distinguir una clase de otra gracias a que pertenecen a paquetes distintos.

Como ya has visto en unidades anteriores, el nombre de un archivo fuente en Java se construye utilizando el nombre de la clase pública que contiene junto con la extensión .java, pudiendo haber únicamente una clase pública por cada archivo fuente. El nombre de la clase debía coincidir (en mayúsculas y minúsculas) exactamente con el nombre del archivo en el que se encontraba definida. Así, si por ejemplo tenías una clase `Punto` dentro de un archivo `Punto.java`, la compilación daría lugar a un archivo `Punto.class`.

En el caso de los paquetes, la correspondencia es a nivel de directorios o carpetas. Es decir, si la clase `Punto` se encuentra dentro del paquete `prog.figuras`, el archivo `Punto.java` debería encontrarse en la carpeta `prog\figuras`. Para que esto funcione correctamente el compilador ha de ser capaz de localizar todos los paquetes (tanto los estándar de Java como los definidos por otros programadores). Es decir, que el compilador debe tener conocimiento de dónde comienza la estructura de carpetas definida por los paquetes y en la cual se encuentran las clases. Para ello se utiliza el `ClassPath` cuyo funcionamiento habrás estudiado en las primeras unidades de este módulo. Se trata de una variable de entorno que contiene todas las rutas en las que comienzan las estructuras de directorios (distintas jerarquías posibles de paquetes) en las que están contenidas las clases.

1.6. Proceso de creación de un paquete.

Para crear un paquete en Java te recomendamos seguir los siguientes pasos:

1. **Poner un nombre al paquete.** Suele ser habitual utilizar el dominio de Internet de la empresa que ha creado el paquete. Por ejemplo, para el caso de **miempresa.com**, podría utilizarse un nombre de paquete **com.miempresa**.
2. **Crear una estructura jerárquica de carpetas equivalente a la estructura de subpaquetes.** La ruta de la raíz de esa estructura jerárquica deberá estar especificada en el `ClassPath` de Java.
3. **Especificar a qué paquete pertenecen la clase** (o clases) de el archivo .java mediante el uso de la sentencia `package` tal y como has visto en el apartado anterior.

Este proceso ya lo has debido de llevar a cabo en unidades anteriores al compilar y ejecutar clases con paquetes. Estos pasos simplemente son para que te sirvan como recordatorio del procedimiento que debes seguir a la hora de clasificar, jerarquizar y utilizar tus propias clases.

2. Librerías Java.

Cuando descargamos el entorno de compilación y ejecución de Java, obtenemos la API de Java. Como ya sabemos, se trata de un conjunto de bibliotecas que nos proporciona paquetes de clases útiles para nuestros programas.



Utilizar las clases y métodos de la Biblioteca de Java nos va ayudar a reducir el tiempo de desarrollo considerablemente, por lo que es importante que aprendamos a consultarla y conozcamos las clases más utilizadas.

Los paquetes más importantes que ofrece el lenguaje Java son:

- `java.io`. Contiene las clases que gestionan la entrada y salida, ya sea para manipular ficheros, leer o escribir en pantalla, en memoria, etc. Este paquete contiene por ejemplo la clase `BufferedReader` que se utiliza para la entrada por teclado.
- `java.lang`. Contiene las clases básicas del lenguaje. Este paquete no es necesario importarlo, ya que es importado automáticamente por el entorno de ejecución. En este paquete se encuentra la clase `Object`, que sirve como raíz para la jerarquía de clases de Java, o la clase `System` que ya hemos utilizado en algunos ejemplos y que representa al sistema en el que se está ejecutando la aplicación. También podemos encontrar en este paquete las clases que "envuelven" los tipos primitivos de datos. Lo que proporciona una serie de métodos para cada tipo de dato de utilidad, como por ejemplo las conversiones de datos.
- `java.util`. Biblioteca de clases de utilidad general para el programador. Este paquete contiene por ejemplo la clase `Scanner` utilizada para la entrada por teclado de diferentes tipos de datos, la clase `Date`, para el tratamiento de fechas, etc.
- `java.math`. Contiene herramientas para manipulaciones matemáticas.
- `java.awt`. Incluye las clases relacionadas con la construcción de interfaces de usuario, es decir, las que nos permiten construir ventanas, cajas de texto, botones, etc. Algunas de las clases que podemos encontrar en este paquete son `Button`, `TextField`, `Frame`, `Label`, etc.
- `java.swing`. Contiene otro conjunto de clases para la construcción de interfaces avanzadas de usuario. Los componentes que se engloban dentro de este paquete se denominan componentes Swing, y suponen una alternativa mucho más potente que AWT para construir interfaces de usuario.
- `java.net`. Conjunto de clases para la programación en la red local e Internet.
- `java.sql`. Contiene las clases necesarias para programar en Java el acceso a las bases de datos.
- `java.security`. Biblioteca de clases para implementar mecanismos de seguridad.

Como se puede comprobar Java ofrece una completa jerarquía de clases organizadas a través de paquetes.

Para saber más

En el siguiente enlace puedes acceder a la información oficial sobre la Biblioteca de Clases de Java (está en Inglés).

[Información oficial sobre la Biblioteca de Clases de Java.](https://docs.oracle.com/javase/10/docs/api/)