Paquetes en Java

Ocultando la implementación

Manuel J. Molino Milla Luis Molina Garzón

IES Virgen del Carmen

Departamento de Informática

10 de diciembre de 2014

Logo

Figura : Logo Java

Contenido

Introduccion

Contenido

Introduccion

Package

Contenido

Introduccion

Package

Modificadores de acceso en Java
- Amistoso
- Public
- Private
- Protected

Contenido

Introduccion

Package

Modificadores de acceso en Java
- Amistoso
- Public
- Private
- Protected

Archivos jar

Contenido

Introduccion

Package

Modificadores de acceso en Java
- Amistoso
- Public
- Private
- Protected

Archivos jar

Miscelánea

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.
- Cuado se crea un fichero fuente de *Java* se crea lo que se conoce como una *unidad de compilacion*

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.
- Cuado se crea un fichero fuente de *Java* se crea lo que se conoce como una *unidad de compilacion*
- Cada una de estas unidades tienen un nombre que acaba en *.java*

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.
- Cuado se crea un fichero fuente de *Java* se crea lo que se conoce como una *unidad de compilacion*
- Cada una de estas unidades tienen un nombre que acaba en *.java*
- Dentro de cada unidad de compilacion solo habra una clase publica.

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.
- Cuado se crea un fichero fuente de *Java* se crea lo que se conoce como una *unidad de compilacion*
- Cada una de estas unidades tienen un nombre que acaba en *.java*
- Dentro de cada unidad de compilacion solo habra una clase publica.
- El resto deben quedar ocultas para todo el exterior. Seran clases de apoyo.

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.
- Cuado se crea un fichero fuente de *Java* se crea lo que se conoce como una *unidad de compilacion*
- Cada una de estas unidades tienen un nombre que acaba en *.java*
- Dentro de cada unidad de compilacion solo habra una clase publica.
- El resto deben quedar ocultas para todo el exterior. Seran clases de apoyo.

Introducción

- Un paquete es lo que se obtiene cuando se usa la palabra clave *import*
- *import java.util.**
- Se importan todas las bibibliotecas desde el paquete *java.util*
- *import java.util.ArrayList*
- Ahora disponemos de la clase ArrayList y sobre todo de sus metodos.
- Cuado se crea un fichero fuente de *Java* se crea lo que se conoce como una *unidad de compilacion*
- Cada una de estas unidades tienen un nombre que acaba en *.java*
- Dentro de cada unidad de compilacion solo habra una clase publica.
- El resto deben quedar ocultas para todo el exterior. Seran clases de apoyo.

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*
- Se pueden tener bastante ficheros *.class* partiendo de un numero pequeño de fichero *.java*

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*
- Se pueden tener bastante ficheros *.class* partiendo de un numero pequeño de fichero *.java*
- Un programa en Java es un compendio de ficheros *.java* que pueden empaquetarse y comprimise en un fichero *.jar*

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*
- Se pueden tener bastante ficheros *.class* partiendo de un numero pequeño de fichero *.java*
- Un programa en Java es un compendio de ficheros *.java* que pueden empaquetarse y comprimise en un fichero *.jar*
- Para esto usaremos la herramienta *jar* de Java.

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*
- Se pueden tener bastante ficheros *.class* partiendo de un numero pequeño de fichero *.java*
- Un programa en Java es un compendio de ficheros *.java* que pueden empaquetarse y comprimise en un fichero *.jar*
- Para esto usaremos la herramienta *jar* de Java.
- El intérprete de Java es el responsable de encontrar, cargar e interpretar estos ficheros.

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*
- Se pueden tener bastante ficheros *.class* partiendo de un numero pequeño de fichero *.java*
- Un programa en Java es un compendio de ficheros *.java* que pueden empaquetarse y comprimise en un fichero *.jar*
- Para esto usaremos la herramienta *jar* de Java.
- El intérprete de Java es el responsable de encontrar, cargar e interpretar estos ficheros.

Introducción

- Cuando se compila un fichero *.java* se obtiene un fichero de salida que tiene exactamente el mismo nombre pero con extension *.class*
- Se pueden tener bastante ficheros *.class* partiendo de un numero pequeño de fichero *.java*
- Un programa en Java es un compendio de ficheros *.java* que pueden empaquetarse y comprimise en un fichero *.jar*
- Para esto usaremos la herramienta *jar* de Java.
- El intérprete de Java es el responsable de encontrar, cargar e interpretar estos ficheros.

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.
- Generalmente cada fichero tiene una clase publica.

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.
- Generalmente cada fichero tiene una clase publica.
- Si todos los ficheros estan relacionados formamos un paquete *package*

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.
- Generalmente cada fichero tiene una clase publica.
- Si todos los ficheros estan relacionados formamos un paquete *package*
- Para eso ponemos como primera linea de cada fichero que forman la biblioteca:

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.
- Generalmente cada fichero tiene una clase publica.
- Si todos los ficheros estan relacionados formamos un paquete *package*
- Para eso ponemos como primera linea de cada fichero que forman la biblioteca:
- *package nombre_del_paquete.*

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.
- Generalmente cada fichero tiene una clase publica.
- Si todos los ficheros estan relacionados formamos un paquete *package*
- Para eso ponemos como primera linea de cada fichero que forman la biblioteca:
- *package nombre_del_paquete*.
- Aquellas clases que quieran utilizar esta biblioteca indicaran en la cabecera:

Bibliotecas

- Una biblioteca tambien es un conjunto de estos fichero de clase.
- Generalmente cada fichero tiene una clase publica.
- Si todos los ficheros estan relacionados formamos un paquete *package*
- Para eso ponemos como primera linea de cada fichero que forman la biblioteca:
- *package nombre_del_paquete*.
- Aquellas clases que quieran utilizar esta biblioteca indicaran en la cabecera:
- *import nombre_del_paquete*

Ejemplos de package e import

Definicion del paquete

```
package mipaquete;
public class MiClase{
.....
```

Ejemplos de package e import

Definicion del paquete

```
package mipaquete;
public class MiClase{
.....
```

Uso sin import

```
mipaquete.MiClase m =  new mipaquete.MiClase();
.....
```

Uso con import

Ejemplos de package e import

Definicion del paquete

```
package mipaquete;
public class MiClase{
.....
```

Uso sin import

```
mipaquete.MiClase m =  new mipaquete.MiClase();
.....
```

Uso con import

```
import mipaquete;
MiClase m =  new MiClase();
.....
```

Package

- Un package es una agrupación de clases afines.

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.
- Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages.

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.
- Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages.
- Los packages delimitan el espacio de nombres (space name).

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.
- Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages.
- Los packages delimitan el espacio de nombres (space name).
- El nombre de una clase debe ser único dentro del package donde se define.

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.
- Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages.
- Los packages delimitan el espacio de nombres (space name).
- El nombre de una clase debe ser único dentro del package donde se define.
- Dos clases con el mismo nombre en dos packages distintos pueden coexistir e incluso pueden ser usadas en el mismo programa.

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.
- Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages.
- Los packages delimitan el espacio de nombres (space name).
- El nombre de una clase debe ser único dentro del package donde se define.
- Dos clases con el mismo nombre en dos packages distintos pueden coexistir e incluso pueden ser usadas en el mismo programa.

Package

- Un package es una agrupación de clases afines.
- Equivale al concepto de librería existente en otros lenguajes o sistemas.
- Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages.
- Los packages delimitan el espacio de nombres (space name).
- El nombre de una clase debe ser único dentro del package donde se define.
- Dos clases con el mismo nombre en dos packages distintos pueden coexistir e incluso pueden ser usadas en el mismo programa.

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.
- Supongamos que definimos una clase de nombre *miClase* que pertenece a un package de nombre *misPackages.Geometria.Base*

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.
- Supongamos que definimos una clase de nombre *miClase* que pertenece a un package de nombre *misPackages.Geometria.Base*
- Cuando la JVM vaya a cargar en memoria miClase buscará el módulo ejecutable (de nombre miClase.class) en un directorio en la ruta de acceso *misPackages/Geometria/Base*

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.
- Supongamos que definimos una clase de nombre *miClase* que pertenece a un package de nombre *misPackages.Geometria.Base*
- Cuando la JVM vaya a cargar en memoria miClase buscará el módulo ejecutable (de nombre miClase.class) en un directorio en la ruta de acceso *misPackages/Geometria/Base*
- Si una clase no pertenece a ningún package (no existe clausula package) se asume que pertenece a un package por defecto sin nombre, y la JVM buscará el archivo .class en el directorio actual.

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.
- Supongamos que definimos una clase de nombre *miClase* que pertenece a un package de nombre *misPackages.Geometria.Base*
- Cuando la JVM vaya a cargar en memoria miClase buscará el módulo ejecutable (de nombre miClase.class) en un directorio en la ruta de acceso *misPackages/Geometria/Base*
- Si una clase no pertenece a ningún package (no existe clausula package) se asume que pertenece a un package por defecto sin nombre, y la JVM buscará el archivo .class en el directorio actual.
- Si una clase no se declara public sólo puede ser usada por clases que pertenezcan al mismo package.

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.
- Supongamos que definimos una clase de nombre *miClase* que pertenece a un package de nombre *misPackages.Geometria.Base*
- Cuando la JVM vaya a cargar en memoria miClase buscará el módulo ejecutable (de nombre miClase.class) en un directorio en la ruta de acceso *misPackages/Geometria/Base*
- Si una clase no pertenece a ningún package (no existe clausula package) se asume que pertenece a un package por defecto sin nombre, y la JVM buscará el archivo .class en el directorio actual.
- Si una clase no se declara public sólo puede ser usada por clases que pertenezcan al mismo package.

Package

- Los packages además también tienen un significado físico que sirve para almacenar los módulos ejecutables (ficheros con extensión .class) en el sistema de archivos del ordenador.
- Supongamos que definimos una clase de nombre *miClase* que pertenece a un package de nombre *misPackages.Geometria.Base*
- Cuando la JVM vaya a cargar en memoria miClase buscará el módulo ejecutable (de nombre miClase.class) en un directorio en la ruta de acceso *misPackages/Geometria/Base*
- Si una clase no pertenece a ningún package (no existe clausula package) se asume que pertenece a un package por defecto sin nombre, y la JVM buscará el archivo .class en el directorio actual.
- Si una clase no se declara public sólo puede ser usada por clases que pertenezcan al mismo package.

Paquetes

Si compilamos el siguiente codigo con javac -d .

```
package libreria.utilidades;
public class Clase{
    ......
}
```

Paquetes

Si compilamos el siguiente codigo con javac -d .

```
package libreria.utilidades;
public class Clase{
    ......
}
```

Generamos automaticamente la siguiente estructura de directorios y archivos:

Paquetes

Si compilamos el siguiente codigo con javac -d .

```
package libreria.utilidades;
public class Clase{
    ......
}
```

Generamos automaticamente la siguiente estructura de directorios y archivos:

```
directorio_raiz
     |
     |--- Clase.java
     |
     |--- libreria
     |        |
     |        |---utilidades
     |        |       |
     |        |       |---Clase.class
```

Classpath

- Es una opción admitida en la línea de órdenes o mediante variable de entorno

Classpath

- Es una opción admitida en la línea de órdenes o mediante variable de entorno
- Indica a la Máquina Virtual de Java dónde buscar paquetes y clases definidas por el usuario a la hora de ejecutar programas.

Classpath

- Es una opción admitida en la línea de órdenes o mediante variable de entorno
- Indica a la Máquina Virtual de Java dónde buscar paquetes y clases definidas por el usuario a la hora de ejecutar programas.
- Cuando ejecutamos *java ClasePrincipal*. Java busca las clases que necesita en unos determinados directorios y ficheros .jar. Por defecto los buscará en los .jar propios de java y en el directorio en el que se esté ejecutando el comando java.

Classpath

- Es una opción admitida en la línea de órdenes o mediante variable de entorno
- Indica a la Máquina Virtual de Java dónde buscar paquetes y clases definidas por el usuario a la hora de ejecutar programas.
- Cuando ejecutamos *java ClasePrincipal*. Java busca las clases que necesita en unos determinados directorios y ficheros .jar. Por defecto los buscará en los .jar propios de java y en el directorio en el que se esté ejecutando el comando java.

Classpath

- Es una opción admitida en la línea de órdenes o mediante variable de entorno
- Indica a la Máquina Virtual de Java dónde buscar paquetes y clases definidas por el usuario a la hora de ejecutar programas.
- Cuando ejecutamos *java ClasePrincipal*. Java busca las clases que necesita en unos determinados directorios y ficheros .jar. Por defecto los buscará en los .jar propios de java y en el directorio en el que se esté ejecutando el comando java.

Ejemplo

```
DirectorioRaiz
     |
     |---DirectorioFuentes
     |          |
     |          |-ClasePrincipal.java
     |          |
     |---DirectorioLibrerias
     |          |
     |          |-ClaseAuxiliar.class
     |          |-Paquetes.jar
```

Ejemplo

```
DirectorioRaiz
      |
      |---DirectorioFuentes
      |          |
      |          |-ClasePrincipal.java
      |          |
      |---DirectorioLibrerias
      |          |
      |          |-ClaseAuxiliar.class
      |          |-Paquetes.jar
```

Compilamos javac -cp
../DirectorioLibrerias/:../DirectorioLibrerias/Paquetest.jar

Ejemplo

```
DirectorioRaiz
      |
      |---DirectorioFuentes
      |          |
      |          |-ClasePrincipal.java
      |          |
      |---DirectorioLibrerias
      |          |
      |          |-ClaseAuxiliar.class
      |          |-Paquetes.jar
```

Compilamos javac -cp
../DirectorioLibrerias/:../DirectorioLibrerias/Paquetest.jar
Se crea en el directorio DirectorioFuentes *ClasePrincipal.class*

Ejemplo

```
DirectorioRaiz
      |
      |---DirectorioFuentes
      |          |
      |          |-ClasePrincipal.java
      |          |
      |---DirectorioLibrerias
      |          |
      |          |-ClaseAuxiliar.class
      |          |-Paquetes.jar
```

Compilamos javac -cp
../DirectorioLibrerias/:../DirectorioLibrerias/Paquetest.jar
Se crea en el directorio DirectorioFuentes *ClasePrincipal.class*
Ejecutamos java -cp ../DirectorioLibrerias/:../DirectorioLibrerias/Paquetes.jar:.

Ejemplo

```
DirectorioRaiz
     |
     |---DirectorioFuentes
     |          |
     |          |-ClasePrincipal.java
     |          |
     |---DirectorioLibrerias
     |          |
     |          |-ClaseAuxiliar.class
     |          |-Paquetes.jar
```

Compilamos javac -cp
../DirectorioLibrerias/:../DirectorioLibrerias/Paquetest.jar
Se crea en el directorio DirectorioFuentes *ClasePrincipal.class*
Ejecutamos java -cp ../DirectorioLibrerias/:../DirectorioLibrerias/Paquetes.jar:.
En windows cambiamos los : por ;

Ejemplo

```
DirectorioRaiz
     |
     |---DirectorioFuentes
     |          |
     |          |-ClasePrincipal.java
     |          |
     |---DirectorioLibrerias
     |          |
     |          |-ClaseAuxiliar.class
     |          |-Paquetes.jar
```

Compilamos javac -cp ../DirectorioLibrerias/:../DirectorioLibrerias/Paquetest.jar
Se crea en el directorio DirectorioFuentes *ClasePrincipal.class*
Ejecutamos java -cp ../DirectorioLibrerias/:../DirectorioLibrerias/Paquetes.jar:.
En windows cambiamos los : por ;

Debemos recordar que tenemos la herramienta ant para automatizar tareas.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*
- Busca el fichero *.class* necesario.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*
- Busca el fichero *.class* necesario.
- Si no localiza dicho clase devuelve un error de clase no encontrada.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*
- Busca el fichero *.class* necesario.
- Si no localiza dicho clase devuelve un error de clase no encontrada.
- Para la creacion de paquetes unicos para su posterior uso se puede usar un dominio publico.

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*
- Busca el fichero *.class* necesario.
- Si no localiza dicho clase devuelve un error de clase no encontrada.
- Para la creacion de paquetes unicos para su posterior uso se puede usar un dominio publico.
- Ejemplo: package com.iesvirgendelcarmen.utilidades;

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*
- Busca el fichero *.class* necesario.
- Si no localiza dicho clase devuelve un error de clase no encontrada.
- Para la creacion de paquetes unicos para su posterior uso se puede usar un dominio publico.
- Ejemplo: package com.iesvirgendelcarmen.utilidades;

Creacion de paquetes unicos

- La agrupacion de ficheros en paquete en un subdirectorio asegura nombre de paquetes unicos.
- Tambien asegura una correcta localizacion de los mismos.
- El interprete de java encuetra primero la variable de entorno *CLASSPATH*
- Establecidad por el sistema operativo o por el programa de instalacion de Java.
- *CLASSPATH* contiene uno o mas directorios como raiz para la busqueda de ficheros *.class*
- Si encuentra un paquete en le codigo fuente llamado *package foo.bar.bax*
- Lo convierte (en caso de Ubuntu) en foo/bar/bax y lo concatena a la estructura de directorios de *CLASSPATH*
- Busca el fichero *.class* necesario.
- Si no localiza dicho clase devuelve un error de clase no encontrada.
- Para la creacion de paquetes unicos para su posterior uso se puede usar un dominio publico.
- Ejemplo: package com.iesvirgendelcarmen.utilidades;

Modificadores usados

Se colocan delante de cada declaración de cada miembro de la clase, bien sea un *atributo* o bien un *método*, controlando el acceso sólo para esa defínción en particular.

- Amistoso.

Modificadores usados

Se colocan delante de cada declaración de cada miembro de la clase, bien sea un *atributo* o bien un *método*, controlando el acceso sólo para esa defínción en particular.

- Amistoso.
- *public*

Modificadores usados

Se colocan delante de cada declaración de cada miembro de la clase, bien sea un *atributo* o bien un *método*, controlando el acceso sólo para esa defínción en particular.

- Amistoso.
- *public*
- *private*

Modificadores usados

Se colocan delante de cada declaración de cada miembro de la clase, bien sea un *atributo* o bien un *método*, controlando el acceso sólo para esa defínción en particular.

- Amistoso.
- *public*
- *private*
- *protected*

Modificadores usados

Se colocan delante de cada declaración de cada miembro de la clase, bien sea un *atributo* o bien un *método*, controlando el acceso sólo para esa defínción en particular.

- Amistoso.
- *public*
- *private*
- *protected*

Modificadores usados

Se colocan delante de cada declaración de cada miembro de la clase, bien sea un *atributo* o bien un *método*, controlando el acceso sólo para esa defínción en particular.

- Amistoso.
- *public*
- *private*
- *protected*

Modificador amistoso

- Cuando no se define ningun modificador se dice que tiene acceso *amistoso*.

Modificador amistoso

- Cuando no se define ningun modificador se dice que tiene acceso *amistoso*.
- Todas las clases del paquete actual tienen acceso al miembro amistoso.

Modificador amistoso

- Cuando no se define ningun modificador se dice que tiene acceso *amistoso*.
- Todas las clases del paquete actual tienen acceso al miembro amistoso.
- Pero las clases fuera del paquete no tienen acceso.

Modificador amistoso

- Cuando no se define ningun modificador se dice que tiene acceso *amistoso*.
- Todas las clases del paquete actual tienen acceso al miembro amistoso.
- Pero las clases fuera del paquete no tienen acceso.
- Este acceso da significado para agrupar juntas las clases de un paquete.

Modificador amistoso

- Cuando no se define ningun modificador se dice que tiene acceso *amistoso*.
- Todas las clases del paquete actual tienen acceso al miembro amistoso.
- Pero las clases fuera del paquete no tienen acceso.
- Este acceso da significado para agrupar juntas las clases de un paquete.

Modificador amistoso

- Cuando no se define ningun modificador se dice que tiene acceso *amistoso*.
- Todas las clases del paquete actual tienen acceso al miembro amistoso.
- Pero las clases fuera del paquete no tienen acceso.
- Este acceso da significado para agrupar juntas las clases de un paquete.

Modificador public

- Dicho atributo es accesible por todo el mundo.

Modificador public

- Dicho atributo es accesible por todo el mundo.
- En especial al programador cliente que hace uso de las bibliotecas.

Modificador public

- Dicho atributo es accesible por todo el mundo.
- En especial al programador cliente que hace uso de las bibliotecas.

Modificador public

- Dicho atributo es accesible por todo el mundo.
- En especial al programador cliente que hace uso de las bibliotecas.

Modificador private

- Nadie puede acceder a ese miembro excepto a traves de los metodos de esa clase.

Modificador private

- Nadie puede acceder a ese miembro excepto a traves de los metodos de esa clase.
- Otras clases del mismo paquete no pueden acceder a dichos miembros.

Modificador private

- Nadie puede acceder a ese miembro excepto a traves de los metodos de esa clase.
- Otras clases del mismo paquete no pueden acceder a dichos miembros.
- Se suelen usar para *métodos ayudantes*

Modificador private

- Nadie puede acceder a ese miembro excepto a traves de los metodos de esa clase.
- Otras clases del mismo paquete no pueden acceder a dichos miembros.
- Se suelen usar para *métodos ayudantes*

Modificador private

- Nadie puede acceder a ese miembro excepto a traves de los metodos de esa clase.
- Otras clases del mismo paquete no pueden acceder a dichos miembros.
- Se suelen usar para *métodos ayudantes*

Acceso privado

```
class Punto {
    private int x , y ;
    static private int numPuntos = 0;
    Punto ( int a , int b ) {
        x = a ; y = b;
        numPuntos ++ ;
    }
    int getX() {
        return x;
    }
    int getY() {
        return y;
    }
    static int cuantosPuntos() {
        return numPuntos;
    }
}
```

Acceso privado

```
class Punto {
    private int x , y ;
    static private int numPuntos = 0;
    Punto ( int a , int b ) {
        x = a ; y = b;
        numPuntos ++ ;
    }
    int getX() {
        return x;
    }
    int getY() {
        return y;
    }
    static int cuantosPuntos() {
        return numPuntos;
    }
}
```

Obtenemos un ERROR si hacemos:

Acceso privado

```
class Punto {
    private int x , y ;
    static private int numPuntos = 0;
    Punto ( int a , int b ) {
        x = a ; y = b;
        numPuntos ++ ;
    }
    int getX() {
        return x;
    }
    int getY() {
        return y;
    }
    static int cuantosPuntos() {
        return numPuntos;
    }
}
```

Obtenemos un ERROR si hacemos:
Punto p = new Punto(0,0);
p.x = 5;

Modificador protected

- Esta relacionado con el concepto de *herencia*

Modificador protected

- Esta relacionado con el concepto de *herencia*
- Proporciona acceso público para las clases derivadas.

Modificador protected

- Esta relacionado con el concepto de *herencia*
- Proporciona acceso público para las clases derivadas.
- Y acceso privado (prohibido) para el resto de clases.

Modificador protected

- Esta relacionado con el concepto de *herencia*
- Proporciona acceso público para las clases derivadas.
- Y acceso privado (prohibido) para el resto de clases.

Modificador protected

- Esta relacionado con el concepto de *herencia*
- Proporciona acceso público para las clases derivadas.
- Y acceso privado (prohibido) para el resto de clases.

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.
- En la mayoría de los casos los constructores deberían ser *public*

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.
- En la mayoría de los casos los constructores deberían ser *public*
- Podemos usar el modificador *private* para constructores cuando queremos asegurarnos que no se puedan crear objetos de esa clase.

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.
- En la mayoría de los casos los constructores deberían ser *public*
- Podemos usar el modificador *private* para constructores cuando queremos asegurarnos que no se puedan crear objetos de esa clase.
- En el caso de la clase *Math* donde sus miembros son static podemos crear un constructor private:

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.
- En la mayoría de los casos los constructores deberían ser *public*
- Podemos usar el modificador *private* para constructores cuando queremos asegurarnos que no se puedan crear objetos de esa clase.
- En el caso de la clase *Math* donde sus miembros son static podemos crear un constructor private:
- private Math() { }

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.
- En la mayoría de los casos los constructores deberían ser *public*
- Podemos usar el modificador *private* para constructores cuando queremos asegurarnos que no se puedan crear objetos de esa clase.
- En el caso de la clase *Math* donde sus miembros son static podemos crear un constructor private:
- private Math() { }

Uso de tipo de accesos

- El modificador *private* solo se aplica a miembros de clase (métodos y atributos)
- El modificador *public* se aplica a clases y atributos de la clase.
- El uso de modificadores *públic y private* en variables locales originará error.
- En la mayoría de los casos los constructores deberían ser *public*
- Podemos usar el modificador *private* para constructores cuando queremos asegurarnos que no se puedan crear objetos de esa clase.
- En el caso de la clase *Math* donde sus miembros son static podemos crear un constructor private:
- private Math() { }

Buenas prácticas

- Se aconseja declarar todos los atributos como *private*

Buenas prácticas

- Se aconseja declarar todos los atributos como *private*
- Cuando necesitemos consultar su valor o modificarlo, utilicemos los métodos *get* y *set*.

Buenas prácticas

- Se aconseja declarar todos los atributos como *private*
- Cuando necesitemos consultar su valor o modificarlo, utilicemos los métodos *get* y *set*.
- En caso de que el tipo de valor devuelto por el método sea un boolean, se utilizará *is* en vez de *get*.

Buenas prácticas

- Se aconseja declarar todos los atributos como *private*
- Cuando necesitemos consultar su valor o modificarlo, utilicemos los métodos *get* y *set*.
- En caso de que el tipo de valor devuelto por el método sea un boolean, se utilizará *is* en vez de *get*.

Buenas prácticas

- Se aconseja declarar todos los atributos como *private*
- Cuando necesitemos consultar su valor o modificarlo, utilicemos los métodos *get* y *set*.
- En caso de que el tipo de valor devuelto por el método sea un boolean, se utilizará *is* en vez de *get*.

Ejemplo 1

```
//Atributos private
private String nombre;
private boolean casado;

//Metodos get y set para estos atributos
public void setNombre(String nombre) {
    this.nombre = nombre;
}
public void setCasado(boolean casado) {
    this.casado = casado;
}
public String getNombre() {
    return nombre;
}
public boolean isCasado() {
    return casado;
}
```

Ejemplo 2

```
public class Vehiculo {
    //Atributos con acceso private
    private String modelo;
    private int velocidad;
    private boolean arrancado;
    //El constructor siempre debe de ser public
    public Vehiculo(String modelo, int velocidad, boolean arrancado) {
        this.modelo = modelo;
        this.velocidad = velocidad;
        this.arrancado = arrancado;
    }
    //Atributos getter con acceso public
    public String getModelo() {
        return modelo;
    }
    public int getVelocidad() {
        return velocidad;
    }
    public boolean isArrancado() {
        return arrancado;
    }
}
```

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.
- La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados.

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.
- La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados.
- El intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.
- La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados.
- El intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.
- Ejemplo:

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.
- La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados.
- El intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.
- Ejemplo:
- java -jar aplic.jar

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.
- La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados.
- El intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.
- Ejemplo:
- java -jar aplic.jar

Archivos jar

- Los ficheros Jar *(Java ARchives)* permiten recopilar en un sólo fichero varios ficheros diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.
- La particularidad de los ficheros .jar es que no necesitan ser descomprimidos para ser usados.
- El intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.
- Ejemplo:
- java -jar aplic.jar

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF
  2. En el fichero *MANIFEST.MF* ponemos: Main-Class: Principal

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF
  2. En el fichero *MANIFEST.MF* ponemos: Main-Class: Principal
  3. Creamos el fichero: *jar cfm fich.jar META-INF/MANIFEST.MF *.class*

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF
  2. En el fichero *MANIFEST.MF* ponemos: Main-Class: Principal
  3. Creamos el fichero: *jar cfm fich.jar META-INF/MANIFEST.MF *.class*
  4. Podemos saber el contenido del fichero: *jar tf fich.jar*

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF
  2. En el fichero *MANIFEST.MF* ponemos: Main-Class: Principal
  3. Creamos el fichero: *jar cfm fich.jar META-INF/MANIFEST.MF *.class*
  4. Podemos saber el contenido del fichero: *jar tf fich.jar*
  5. Lo lanzamos con: *java -jar fich.jar*

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF
  2. En el fichero *MANIFEST.MF* ponemos: Main-Class: Principal
  3. Creamos el fichero: *jar cfm fich.jar META-INF/MANIFEST.MF *.class*
  4. Podemos saber el contenido del fichero: *jar tf fich.jar*
  5. Lo lanzamos con: *java -jar fich.jar*

Crear jar

- Si el fichero .jar contiene muchos ficheros .class, ¿cuál de todos se ejecuta?
- Sabemos que debe ejecutarse el que contiene el método main pero, ¿cómo sabe el intérprete de Java que clase de todas las del fichero contiene este método?
- La respuesta es que lo sabe porque se lo tenemos que decir durante la creación del fichero jar.
- Para crearlo debemos seguir los siguientes pasos:
  1. Crear un directorio META-INF (las mayúsculas son necesarias) y dentro un fichero MANIFEST.MF
  2. En el fichero *MANIFEST.MF* ponemos: Main-Class: Principal
  3. Creamos el fichero: *jar cfm fich.jar META-INF/MANIFEST.MF *.class*
  4. Podemos saber el contenido del fichero: *jar tf fich.jar*
  5. Lo lanzamos con: *java -jar fich.jar*

API java Fechas

Las nuevas clases que se aportan en *Java 8*:

Instant es, básicamente, un timestamp numérico. Es una clase útil para realizar logs y usar para frameworks de persistencia.

API java Fechas

Las nuevas clases que se aportan en *Java 8*:

- **Instant** es, básicamente, un timestamp numérico. Es una clase útil para realizar logs y usar para frameworks de persistencia.
- **LocalDate** sirve para almacenar una fecha sin hora. Por ejemplo, un cumpleaños como "16-12-1979"

API java Fechas

Las nuevas clases que se aportan en *Java 8*:

- **Instant** es, básicamente, un timestamp numérico. Es una clase útil para realizar logs y usar para frameworks de persistencia.
- **LocalDate** sirve para almacenar una fecha sin hora. Por ejemplo, un cumpleaños como "16-12-1979"
- **LocalTime** sirve para almacenar una hora sin fecha. Por ejemplo, un horario de apertura a las "9:30".

API java Fechas

Las nuevas clases que se aportan en *Java 8*:

- **Instant** es, básicamente, un timestamp numérico. Es una clase útil para realizar logs y usar para frameworks de persistencia.
- **LocalDate** sirve para almacenar una fecha sin hora. Por ejemplo, un cumpleaños como "16-12-1979"
- **LocalTime** sirve para almacenar una hora sin fecha. Por ejemplo, un horario de apertura a las "9:30".
- **LocalDateTime** sirve para almacenar una fecha con hora. Por ejmplo, puede almacenar "1979-12-16T12:30"

API java Fechas

Las nuevas clases que se aportan en *Java 8*:

Instant es, básicamente, un timestamp numérico. Es una clase útil para realizar logs y usar para frameworks de persistencia.

LocalDate sirve para almacenar una fecha sin hora. Por ejemplo, un cumpleaños como "16-12-1979"

LocalTime sirve para almacenar una hora sin fecha. Por ejemplo, un horario de apertura a las "9:30".

LocalDateTime sirve para almacenar una fecha con hora. Por ejmplo, puede almacenar "1979-12-16T12:30"

ZonedDateTime almacena hora y fecha con información de uso horario.

Ejemplo

```
import java.time.*;
public class Fechas{
   public static void main(String[] arg){
      //imprimir la fecha actual
      LocalDate date = LocalDate.now();
      System.out.println(date);
      System.out.printf("%s-%s-%s%n", date.getDayOfMonth(),
            date.getMonthValue(),  date.getYear());
      //vamos a sumar 5 horas
      LocalTime time = LocalTime.now();
      System.out.println(time);
      LocalTime newTime;
      newTime = time.plusHours(5);
      System.out.println(newTime);
      //imprimimos fecha y hora
      LocalDateTime dateTime = LocalDateTime.now();
      System.out.println(dateTime);
   }
}
```

Preguntas