5. HERRAMIENTAS DE AUTOMATIZACIÓN ANT

Las herramientas de automatización empiezan a ser de utilidad cuando nuestros programas empiezan a crecer y a depender de librerías de terceros.
Algunas herramientas de automatización son ANT, MAVEN y GRADLE.

1. PROYECTO SALUDABASE

Disponemos de dos ficheros, **Hola.java** y **Saluda.java**, organizados en la siguiente estructura de directorios en la carpeta **saludaBase**:

Y con el contenido siguiente:

Fichero src/com/mgh/edd/**Saluda.java**

```
package com.mgh.edd;

public class Saluda {
    public static void saluda(String nombre) {
        System.out.println("Hola "+nombre);
    }
}
```

Fichero src/com/mgh/edd/**Hola.java**

```
package com.mgh.edd;

public class Hola {
    public static void main(String[] args) {
        String nombre=args.length>0?args[0]:"";
        Saluda.saluda(nombre);
    }
}
```

- El programa se compone de dos ficheros,
  - el principal (**Hola.java**), que contiene el método main
    - main recoge un argumento opcional de la línea de órdenes para personalizar el mensaje que mostrará en pantalla.
  - el fichero **Saluda.java**, que ofrece la funcionalidad saluda.

- Cuando trabajamos con **VARIOS FICHEROS FUENTE** en Java, podemos **organizar el código en carpetas**, que dan lugar a lo que se conoce como '**paquetes**' (package).
- Los **nombres de paquetes** se escriben en **minúscula**, y generalmente las empresas utilizan su **dominio** de Internet **a la inversa** como parte de estos.
  - Por ejemplo, si nuestro dominio fuese **mgh.com**, el **paquete** de nuestra aplicación podría ser **com.mgh.edd**.
- Esto tiene dos **IMPLICACIONES** en los ficheros fuente:
  - Deben indicar que **pertenecen al paquete** : package **com.mgh.edd**

    - Debe existir **correlación entre el nombre del paquete y la estructura de carpetas** a que se corresponde: **com/mgh/edd** con **com.mgh.edd**.
- Observa que hemos ubicado el directorio **com** dentro de otro directorio **src** (source) (directorio raíz para nuestro código fuente)

Para **COMPILAR** nos situaremos en la carpeta **raíz** del código **src**:

```
$ javac com/mgh/edd/Saluda.java

$ javac com/mgh/edd/Hola.java
```

- Realmente solo compilando el fichero **Hola.java** hubiese sido suficiente, ya que el compilador detecta que se utiliza una funcionalidad implementada en otro fichero fuente del mismo paquete y compilará también este.
- Se genera en **com/mgh/edd** los dos ficheros en bytecode: **Saluda.class** y **Hola.class**.

Para **EJECUTAR** utilizamos:

```
$ java com.mgh.edd.Hola Jose

Hola Jose
```

- Hemos utilizado el nombre de la **clase** Hola precedida del nombre del **paquete**.
- Podemos pasarle cualquier **argumento** para que emita un saludo personalizado.

2. ANT

Apache Ant es una **biblioteca de Java** que nos permite automatizar el proceso de construcción de aplicaciones.
Aunque se utiliza principalmente en Java, también soporta otros lenguajes.
Inicialmente, formó parte del proyecto Apache Tomcat, pero en el año 2000 se lanzó como proyecto independiente.

A. INSTALACIÓN EN LINUX

- Apache Ant está disponible en los repositorios de todas las distribuciones GNU/Linux.
- Refrescamos la lista de paquetes con: **sudo apt update**
- Instalamos Ant: **sudo apt install ant**

B. EL FICHERO build.xml

- Todo proyecto en Ant **se basa en un fichero de construcción**, llamado **build.xml**
- Ubicado en la **raíz** del proyecto
- Contiene los **objetivos o target:** diferentes fases de construcción del proyecto.

```xml
<project name="saludaAnt">
    <target name="clean">
        <delete dir="classes" />
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="classes" />
        <javac includeantruntime="false" srcdir="src/com/mgh/edd" destdir="classes" />
    </target>

    <target name="run" depends="compile">
        <property name="arg0" value=""/>
        <java classpath="classes" classname="com.mgh.edd.Hola">
            <arg value="${arg0}"/>
        </java>

    </target>
</project>
```

El fichero XML contiene:
-un elemento raíz <**project**> con el atributo **name**=”saludaAnt”, y
-tres elementos **target:**

• **clean**: Se encarga de limpiar el proyecto.
  - Contiene un elemento **delete** con un atributo **dir**, con valor **classes**.
  - Se indica que en la fase de limpieza se **borrará la carpeta** llamada **classes**.

• **compile**: Se encarga de compilar nuestro proyecto.
  - Tiene el atributo **depends**=”clean”, lo que indica que la fase de compilación depende de la fase de limpieza. (para compilar nuestro proyecto necesitamos limpiarlo antes)
  - Tiene dos etiquetas:
    - **mkdir:**se crea una nueva carpeta llamada **classes**, en la que ubicaremos los archivos .class para así separarlos del código fuente;
    - **javac:** se indica que se invocará al **compilador**, proporcionándole los archivos situados en la carpeta srcdir (src/com/mgh/edd) y dejando los ficheros compilados en la carpeta classes.
  - Incluye el atributo **includeantruntime**="false" para no incluir las bibliotecas de Ant

• **run**: Se encarga ejecutar la aplicación.
  - Depende de la compilación (para ejecutar bytecode debemos compilarlo previamente).
  - Contiene la etiqueta <**java**>, que invoca a la máquina virtual de Java para lanzar la clase **com.mgh.edd.Hola**, donde se ubica nuestro método **main**.
  - Establece el atributo **classpath** al valor **classes** para indicar que el resto de recursos del programa estarán en la carpeta **classes**.
  - Dentro de esta etiqueta, hemos añadido <**arg**>, con valor **"${arg0}"**, para indicar que se pasará al programa principal el primer argumento.
  - Además, se ha añadido un valor por defecto que será la cadena vacía, **<property name="arg0" value=""/>.**

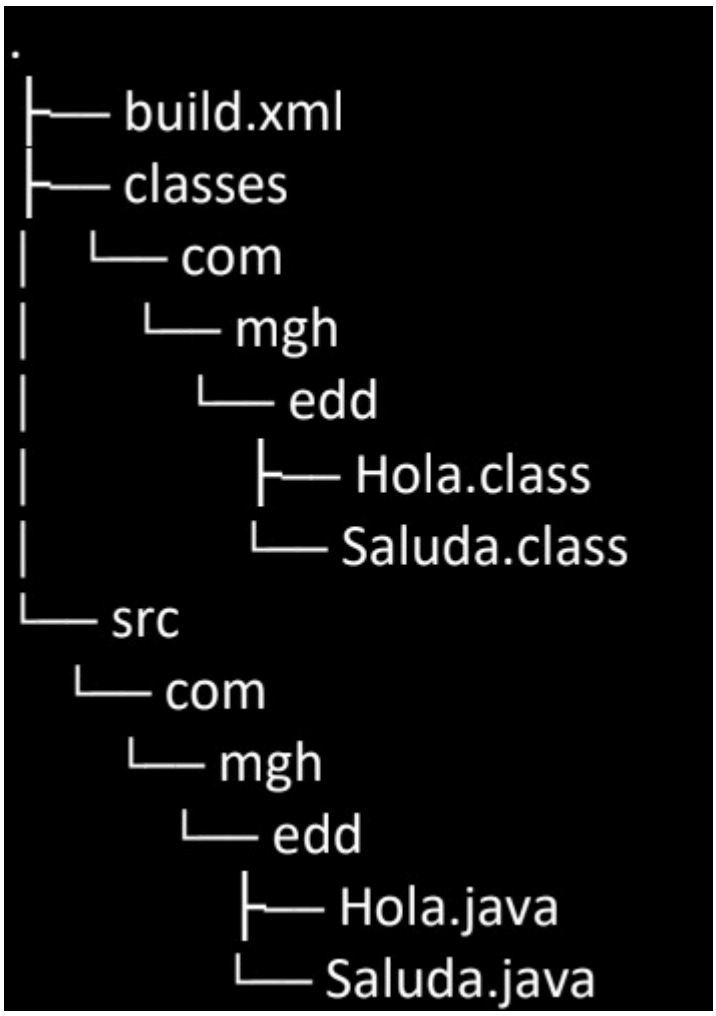
3. ANT EN EL TERMINAL

C. USO DE ANT Y EL FICHERO build.xml

• **COMPILAR** el proyecto: **ant compile**

```
$ ant compile
Buildfile: /home/dam/edd/scv/ud3-herramientas-
automatizacion/saludaAnt/build.xml


clean:
   [delete] Deleting directory /home/dam/edd/scv/ud3-herramientas-
automatizacion/saludaAnt/classes


compile:
    [mkdir] Created dir: /home/dam/edd/scv/ud3-herramientas-
automatizacion/saludaAnt/classes
    [javac] Compiling 2 source files to /home/dam/edd/scv/ud3-
herramientas-automatizacion/saludaAnt/classes


BUILD SUCCESSFUL
Total time: 1 second
```

- En primer lugar, se lanza la tarea **clean**.
  - Si es la primera vez , esta tarea no hace nada, ya que no tenemos nada que limpiar.
  - Si ya existiese la carpeta **classes** de otras compilaciones, borraría esta carpeta.
- En segundo lugar, se **compila**
  - con lo que se creará la carpeta **classes**, y
  - luego se invoca al compilador de Java para generar los ficheros en bytecode (.**class**).
- Después de la compilación, la estructura de nuestro proyecto es:



- Se ha generado la carpeta **classes.**
- Dentro de esta carpeta, se ha replicado toda la estructura de carpetas correspondiente al paquete, pero que contiene los ficheros **.class**.

EJECUTAR el proyecto: ant run -Darg0=Jose

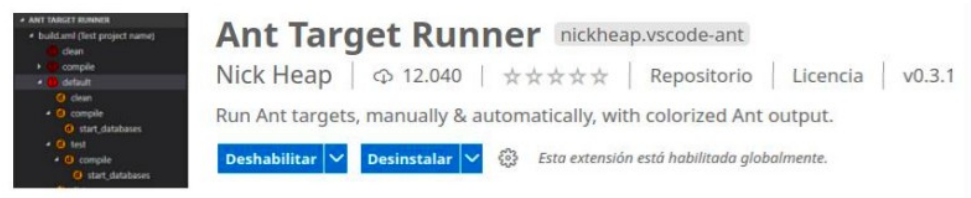
```
$ ant run -Darg0=Jose
Buildfile: /home/dam/edd/scv/ud3-herramientas-automatizacion/saludaAnt/build.xml


clean:
   [delete] Deleting directory /home/dam/edd/scv/ud3-herramientas-
automatizacion/saludaAnt/classes


compile:
    [mkdir] Created dir: /home/dam/edd/scv/ud3-herramientas-
automatizacion/saludaAnt/classes
    [javac] Compiling 2 source files to /home/dam/edd/scv/ud3-herramientas-
automatizacion/saludaAnt/classes


run:
     [java] Hola Jose


BUILD SUCCESSFUL
Total time: 1 second
```

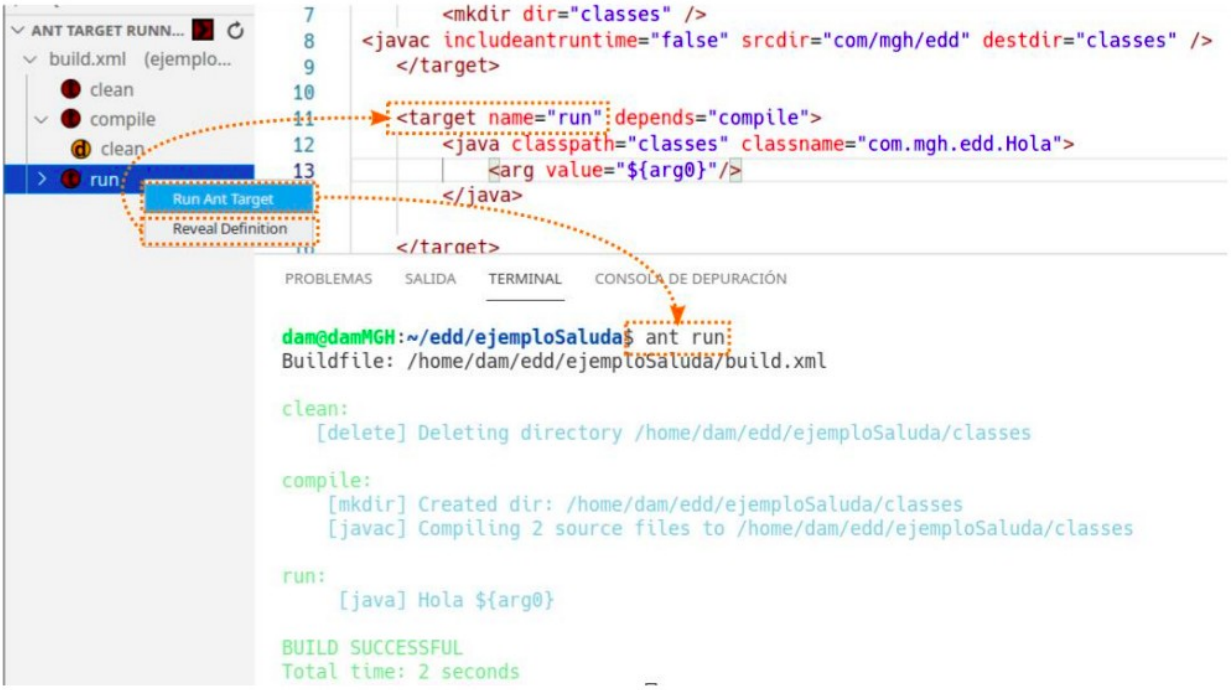
Observa que **ant run** debemos ofrecerle los **argumentos** mediante **-Darg0=Jose**.

- Añadimos la opción **-D,** seguida del nombre de argumento usado en el build.xml.
- Además, en este caso, la tarea **clean** sí ha borrado la carpeta classes; posteriormente realiza la compilación y finalmente, la ejecución.
- Si deseásemos **limpiar** todo el proyecto, ahora lanzaríamos el **ant clean**.

4. ANT EN VSCODE

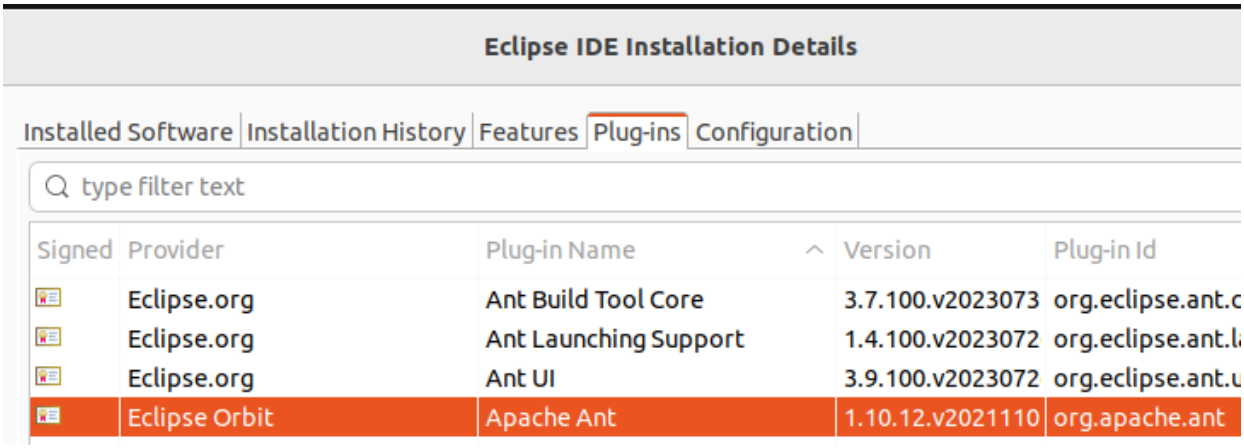
- La forma más sencilla es mediante la **terminal** integrada.
- Disponemos de la **extensión Ant Target Runner**
  - Añade una nueva **vista** en la barra lateral
  - Se nos muestran los diferentes targets y, de forma anidada, las dependencias entre estos.
  - Si pulsamos el botón derecho del ratón sobre un target de esta vista, aparece la posibilidad de ejecutarlo (**Run Ant Target**) o de ir a su definición (Reveal Definition).

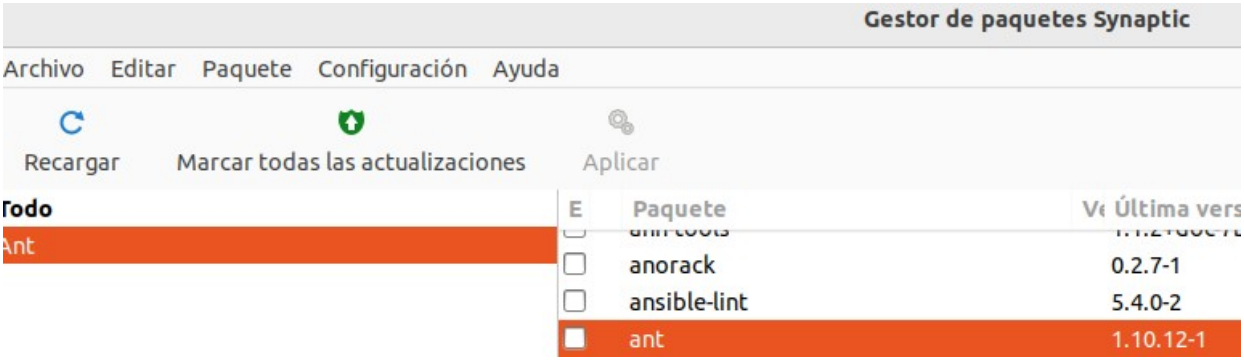




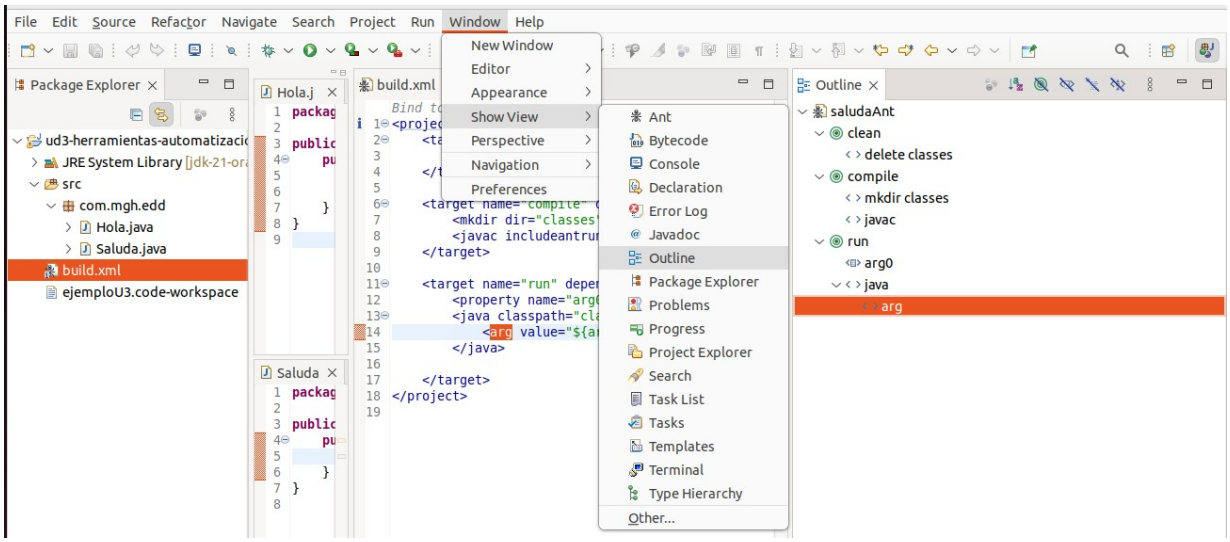
5. ANT EN ECLIPSE

- La **versión** de Ant **integrada** en la versión de Eclipse es 1.10.12 v2 y la que se encuentra por defecto en los **repositorios** de Ubuntu 22.04 (1.10.12-1).

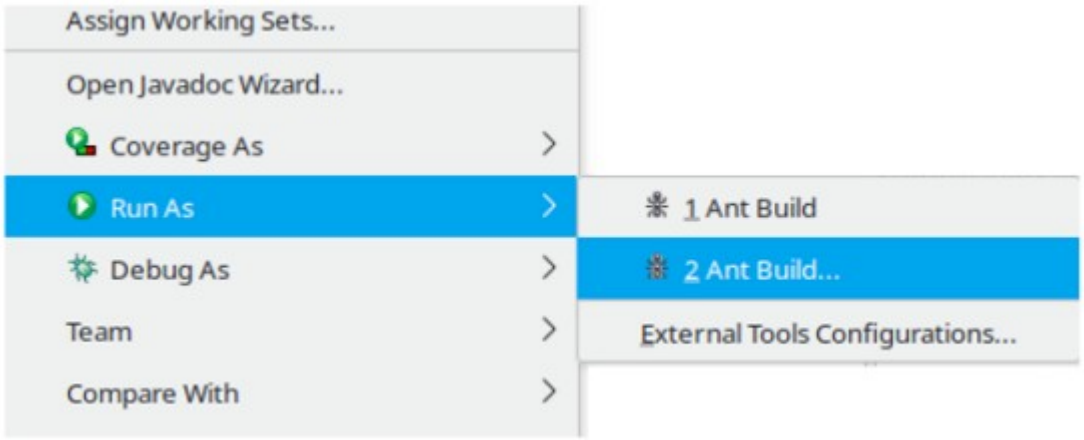




- Si deseamos **cambiar la versión de Ant** por la que tenemos en el equipo: la ventana de **preferencias**.
- Para **navegar** por un proyecto en **Ant**, usamos el **Explorador de proyectos.**
  - Si accedemos al fichero **build.xml**, en la ventana de **resumen** (**Outline**) se muestran los diferentes targets especificados en él.



- Desde el Explorador, **EJECUTAR** NUESTRO PROYECTO CON ANT.
  - clic con el botón d**erecho del ratón sobre el fichero build.xml**
  - **Run As,** y dentro de ella nos aparecen dos subopciones: **Ant Build** y **Ant Build...**



Ant Build:ejecuta directamente el proyecto con la última configuración utilizada
Ant Build... permite realizar la configuración antes de la ejecución.

- Si escogemos **Ant Build...**, se podrá escoger tanto el target a lanzar como los argumentos a utilizar, entre muchas otras opciones.