T7.Distribución de Aplicaciones

Índice

Caso práctico

BK programación ha terminado de desarrollar la aplicación de Gestión Hotelera y necesitan gestionar la forma de empaquetar y distribuir la aplicación, para poder ser entregada e instalada por los clientes.

El equipo de BK programación no tiene claro cómo distribuir la aplicación, si crear un paquete ejecutable o utilizar algún software de libre distribución, para crear un instalador que ayude en la instalación en la máquina cliente.

Ada quiere tantear todas las posibilidades de distribución de la aplicación, así que encarga a Juan que cree un paquete con la aplicación y a María le pide que pruebe a crear instaladores de la aplicación, usando las herramientas de software libre que hay en el mercado.

Como la aplicación puede tener éxito comercial, Ada propone a Carlos y a Ana, que investiguen las diferentes posibilidades que hay para poder distribuir la aplicación a través de Internet.

Por último, Ada quiere que las aplicaciones desarrolladas por BK Programación sean de calidad, por lo que se propone que la aplicación, para poder ser distribuida, sea firmada digitalmente.

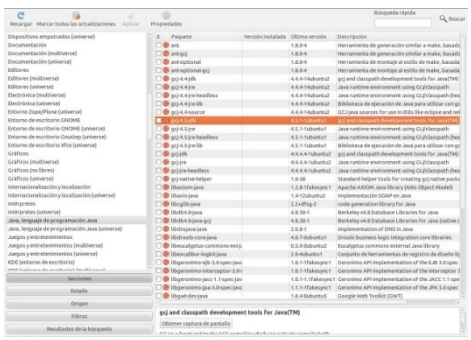
1. Definición y composición de una distribución. Sistema de gestión de paquetes.

Caso práctico

El equipo de desarrollo de BK Programación se reúne para decidir la forma de distribución de la aplicación de Gestión Hotelera. ¿Cómo hacerlo? ¿Qué patrones hay que seguir?

De igual forma, las **distribuciones** también hacen referencia a la colección de un conjunto variado de aplicaciones y paquetes de software junto con el sistema operativo, siendo muy común en las distribuciones de Linux, como Ubuntu.

Cuando nos encontramos con una distribución, normalmente tiene asociada una licencia, siendo la más habitual la licencia GPL u Open source. Otro tipo de distribución presente en el mercado, es la distribución binaria, donde nos encontramos con un instalador (fichero .exe en sistemas Microsoft Windows), que puede ser descargado desde Internet. Las distribuciones pueden ser oficiales si provienen de los autores originales o distribuciones 3rd party.



Un **sistema de gestión de paquetes** es una colección de herramientas que sirven para automatizar el proceso de

instalación, actualización, configuración y eliminación de paquetes de software. Normalmente, este término se usa para referirse a los gestores de paquetes en sistema Linux, que los usan como base principal de gestión de paquetes. Como ejemplo, en Ubuntu nos encontramos el Gestor de Paquetes apt o en su versión gráfica Synaptic.

En estos sistemas, el software se distribuye en forma de paquetes, que se encapsulan en un único fichero. Dentro del paquete nos encontramos: el software propiamente dicho, el nombre completo del paquete, una descripción de su funcionalidad, el número de versión, el distribuidor del software, la suma de verificación y una lista de otros paquetes requeridos para el correcto funcionamiento del software . Esta información se suele introducir normalmente en una base de datos de paquetes local.

En el siguiente enlace se da información de la instalación de paquetes en Ubuntu que ya debemos conocer: https://help.ubuntu.com/lts/serverguide/package-management.html.

Una distribución de software es un conjunto de programas específicos que se presenta compilado y configurado.

2. Instaladores. Pasos en la instalación. Asistente de instalación.

Caso práctico

Juan y María se van a encargar de preparar la aplicación para poder distribuirla. ¿Qué es necesario en una instalación? ¿Cómo puedo crear instaladores? ¿Qué puede modificar el usuario o usuaria final?

Un instalador es un programa especial que realiza las tareas de instalación de software de forma automática.

En la mayoría de los casos, un programa está formado por un conjunto de archivos. Normalmente, esos archivos necesitan ser copiados en determinadas carpetas o directorios, y en muchos casos, deben registrarse en el registro de Windows, si utilizamos ese sistema operativo.

Los instaladores van a realizar todas las operaciones anteriores de forma transparente al usuario. El uso del instalador suele ser muy sencillo. En la actualidad, los instaladores presentan al usuario una serie de formularios donde le van mostrando las indicaciones pertinentes, limitando al usuario a pequeña modificaciones o directamente a pulsar el botón siguiente. El instalador copiará los archivos de la aplicación a instalar en los directorios adecuados, registrará la aplicación, creará los menús y los accesos directos en el Escritorio.

Ejemplos de instaladores son: InstallAnywhere, Jexpress, InstallBuilder, Windows Installer, InstallShield, InstallAware, Wise Installation Studio, MSI Studio, NSIS, IzPack, etc.

Los pasos en la instalación son los siguientes:

1. **Verificación de la compatibilidad**: se debe comprobar que se cumplen los requisitos para la instalación, tanto hardware como software.
2. **Verificación de la integridad**: se verifica que el paquete de software es el original.
3. **Creación de los directorios requeridos**.
4. **Creación de los usuarios requeridos**: cada grupo de usuarios puede usar un determinado software.
5. **Copia, desempaquete y descompresión de los archivos desde el paquete de software**.
6. **Compilación y/o enlace con las bibliotecas requeridas**.
7. **Configuración**.
8. **Definición de las variables de entorno requeridas**.
9. **Registro de la aplicación** ante el autor de la aplicación.

Los asistentes de instalación son aplicaciones que ayudan al usuario a personalizar la instalación de software. Cuando utilizamos un instalador, normalmente tiene asociado un asistente de instalación. Cada paso de la instalación se muestra mediante un formulario con la información del paso se está dando. Los asistentes nos permiten elegir los directorios donde queremos que se instale la aplicación, el grupo de programas donde se integra la aplicación en el menú del escritorio, información sobre la licencia, registro de la aplicación, etc.

RESPONDE

1. ¿Qué paso se realiza en la instalación de un programa?:
   a) Creación de directorios requeridos.
   b) Verificación de la compatibilidad.
   c) Compilar el programa.
   d) Todas las anteriores

3. Paquetes autoinstalables.

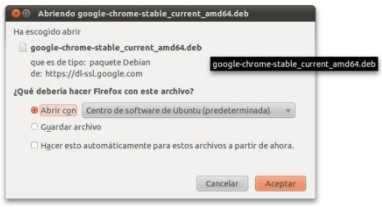
Caso práctico

Juan está probando la creación de un paquete autoinstalable para la aplicación. Él se propone crear un paquete para distribuir la aplicación en sistemas Linux tipo Ubuntu. Desea empaquetar la aplicación en un paquete Debian.

Cuando se está finalizando el ciclo de desarrollo de una aplicación llega el momento de decidir la mejor manera de distribuirla. No sólo tenemos que decidir la mejor manera de distribuirla, sino que también hay que tener en cuenta añadir características adicionales, parches y revisiones de la aplicación.

Cuando se decide distribuir una aplicación usando un paquete autoinstalable, lo que se está haciendo es empaquetar la aplicación en un único archivo, que contendrá todos los archivos y directorios que forman la aplicación.

Ese fichero será un fichero ejecutable en Windows (extensión exe), un paquete debian (fichero con extension deb) en distribuciones Linux basadas en Debian (Debian, Ubuntu, etc.), un paquete rpm en distribuciones estilo Red Hat (Red Hat, Suse, etc.).



Si nos encontramos en **Windows**, el paquete autoinstalable será una aplicación, que una vez lanzada por el usuario, realizará la descompresión de todos los archivos de la aplicación, creará las carpetas que la aplicación necesita, copiará los archivos a sus directorios de destino, añadirá y/o modificará entradas en el Registro de Windows, añadirá las entradas en el menú de aplicaciones y mostrará acceso directos en el Escritorio.

Durante todo este proceso, el usuario puede interaccionar eligiendo componentes a instalar, modificando los directorios de instalación, si se crean o no accesos directos, o puede optar por las opciones por defecto.

Si estamos distribuyendo una aplicación para Ubuntu, por ejemplo, lo que se crea es un paquete deb. Este tipo de archivo contiene todos los archivos y directorios de la aplicación. Cuando el usuario quiere realizar la instalación del paquete, ejecuta el "Centro de software de Ubuntu", que nos irá mostrando las ventanas de instalación de la aplicación.

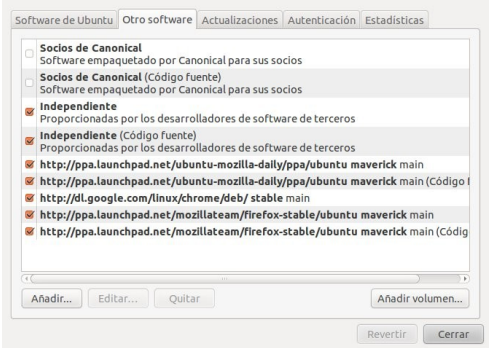
4. Herramientas para crear paquetes de instalación. Repositorios.

Caso práctico

Juan ha creado el paquete deb con la aplicación de Gestión Hotelera, su objetivo ahora es poder añadirlo a un repositorio. Mientras, María está analizando el software más conocido para la creación de paquetes de instalación.

En la actualidad existe un amplio abanico de herramientas de creación de paquetes de instalación. Las más extendidas para crear instaladores son las siguientes: InstallAnywhere, Jexpress, InstallBuilder, Windows Installer, InstallShield, InstallAware, WiseInstallation Studio, MSI Studio, NSIS, IzPack, Lift-Off, etc.

Si el proceso de instalación de software es lento o no termina de forma correcta, supone uno de los problemas más irritantes en un ordenador. Un instalador rápido y amigable debe ser parte esencial en cualquier producto software.



La instalación de una aplicación es el primer contacto que va a tener un usuario con la aplicación.

Las herramientas de instalación señaladas con anterioridad permiten a los programadores crear instaladores para Windows y algunas también para Linux. Si tomamos como ejemplo **NSIS** (Nullsoft Scriptable Install System) nos

encontramos con una herramienta open source que permite crear instaladores para nuestras aplicaciones en **Windows**. NSIS crea instaladores capaces de instalar, desinstalar, configurar el sistema, extraer archivos, etc. Este sistema está basado en **scripts**, con lo que el programador va a tener el control total en cualquier parte del instalador. El lenguaje script soporta variables, funciones, manipulación de cadenas, como un lenguaje de programación normal, pero diseñado para la creación de instalables.

Si trabajamos en **Linux**, necesitamos crear un paquete de instalación acorde con la distribución Linux donde queramos instalarlo. Si nos encontramos el Linux Ubuntu, el tipo de paquete será debian (.deb). El paquete de instalación deb, va a empaquetar todos los fichero que necesita nuestra aplicación y debe de estar configurado para indicar al "Centro de software de Ubuntu", donde se deben copiar esos archivos y que opciones de configuración se deben modificar.

En los sistemas Windows o Mac OS, los programas que queremos instalar se suelen buscar en Internet y se encuentran, en su mayoría, en forma de instaladores ejecutables. Ya cada vez es menos común la distribución de software en Cds y DVDs. En sistemas open source como Ubuntu GNU/Linux nos encontramos esta forma de distribución de software, pero la mayoría del software se encuentra empaquetado en ficheros .deb, (.rpm en Red Hat), que contienen programas y las bibliotecas que necesitan. Los repositorios son servidores que contienen conjuntos de paquetes. A esto servidores se accede con herramientas como apt o Synaptic.

Los repositorios van a centralizar todos los paquetes que se pueden instalar, ofreciendo un amplio abanico de software. En Ubuntu generalmente se querrá tener al menos los repositorios oficiales de Ubuntu pero es bastante común tener otros repositorios (de otros empaquetadores) activados.

5. Personalización de la instalación.

Caso práctico

María quiere crear un instalador para la aplicación lo más amigable e intuitivo posible. Ella cree necesario que la aplicación pueda ser instalada en sistemas Windows y Linux, por lo que está investigando el software disponible para la creación de instalaciones en Windows.

Una vez que un programador ha finalizado el desarrollo de una aplicación, debe decidir el mecanismo que va a utilizar para su distribución.

Si el producto desarrollado se va a instalar en sistemas Windows deberá crear instaladores para Windows (fichero ejecutables .exe). Si lo va a distribuir en Linux lo normal es que cree un paquete de instalación (paquete .deb en Ubuntu).

En ambos casos, el programador utilizará herramientas de generación de instaladores que le permitan personalizar su instalación.

Generalmente, cuando se crea un instalador para una aplicación, este instalador mostrará el logotipo de la aplicación, o de la empresa de desarrollo, tendrá un icono propio, unos colores y formato de ventana propios, formularios en los que se muestren los acuerdos de licencia, donde se pueden seleccionar el idioma de instalación, los directorios donde se desean copiar los archivos de la aplicación, etc.

Todos estos parámetros permiten crear un instalador propio para cada aplicación.

En la web de Debian tenemos un tutorial introductorio para llevar a cabo el proceso de empaquetamiento Debian (.deb): https://wiki.debian.org/es/IntroDebianPackaging.

En el siguiente enlace tenemos la Web oficial de la guía de empaquetado de Ubuntu: http://packaging.ubuntu.com/es/html/. Disponemos también del documento en formato pdf ubuntu-packaging-guide.pdf en el recurso compartido \\Profe2\Material\DI\Tema7.

En esta guía se explica que hay varias cosas que se deben hacer para comenzar a desarrollar en Ubuntu. En los artículos de esta guía se explica cómo configurar el sistema para poder trabajar con paquetes y cargarlos en la plataforma de alojamiento de Ubuntu, Launchpad. Entre otras cosas requiere: Instalación de utilidades de empaquetado específicas de Ubuntu, software de cifrado para que estos paquetes puedan ser verificado y certificar tu autoría, software de cifrado adicional para poder transferir archivos de forma segura, y la creación y configuración de una cuenta en Launchpad.

NSIS (Nullsoft Scriptable Install System) es un sistema profesional de código abierto para crear instaladores de Windows. Está diseñado para ser lo más pequeño y flexible posible y, por lo tanto, es muy adecuado para la distribución de Internet. El enlace siguiente procede de la página oficial de NullSoft y ofrece el manual de usuario: https://nsis.sourceforge.io/Docs/.

RESPONDE

2. ¿Qué herramienta no crea programas de instalación?:
   a) IzPack.
   b) NSIS.
   c) InstallShield.
   d) Synaptic.

5.1. Logotipos.

Caso práctico

> Ada propone que el programa de Gestión Hotelera tenga un logotipo que lo identifique como un programa de gestión de hoteles, que sea legible y fácilmente recordable. Este logotipo debe de estar presente en la instalación de la aplicación.

Dentro de los programas de instalación de software, un componente muy importante va a ser el logotipo.

El logotipo es un elemento gráfico que va a identificar a la empresa que ha desarrollado el programa o al propio programa.

Normalmente, los logotipos suele incluir símbolos que se asocian claramente con quienes representan.

El logotipo es el activo más importante de un servicio y producto y se utiliza como sello distintivo. Es por ello que a la hora de diseñar el logotipo de una aplicación, debemos tener presente algunos conceptos. Para que un logotipo resulte exitoso se debe seguir el principio fundamental de diseño de "menos es más", la simplicidad va a permitir que el logotipo sea:

- ✓ Legible.
- ✓ Escalable.
- ✓ Reproducible.
- ✓ Distinguible.
- ✓ Memorable.

En un instalador se puede insertar el logotipo arriba, abajo, a la derecha o la izquierda del instalador. El tamaño del logotipo vendrá en función del ancho/alto especificado, del ancho y alto del instalador y de la fuente utilizada en el instalador. Normalmente el logotipo será una imagen que insertaremos en el instalador, usando la herramienta de creación de instaladores.

Si tomamos como ejemplo NSIS, para añadir el logotipo en el instalador, se hace con el atributo AddBrandingImage. Este atributo presenta la siguiente sintaxis:

```
(left|right|top|bottom) (width|height)[u] [padding[u]]
```

Nota: http://nsis.sourceforge.net/Reference/AddBrandingImage

En el siguiente enlace se puede acceder a una página web donde se exponen las pautas de diseño a tener en cuenta en la creación de un logotipo: http://www.maestrosdelweb.com/anatomia-de-un-logotipo/.

5.2. Fondos.

Caso práctico

A la hora de crear el programa instalador de la aplicación de Gestión Hotelera, María va a utilizar como fondo del programa, imágenes y colores acordes con la imagen y colores corporativos de la empresa para la que diseñan la aplicación.

Cuando se diseña un instalador para una aplicación de interfaz gráfica, la interfaz del instalador deber adquirir y presentar la información de forma consecuente a la interfaz de la aplicación a la que sirve. En el caso de los fondos del instalador, estarán organizados en función del diseño estándar que se mantiene en todas las ventanas de la aplicación.

El fondo del instalador, por tanto, deberá implementar las mismas reglas de diseño para mantener la interacción en toda la aplicación.

Está comprobado que aquellos contenidos con color de fondo son interpretados por los usuarios como contenido poco importante; normalmente, se trata de contenido publicitario.

Cuando el color de fondo es blanco (o, sencillamente, no hay color de fondo) el usuario interpreta esa información como relevante y su nivel de atención aumenta considerablemente.

RESPONDE

3. ¿Qué tipo de archivo es el que se utiliza en Ubuntu para distribuir aplicaciones?:
   a) Ficheros ejecutables de extensión exe.
   b) Fichero ejecutable jar.
   c) Script ejecutables sh.
   d) Paquetes deb.

5.3. Botones.

Caso práctico

María, en su afán por personalizar el programa de instalación, también está considerando la posibilidad de cambiar el aspecto estándar de los botones del programa de instalación, para que sean más amigables y fáciles de utilizar.

Cuando se hace un instalador, los botones que se implementan son los de aceptar o rechazar el acuerdo de licencia. Los botones normalmente que aparecen son los siguiente, anterior y finalizar.

En el proceso de instalación gráfica de una aplicación, lo que se nos presentan son un conjunto de ventanas, en las que el usuario toma algunas decisiones. Los botones de cada una de las ventanas, son de aceptar o cancelar los valores ofrecidos por el instalador, y siguiente o anterior, para avanzar o retroceder en las ventanas de instalación.
Siempre hay que ser consistente con todos los elementos que forman parte de una aplicación, incluido su instalador.

Tenemos que ser consistentes con el diseño gráfico de los botones del instalador y del diseño gráfico de la aplicación a la que sirve.

Mantener la consistencia implica que el formato de las ventanas, los colores, iconos, imágenes, botones y demás componentes gráficos guarden el mismo formato, tanto en colores, fuentes y tamaños.

La forma más habitual de los botones gráficos es la rectangular, aunque a veces pueden tener las esquinas redondeadas. Otras formas cada vez más comunes son la circular, elíptica, trapezoidal…

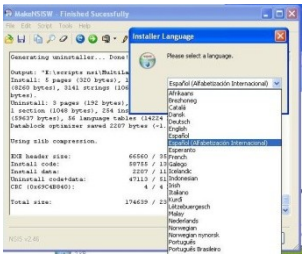
5.4. Idioma.

Caso práctico

Ada propone que la instalación se pueda realizar en varios idiomas, proponiendo que la instalación se pueda realizar en castellano, en los demás idiomas cooficiales de España y en inglés. Cree que con estas adaptaciones el programa se podrá instalar en todos los hoteles de la cadena, independientemente de donde se encuentren.

La mayoría de las aplicaciones que se distribuyen están a disposición de cualquier usuario en alguna página web, o páginas de descargas. Esta disposición global de las aplicaciones, implica que cualquier usuario de cualquier parte del mundo, pueda tener acceso a la aplicación y pueda descargarla e instalarla.

Cualquier aplicación que se desee distribuir a cualquier lugar, debe tener la interfaz implementada en inglés, pero es razonable traducir la interfaz a varios idiomas, y que el usuario final elija durante el proceso de instalación el idioma en el que desea que sea instalada.



El programador deberá tener en cuenta es su diseño, la posibilidad de que su aplicación se pueda distribuir en varios idiomas. Si esta posibilidad está presente, el programa instalador también deberá presentarse en diferentes idiomas y en él se podrá elegir el idioma de instalación final.

RESPONDE

4. Para crear un instalador personalizado en Windows deberemos:
   a) Crear un paquete de instalación deb.
   b) Crear un paquete ejecutable jar.
   c) Utilizar algún software específico como NSIS.

6. Generación de paquetes de instalación.

Caso práctico

Con la aplicación totalmente finalizada y probada, Juan se dispone a enseñar a Antonio a empaquetar la aplicación en paquetes JAR, para poder distribuirla y probarla en los ordenadores de los clientes.

Para generar paquetes de instalación de una aplicación, disponemos de varias alternativas:

- ✓ Utilizar entornos de desarrollo.
- ✓ Hacer uso de herramientas externas.
- ✓ Instalar en modo desatendido.

La primera alternativa es la utilización de las herramientas de generación de paquetes incorporadas en el Entorno de Desarrollo. Los entornos de desarrollo, suelen disponer de herramientas de generación de paquetes, pero no suelen incorporar herramientas que creen instaladores "amigables". En el caso del entorno de desarrollo **NetBeans**, podemos crear un paquete de instalación JAR de nuestra aplicación Java, sin embargo, la instalación y la ejecución del paquete, deberá realizarla el usuario.

El formato de archivo Java ™ Archive (JAR) permite agrupar varios archivos en un solo archivo. Normalmente, un archivo JAR contiene **los archivos CLASS** y los recursos auxiliares asociados con los applets y las aplicaciones.

El formato de archivo JAR proporciona muchos beneficios:

- ✓ **Seguridad**: se puede firmar digitalmente el contenido de un archivo JAR. Los usuarios que reconocen su firma pueden otorgar opcionalmente los privilegios de seguridad de su software que de otra forma no tendrían.
- ✓ **Disminución del tiempo de descarga**: si el applet se incluye en un archivo JAR, los archivos de clase del applet y los recursos asociados se pueden descargar a un navegador en una sola transacción HTTP sin la necesidad de abrir una nueva conexión para cada archivo.
- ✓ **Compresión**: el formato JAR permite comprimir los archivos para un almacenamiento eficiente.
- ✓ **Empaquetado para extensiones**: el marco de extensiones proporciona un medio por el cual se puede agregar funcionalidad a la plataforma central de Java, y el formato de archivo JAR define el empaquetado para extensiones. Al utilizar el formato de archivo JAR, también se puede convertir el software en extensiones.
- ✓ **Sellado de paquetes**: los paquetes almacenados en archivos JAR se pueden sellar opcionalmente para que el paquete pueda imponer la consistencia de la versión. Sellar un paquete dentro de un archivo JAR significa que todas las clases definidas en ese paquete deben encontrarse en el mismo archivo JAR.
- ✓ **Versión del paquete**: un archivo JAR puede contener datos sobre los archivos que contiene, como información del proveedor y la versión.
- ✓ **Portabilidad**: el mecanismo para manejar archivos JAR es una parte estándar de la API del núcleo de la plataforma Java

Documentación de Oracle en relación al empaquetado JAR de aplicaciones: https://docs.oracle.com/javase/tutorial/deployment/jar/.

Existen en el mercado herramientas que complementan a los entornos de desarrollo para la creación de paquetes de instalación. Esta herramientas, en el caso de aplicaciones Java, pueden crear el

paquetes JAR, o directamente partiendo del paquete JAR generado a partir del entorno de desarrollo, crear un nuevo proyecto de instalación, donde se cree un instalable o un autoinstalable, para la aplicación desarrollada.

En este punto, vamos a profundizar en la creación de paquetes de instalación para aplicaciones Java, usando el entorno de desarrollo NetBeans y con herramientas externas. Por último, veremos cómo poder realizar una aplicación desatendida, donde el usuario no interviene en la instalación de un paquete, ni elige la ubicación ni la configuración de la instalación.

RESPONDE

5. Los ficheros JAR:

a) Son paquetes ejecutables que contienen clases Java y otros recursos.
b) Son ficheros con código fuente Java.
c) Son ficheros que deben compilarse para poder ser ejecutados.

6.1. Entorno de desarrollo.

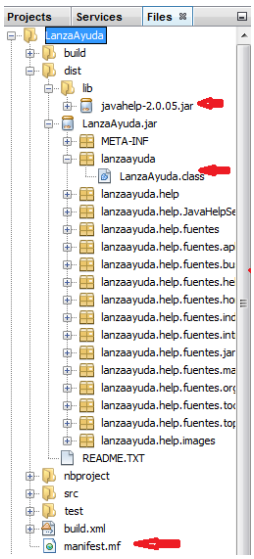
Caso práctico

Dado que, para el desarrollo de la aplicación de Gestión Hotelera, el equipo de desarrollo de BK Programación ha utilizado NetBeans, Juan va a empaquetar la aplicación en un fichero JAR utilizando las herramientas y opciones, suministradas por el propio entorno de desarrollo.

Para crear un paquete de instalación desde un entorno de desarrollo, lo primero a realizar, es la creación de la aplicación, la depuración de la aplicación y en su caso la realización de pruebas. Una vez que hemos finalizado todo el ciclo de vida de desarrollo de la aplicación, y si éste ha tenido éxito, procedemos a la fase de distribución de la aplicación.

En el caso del desarrollo de aplicaciones Java, el objetivo para distribuir una aplicación, es generar el fichero de empaquetado Jar.

En un entorno de desarrollo para Java como NetBeans, una vez depurado y probado el proyecto (Java with Ant), vamos a proceder a construirlo y crear el fichero JAR. Un fichero JAR es un archivo que contiene múltiples ficheros y carpetas. Los archivos JAR son similares a los ficheros comprimidos ZIP, pero los ficheros JAR pueden tener atributos adicionales que los hacen muy útiles para la distribución de aplicaciones Java. Cuando se realiza este paso, en el explorador del proyecto aparece una nueva carpeta de nombre **dist**. En esta carpeta se va a crear el fichero JAR. Si la aplicación requiere bibliotecas adicionales para poder ejecutarse, algo habitual en aplicaciones con interfaz gráfico, dentro de dist se crea una subcarpeta de nombre **lib**, que contiene las librerías necesarias para la ejecución (las que no estén incluidas en el JDK, Java Development Kit, este software provee herramientas de desarrollo para la creación de programas en Java).



La aplicación así generada, se puede ejecutar dentro del IDE y fuera de él. Para ejecutarla dentro de NetBeans, se selecciona el proyecto con el ratón en el explorador de proyectos, y se selecciona la opción ejecutar del menú contextual. Si la generación realizada en el paso anterior fue correcta, nos mostrará la ejecución de la aplicación desarrollada. Si queremos ejecutar la aplicación implementada fuera del IDE, navegamos por el explorador de nuestro sistema operativo (Windows, o Linux) y hacemos doble click sobre el fichero Jar que se encuentra en la carpeta dist de nuestro proyecto.

Para distribuir la aplicación a otros usuarios, en primer lugar, debemos comprobar que la aplicación funciona fuera del IDE. Para distribuir la aplicación, primero creamos un fichero zip que contenga el fichero de aplicación JAR y el directorio lib conteniendo las librerías que necesita la aplicación (aparte de las del JDK).Ya hemos comprobado que para poder crear el JAR de la aplicación es necesario que el proyecto tenga definida la clase principal. La clase principal se debe especificar en el interior del archivo manifest del fichero JAR con la entrada *Main-Class*. De la misma forma encontraremos una entrada *Class-Path* donde se especifiquen las librerías añadidas al proyecto. **Manifest** es una parte importante del fichero JAR que contiene información del fichero JAR que es necesaria para el lanzador java cuando queramos ejecutar la aplicación:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.9.7
Application-Name: LanzaAyuda
X-COMMENT: Main-Class will be added automatically by build
Caller-Allowable-Codebase: *
Permissions: all-permissions
Codebase: *
Application-Library-Allowable-Codebase: *
Class-Path: lib/javahelp-2.0.05.jar
Created-By: 1.8.0_181-b13 (Oracle Corporation)
Main-Class: lanzaayuda.LanzaAyuda
```

Cuando construimos un proyecto, el IDE construye el fichero JAR e incluye un manifest. Cuando configuramos la clase principal, debemos asegurarnos que la clase principal será designada en el manifest cuando se construya el proyecto. En las propiedades del proyecto menú Run podemos indicar la clase principal del proyecto. Recordamos que para crear el JAR en NetBeans solo hemos de hacer clic en el menú contextual del proyecto y pulsar ***Clean and Build Project*** o ***Build Project*** desde el menú principal Run.

Cuando se construye el proyecto además del directorio **dist** se ha creado el directorio **build**. Todas la fuentes son compiladas a ficheros .class que se encuentran en build.

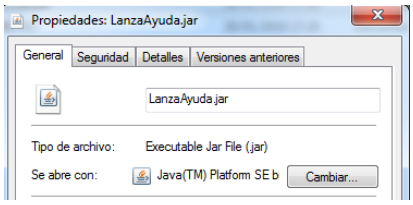
Una vez que se ha distribuido el fichero zip, la ejecución de la aplicación fuera del IDE se puede realizar de varias formas. La manera más rápida de ejecutar la aplicación, es descomprimir el fichero zip en cualquier carpeta seleccionada por el usuario final, y hacer doble clic sobre el fichero JAR. Además podemos invocar a la aplicación desde la línea de comandos o llamarla desde un fichero de script.

Si la aplicación no se ejecuta, puede que tengamos problemas con la asociación de fichero JAR con su aplicación, en nuestro sistema operativo. Esto puede deberse a dos razones:

- ✓ Que el tipo de fichero JAR no esté asociado a Java Runtime Environment (JRE, conjunto de utilidades que permite la ejecución de programas Java).
- ✓ Que el tipo de fichero JAR estén asociados a JRE, pero la opción -jar no está incluida en el comando que se pasa a JRE al hacer doble click en el icono.

Para añadir la asociación del fichero JAR a su aplicación en los sistemas Microsoft Windows:

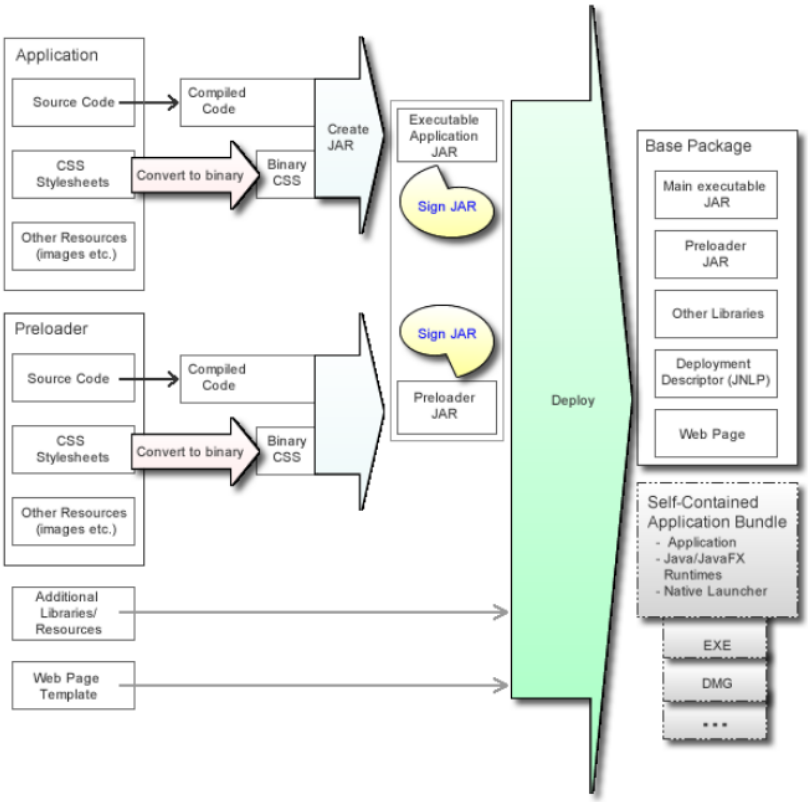
1. Nos aseguramos que tenemos una versión del JRE instalada en el sistema.
2. Si no está instalada podemos descargarla desde el sitio web de Java: http://www.oracle.com/technetwork/java/javase/downloads/index.html.
3. Si tenemos instalada JRE en nuestro sistema, lo que debemos es asociar los ficheros JAR con el JRE. Para ello en la opción "Abrir con", hay que asociar los ficheros JAR con "Java Platform SE Binary".



En los sistemas Unix y Linux: el procedimiento para cambiar la asociación de ficheros, depende del tipo de entorno de escritorio (Gnome o KDE) que se esté usando. En las preferencias de escritorio o en la documentación.

Para ampliar información de empaquetado y distribución de aplicaciones con NetBeans, visitaremos el enlace: https://docs.oracle.com/netbeans/nb82/netbeans/NBDAG/build_japps.htm#NBDAG510.

En cualquier caso el IDE sigue las directrices de la implantación de aplicaciones Java.
En el siguiente enlace de Oracle se documenta el despliegue de aplicaciones:
https://docs.oracle.com/javase/10/deploy/.



La imagen de arriba da una visión general de las tareas del empaquetado.

6.2. Herramientas externas.

Caso práctico

Como la idea de BK Programación es que la aplicación de Gestión Hotelera sea multiplaforma y pueda ser instalada en diferentes sistemas, María está utilizando diferentes programas para crear instaladores que se encuentran en el mercado. Como parte de un paquete JAR, se decanta por probar NSIS, que es una herramienta que permite crear instaladores para aplicaciones Java y es libre.

Hay muchas herramientas externas a los entornos de desarrollo que nos permiten crear paquetes de instalación para aplicaciones. Dentro de las aplicaciones Java, hay varias herramientas que nos permiten crear un instalador a partir del paquete JAR. Entre esas herramientas nos encontramos con IzPack y NSIS.

Izpack es una herramienta que nos permite crear instaladores a partir de aplicaciones Java. Funciona en cualquier sistema operativo que tenga instalada una Máquina Virtual Java (JVM). Con esta herramienta podemos personalizar el instalador de una aplicación Java. La aplicación se puede descargar para cualquier sistema operativo que tenga instalada un JRE o JDK de Java. Una vez instalada la herramienta, ya podemos implementar instaladores para aplicaciones Java que hayamos desarrollado.

Para crear instaladores con IzPack, debemos crear un documento XML, donde cada tag (etiqueta) tiene un significado para IzPack. Definidos todos los atributos que queramos en el fichero xml, ya podemos generar nuestro programa de instalación; para ello, sólo debemos ejecutar el siguiente comando:

```
"ruta de instalación IzPack" /bin/compile instalacion.xml -o install.jar
```

Donde el fichero instalacion.xml contiene las características de la instalación e install.jar en nuestra aplicación Java ya empaquetada.

Más información en: https://izpack.atlassian.net/wiki/spaces/IZPACK/overview.

A diferencia de IzPack, **NSIS** es una herramienta específica para entornos Windows, pero también es libre. NSIS nos proporciona un completo entorno para crea instaladores, tanto para aplicaciones Java, como aplicaciones desarrolladas en otros lenguajes. Doc en: https://nsis.sourceforge.io/Docs/.

Nullsoft Scriptable Install System funciona a través de un lenguaje propio de scripts. Para implementar un instalador, el programador escribe el script correspondiente, que una vez finalizado, será compilado por NSIS, creando un ejecutable como instalador.

NSIS proporciona un amplio abanico de script de ejemplo, que combinándolos y adaptándolos a nuestras necesidades, nos permiten crear instaladores completos, incluidos los desinstaladores.

6.3. Modo desatendido.

Caso práctico

Juan se centra en crear la instalación, de forma que no necesite la interacción del usuario, es decir, la aplicación se va a instalar utilizando una serie de parámetros que no van a ser modificados por el usuario final.

Una instalación en modo desatendido, es aquélla en la que el usuario no tiene que decidir el lugar de instalación de la aplicación, ni tampoco características tales como el idioma, variable de entorno, etc.

El uso de instalaciones desatendidas es útil cuando se deben instalar programas en gran cantidad de ordenadores, como cuando hay que instalar sistemas operativos.

Para automatizar el proceso de instalación, podemos crear un script que nos guarde las pulsaciones del ratón y del teclado y los datos que introducimos. De esta forma, al ejecutarse el instalador, reproduciría el patrón que nosotros hemos grabado previamente en el instalador.

Otra forma de realizar instalaciones desatendidas, es la utilización de los parámetros que los instaladores ponen a nuestra disposición. Por ejemplo, con el instalador NSIS podemos utilizar el parámetro **/S** para ejecutarlo en modo silencioso. **INNO Setup** es otro instalador para aplicaciones en Windows, que permite la instalación desatendida.

Cuando arrancamos un programa que ha sido empaquetado con este instalador, y le hacemos clic en el pequeño icono, podemos ver que debajo de la opción cerrar aparece la opción "AboutInnoSetup". Este es el claro ejemplo de un instalador del tipo INNO. Para poder instalar en modo silencioso un programa que ha sido empaquetado con el instalador de INNO, el parámetro que tenemos que usar es /SP- /VERYSILENT /NORESTART.

Los paquetes **MSI**, o también conocidos como Windows Installer suelen estar en combinación con InstallShield. En el caso del segundo lo que tenemos que intentar ver, es si podemos extraer de alguna manera los ficheros de ahí dentro, bien sea con un descompresor como Winrar, o bien haciendo una instalación administrativa con el parámetro /a. Esto último, lo que hará será extraernos todos los ficheros que están empaquetados en el instalador hacia una carpeta determinada. Una vez tengamos sus archivos fuera seguramente encontraremos un archivo MSI, que es el instalador de Windows. Este tipo de instalador lo podemos poner en modo silencioso con los comandos /qn (o/qb o /quiet) /norestart. También puede ser que directamente nos saque el contenido del programa. En este caso, lo podemos re-empaquetar con otro instalador que si que admita switches silenciosos o con un compresor con el modo silencioso marcado (como por ejemplo WinRar).

RESPONDE

6. En una instalación desatendida:
   a) El usuario decide la carpeta de instalación de la aplicación y todas las opciones de instalación.
   b) El instalador interactúa continuamente con el usuario final.
   c) La aplicación se instala de forma transparente al usuario.

7. Parámetros de la instalación.

Caso práctico

María quiere que el programa de instalación que está desarrollando, permita al usuario interactuar con él. Para ello, el instalador va a presentar al usuario todas las opciones de la instalación para que pueda personalizarlas.

Los parámetros de la instalación van a personalizar la aplicación, dando la opción al usuario que realiza la instalación de elegir entre diferentes alternativas y opciones, para ajustar la instalación de la aplicación a sus necesidades.

Dado que la mayoría de las aplicaciones actuales se distribuyen a través de repositorios que se encuentran alojados en sitios web, las aplicaciones son accesibles para cualquier usuario de cualquier parte del mundo.

Las aplicaciones que se desean que tenga mayor difusión, requieren estar disponibles en diferentes idiomas, de forma que el usuario que desea utilizar la aplicación pueda instalarlo en su propia lengua. Es por ello, que el primer paso de la instalación en esas aplicaciones es la selección del **idioma**.

Todas las aplicaciones se distribuyen bajo diferentes acuerdos de licencia, por tanto el usuario debe configurar la aceptación o rechazo del acuerdo de licencia, si el usuario no acepta los términos de la licencia, la instalación será abortada.

En las instalaciones actuales se suelen incluir aplicaciones o barras de herramientas para exploradores web adicionales, estas instalaciones son opcionales, eligiendo el usuario final su instalación o no.

Una vez que el usuario acepta los términos de la licencia, el siguiente parámetro a configurar, suele ser la ruta de instalación de los archivos de la aplicación, el instalador ofrece una ruta por defecto, que el usuario puede modificar si lo desea.

Elegido las carpetas donde se van a copiar los archivos necesarios de la aplicación, el siguiente paso que se parametriza es la creación de accesos directos en los menú de inicio del sistema operativo y de un acceso directo en el escritorio, estos parámetros deben ser configurables. En Windows se suele dar la opción al usuario de crear o no un acceso directo en el escritorio y un acceso directo en el Inicio Rápido.

El último parámetro que suele ser accesible por el usuario es la opción de ejecutar la aplicación una vez instalada.

8. Interacción con el usuario.

Caso práctico

En el instalador que María está creando, el usuario va a poder establecer las carpetas donde se instala el programa, la instalación o no de paquetes adicionales, la configuración del idioma, etc. La interfaz será amigable, para facilitar el proceso de instalación por parte del usuario.

Cuando se implementa una aplicación para ejecutar en interfaz gráfica (GUI), se basa en requerimientos no funcionales de usabilidad. En el caso de los instaladores de interfaz gráfica, también nos encontramos con una interfaz amigable.

En todo proceso de instalación de software en interfaces gráficas, se siguen una serie de pasos aceptados y estandarizados, que son los siguientes:

1. Ventana de selección de idioma.

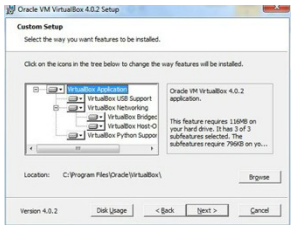
Lo primero que decide el usuario es el idioma en el que se le mostrarán los mensajes del proceso de instalación del programa.

2. Ventana de bienvenida.

Información de la versión de la aplicación, recomendaciones, etc.

3. Acuerdo de licencia. Aceptación de los términos de uso.

Se tienen que aceptar los términos de licencia del software que se quiere instalar. Si no es así, la instalación será abortada en este momento.



4. Aceptación o no aceptación de herramientas opcionales a instalar.

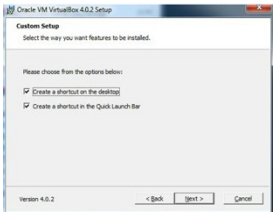
Se trata normalmente de complementos web para nuestro navegador. Podemos seleccionarlos o no, y esa elección no tendrá efecto sobre nuestra instalación.

5. Selección de la ubicación donde se guardan los archivos.

Normalmente, el programa nos propone por defecto para Windows la ruta C:\Archivos de programa pero el usuario debe poder cambiar esta ubicación y elegir otra distinta.

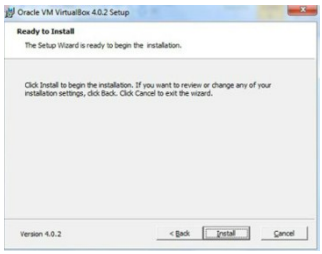
6. Selección de accesos directos.

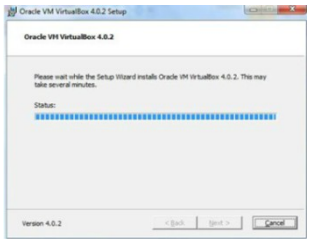
Opcional en todo caso. Al trabajar con sistemas operativos con interfaz gráfica, se debe elegir en que grupo de menú se instalan los accesos directos de la aplicación y si queremos tener un acceso directo desde el escritorio. Podemos configurar el nombre del grupo de aplicaciones y si esta tendrá un icono en el escritorio y el menú de inicio.



7. Proceso de instalación.

En este punto, comienzan a copiarse los archivos en el disco duro del ordenador del usuario, en la ubicación que él haya elegido. Creará las entradas necesarias en el registro (si de Windows se tratase), mostrará las entradas en los menús y los iconos en el escritorio. Se suelen mostrar ventanas con barra de progreso de la instalación para mantener informado al usuario.





8. Finalización.

El proceso de instalación termina en este punto y debe avisar al usuario de su terminación. Se suele usar una ventana con un resumen de la instalación al usuario y se suele dar la opción de ejecutar la aplicación por primera vez.

9. Creación de un instalador con NSIS.

1. Para descargar NSIS, lo haremos desde http://nsis.sourceforge.net/Download, o bien podéis obtener el archivo nsis-3.05-setup.exe (para Windows) o bien nsis-3.05-src.tar.bz2 del recurso compartido \\Profe2\Material\DI\Tema7\.
   Una vez instalado, disponemos del compilador que nos va a permitir generar instaladores a partir de los ficheros de **script.nsi** que desarrollemos.

2. Para crear un instalador con NSIS, debemos escribir un script NSIS. Un script NSIS es un fichero de texto plano con una sintaxis especial. Son script en los que cada línea es tratada como un comando. Dentro de la instalación de la aplicación, nos encontramos con una gran cantidad de scripts desarrollados como ejemplo. En función de nuestras necesidades, podremos editarlos y cambiarlos.

3. La extensión por defecto de los script es .nsi. También existen ficheros header (al estilo de los .h de C/C++) que tienen la extensión .nsh.
   Un script NSIS puede contener atributos del instalador, páginas, secciones y funciones.
   a. **Atributos del instalador**. Determinan el comportamiento y el aspecto del instalador. Estos atributos determinan los diferentes mensajes que se irán mostrando durante el proceso de instalación.
      Por ejemplo Name, es el atributo correspondiente al nombre de nuestra aplicación, InstallDir será el directorio elegido para instalar la aplicación etc.

   b. **Páginas**:
      Un instalador puede mostrar diferentes páginas al usuario, como por ejemplo la página de bienvenida, la de aceptación de licencia, la de selección del directorio de instalación, etc.:

```
; Pages
```

```
Page license
Page components
Page directory
Page instfiles
UninstPageUninstConfirm
UninstPageinstfiles
```

En caso de usar el UI moderno al incluir: !include "MUI.nsh" usaremos sus macros:

```
; mostramos la página de bienvenida
!insertmacro MUI_PAGE_WELCOME
;Página donde mostramos el contrato de licencia
!insertmacro MUI_PAGE_LICENSE "Licencia.txt"
; Página donde se muestran las distintas secciones definidas
!insertmacro MUI_PAGE_COMPONENTS
;Página donde se selecciona el directorio donde instalar
nuestra aplicación
!insertmacro MUI_PAGE_DIRECTORY
;Página de instalación de ficheros
!insertmacro MUI_PAGE_INSTFILES
;Página final
!insertmacro MUI_PAGE_FINISH
```

c. **Secciones**:

En un instalador pueden hacerse categorías de instalación y así separar la instalación en varios componentes, dando a elegir al usuario cuales instalar y cuáles no.

```
Section "My Program"
    SetOutPath $INSTDIR
    File "My Program.exe"
    File "Readme.txt"
SectionEnd
```

Dentro de cada sección usamos instrucciones que son ejecutadas en tiempo de ejecución. Estas instrucciones, leen y escriben en el registro, crean, borran y copian ficheros y directorios, crean accesos directos etc.

Los desinstaladores también pueden tener varias secciones teniendo como prefijo "un.":

```
Section "Installer Section"
SectionEnd
Section "un.Uninstaller Section"
SectionEnd
```

d. **Funciones**:

Las funciones contienen código semejante a las secciones, pero se diferencian de estas en el modo en el que se llaman. Hay dos tipos de funciones, las definidas por el usuario, que se llaman con la instrucción Call y las que se activan cuando ocurren determinados eventos en la instalación:

```
Function .onInit
   MessageBox MB_YESNO "Esto instalará mi programa ¿Quieres continuar?"
  IDYES gogogo
   Abort
   gogogo:
```

```
FunctionEnd
```

Abort es una función especial que hace que el instalador termine inmediatamente.

e. **Variables**:

En este lenguaje se declaran las variables mediante Var:

Sintaxis: `Var [/GLOBAL] VARIABLE` (El indicador / GLOBAL no es necesario fuera de las secciones y funciones)

Ejemplo:

```
Var ejemplo
;Declaramos la variable
Function PruebaVar
 Var /GLOBAL ejemplo2
 StrCpy $ejemplo "valor"
 StrCpy $ejemplo2 "otro valor"
;ahora la variable ejemplo vale "valor" y la variable ejemplo2 vale "otro valor"
FunctionEnd
```

Una vez que hemos implementado nuestro script de instalación, la herramienta NSIS lo compila, creando un ejecutable que será nuestro instalador.

Estos son a grandes rasgos los componentes del lenguaje de script de NSIS, con el paquete se incluye un completo sistema de ayuda del lenguaje, así como diferentes ejemplos.

4. Ejemplos:

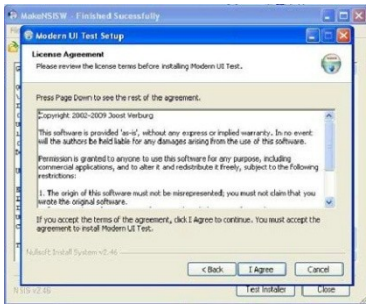
Para facilitar el uso de la herramienta, la aplicación NSIS incorpora un conjunto de scripts de ejemplo, que nos ayudan a crear nuestros instaladores de forma personalizada. El objetivo de NSIS es compilar el script de instalación, y crear un instalador, para ello podemos trabajar a partir de un script de ejemplo y adaptarlo a nuestras necesidades.

El resultado de la compilación y prueba de algunos scripts de ejemplo que vienen con la aplicación serían:

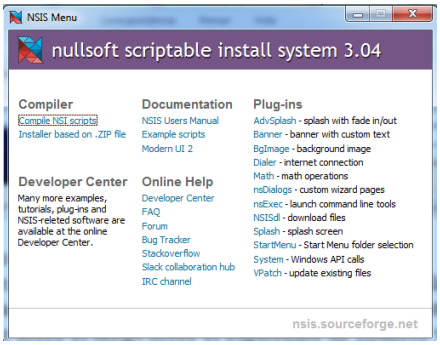
Donde tenemos una plantilla para instalar una aplicación .exe dentro de un sistema Windows.

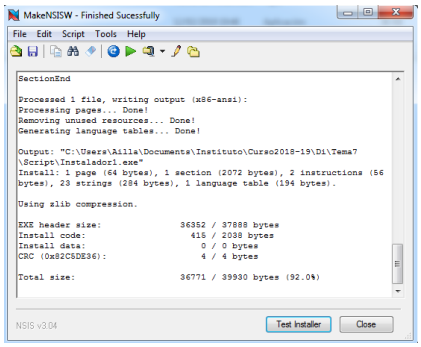
A continuación podemos ver la plantilla para crear un instalador Moderno UI, donde en la segunda ventana se visualiza el fichero de licencia (licencia.txt).

El programa nos proporciona un conjunto muy amplio de **plantillas**, que podremos reutilizar y adaptar, para crear nuestros propios scripts de instalación.



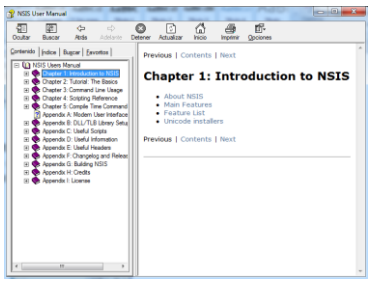
5. Entorno de compilación:
   Una vez que hemos diseñado nuestro script de instalación, nos queda el paso final, que es compilarlo. Para generar el fichero ejecutable del instalador nos bastará con pulsar el botón derecho sobre el script sobre el explorador de ficheros y seleccionar "Compile NSIS Script":





   El compilador va a ir interpretando una a una todas las líneas del script, de forma que si no se produce ningún error, se genera un archivo ejecutable, que será nuestro instalador.

Además de la documentación ya referenciada en la Web, la herramienta dispone de su manual de usuario:



10. Ficheros firmados digitalmente.

Caso práctico

Para garantizar la autenticidad de la aplicación de Gestión Hotelera, Ada propone que los ficheros de instalación, ya sean paquetes JAR o instalables exe, vayan firmados digitalmente.

Una firma digital o esquema de firma digital, es un esquema matemático que permite demostrar la autenticidad de un documento digital, en nuestro caso un fichero.

Una firma digital válida nos garantiza que el fichero ha sido creado por una empresa o autor conocido y que no ha sido modificado durante su transferencia.

En el caso de la distribución de software, sobre todo a través de descargas de Internet, nos garantiza la autenticidad del fichero, el autor y que durante la descarga no ha sufrido ningún tipo de alteración.

La plataforma Java nos permite firmar digitalmente ficheros Jar. Cuando se firma un fichero Jar, lo que se está asegurando a los usuarios que ejecuten la aplicación es la autoría de ese software. Cuando digitalizamos el fichero, cualquiera puede reconocer la firma digital del autor. El proceso de reconocimiento de la **firma digital** se denomina **verificación**.

La firma y verificación de ficheros es una parte importante de la arquitectura de seguridad de la plataforma Java. Podemos configurar la política de seguridad para permitir a un applet y a una aplicación puedan realizar operaciones normalmente prohibidas como pueden ser la lectura y escritura de ficheros locales, ejecución programas, etc.

La plataforma Java permite firmar y verificar usando números especiales denominados **claves públicas y privadas**. Las claves públicas y privadas vienen en parejas, y tienen roles complementarios. La clave privada es el "lápiz" electrónico que nos permite firmar un fichero. La clave privada sólo es conocida por el autor de la firma. Un fichero firmado con llave privada, solo puede ser verificado con su correspondiente llave pública. Existe un elemento adicional necesario para la firma y verificación. Este elemento es el certificado que el firmador incluye en un fichero JAR firmado. El certificado es una declaración digital firmada reconocida por una autoridad de certificación que indica quién es el dueño de una clave pública determinada. Resumiendo, la firma digital:

- ✓ El desarrollador firma el fichero JAR usando una llave privada.
- ✓ La correspondiente llave pública se coloca en el fichero JAR, junto con el certificado, que estará disponible para que cualquiera pueda verificar la firma.

RESPONDE
7. El reconocimiento de la firma digital de una archivo JAR se conoce como:
   a) Clave privada.
   b) Verificación.
   c) Clave pública.
   d) Certificación.

11. Herramientas de compilación: Ant vs Maven vs Gradle.

Exploraremos tres herramientas de automatización de compilación de Java que dominaron el ecosistema de JVM: Ant, Maven y Gradle.

Presentaremos cada uno de ellos y exploraremos cómo evolucionaron las herramientas de automatización de compilación de Java.

11.1. Apache Ant

Al principio, Make era la única herramienta de automatización de construcción disponible más allá de las soluciones de cosecha propia. Make existe desde 1976 y, como tal, se utilizó para crear aplicaciones Java en los primeros años de Java.

Sin embargo, muchas convenciones de los programas C no encajaban en el ecosistema de Java, por lo que con el tiempo Ant asumió el control como una mejor alternativa.

Apache Ant ("Otra herramienta ordenada") es una biblioteca de Java que se utiliza para automatizar los procesos de creación de aplicaciones Java. Además, Ant se puede utilizar para crear aplicaciones

que no sean Java. Inicialmente formaba parte del código base de Apache Tomcat y se lanzó como proyecto independiente en 2000.

En muchos aspectos, Ant es muy similar a Make, y es lo suficientemente simple como para que cualquiera pueda comenzar a usarlo sin ningún requisito previo en particular. Los archivos de compilación de Ant están escritos en XML y, por convención, se denominan build.xml.

Las diferentes fases de un proceso de construcción se denominan objetivos "targets".

Aquí hay un ejemplo de un archivo **build.xml** para un proyecto Java simple con la clase principal HelloWorld:

```
<project>
    <target name="clean">
        <delete dir="classes" />
    </target>
    <target name="compile" depends="clean">
        <mkdir dir="classes" />
        <javac srcdir="src" destdir="classes" />
    </target>
    <target name="jar" depends="compile">
        <mkdir dir="jar" />
        <jar destfile="jar/HelloWorld.jar" basedir="classes">
            <manifest>
                <attribute name="Main-Class" value="antExample.HelloWorld"/>
            </manifest>
        </jar>
    </target>
    <target name="run" depends="jar">
      <java jar="jar/HelloWorld.jar" fork="true" />
      </target>
</project>
```

Este archivo de compilación define cuatro objetivos: clean, compile, jar y run. Por ejemplo, podemos compilar el código ejecutando:

ant compile

Esto activará primero la limpieza de destino, que eliminará el directorio "classes". Después de eso, la compilación de destino volverá a crear el directorio y compilará la carpeta src en él.

El principal beneficio de Ant es su **flexibilidad**. Ant no impone convenciones de codificación ni estructuras de proyecto. En consecuencia, esto significa que Ant requiere que los desarrolladores escriban todos los comandos por sí mismos, lo que a veces conduce a enormes archivos de compilación XML que son difíciles de mantener.

Dado que no existen convenciones, el simple hecho de conocer Ant no significa que entenderemos rápidamente cualquier archivo de compilación de Ant. Es probable que te lleve un tiempo

acostumbrarte a un archivo Ant desconocido, lo cual es una desventaja en comparación con las otras herramientas más nuevas.

Al principio, Ant no tenía soporte integrado para la gestión de dependencias. Sin embargo, como la gestión de dependencias se convirtió en una necesidad en los últimos años, Apache Ivy se desarrolló como un subproyecto del proyecto Apache Ant. Está integrado con Apache Ant y sigue los mismos principios de diseño.

Sin embargo, las limitaciones iniciales de Ant debido a la falta de soporte integrado para la gestión de dependencias y las frustraciones al trabajar con archivos de compilación XML no gestionables llevaron a la creación de Maven.

11.2. Maven

Apache Maven es una herramienta de automatización de compilación y gestión de dependencias, que se utiliza principalmente para aplicaciones Java. Maven continúa usando archivos XML como Ant pero de una manera mucho más manejable. El nombre del juego aquí es la convención sobre la configuración.

Mientras que Ant ofrece flexibilidad y requiere que todo se escriba desde cero, Maven se basa en convenciones y proporciona comandos predefinidos (objetivos).

En pocas palabras, Maven nos permite enfocarnos en lo que debería hacer nuestra construcción y nos brinda el marco para hacerlo. Otro aspecto positivo de Maven fue que proporcionó soporte integrado para la gestión de dependencias.

El archivo de configuración de Maven, que contiene las instrucciones de gestión de dependencia y compilación, se llama por convención **pom.xml**. Además, Maven también prescribe una estructura de proyecto estricta, mientras que Ant también proporciona flexibilidad allí.

Aquí hay un ejemplo de un archivo pom.xml para el mismo proyecto Java simple con la clase principal HelloWorld de antes:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ieslosmontecillos</groupId>
    <artifactId>mavenExample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <description>Maven example</description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
```

```
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>.
```

Sin embargo, ahora la estructura del proyecto también se ha estandarizado y se ajusta a las convenciones de Maven:

```
+---src
|   +---main
|   |   +---java
|   |   |   \---com
|   |   |       \---ieslosmontecillos
|   |   |           \---maven
|   |   |                   HelloWorld.java
|   |   |
|   |   \---resources
|   \---test
|   +---java
|   \---resources
```

A diferencia de Ant, no es necesario definir cada una de las fases en el proceso de construcción manualmente. En su lugar, podemos simplemente llamar a los comandos integrados de Maven.

Por ejemplo, podemos compilar el código ejecutando:

mvn compile

En esencia, como se indica en las páginas oficiales, Maven puede considerarse un framework de ejecución de plugins, ya que todo el trabajo se realiza mediante plugins. Maven admite una amplia gama de plugins disponibles, y cada uno de ellos se puede configurar adicionalmente.

Uno de los plugins disponibles es Apache Maven Dependency Plugin, que tiene un objetivo de copia de dependencias que copiará nuestras dependencias en un directorio específico.

Para mostrar este plugins en acción, incluyamos este complemento en nuestro archivo pom.xml y configuremos un directorio de salida para nuestras dependencias:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
```

```
                        </goals>
                        <configuration>
                            <outputDirectory>target/dependencies</outputDirectory>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

Este plugin se ejecutará en una fase de paquete, así que si ejecutamos:

mvn package

Ejecutaremos este plugin y copiaremos las dependencias en la carpeta de target/dependencies.

Maven se hizo muy popular ya que los archivos de compilación ahora estaban estandarizados y llevó mucho menos tiempo mantener los archivos de compilación, en comparación con Ant. Sin embargo, aunque más estandarizados que los archivos Ant, los archivos de configuración de Maven aún tienden a ser grandes y engorrosos.

Las estrictas convenciones de Maven tienen el precio de ser mucho menos flexible que Ant. La personalización de objetivos es muy difícil, por lo que escribir scripts de compilación personalizados es mucho más difícil de hacer, en comparación con Ant.

11.3. Gradle

Gradle es una herramienta de automatización de compilación y gestión de dependencias que se basó en los conceptos de Ant y Maven.

Una de las primeras cosas que podemos notar sobre Gradle es que no usa archivos XML, a diferencia de Ant o Maven.

Con el tiempo, los desarrolladores se interesaron cada vez más en tener y trabajar con un lenguaje específico de dominio, lo que, en pocas palabras, les permitiría resolver problemas en un dominio específico utilizando un lenguaje adaptado a ese dominio en particular.

Esto fue adoptado por Gradle, que utiliza un DSL basado en Groovy o Kotlin. Esto condujo a archivos de configuración más pequeños con menos desorden, ya que el lenguaje fue diseñado específicamente para resolver problemas de dominio específicos. El archivo de configuración de Gradle se llama por convención build.gradle en Groovy, o build.gradle.kts en Kotlin.

Tenga en cuenta que Kotlin ofrece mejor soporte IDE que Groovy para autocompletar y error.

Aquí hay un ejemplo de un archivo **build.gradle** para el mismo proyecto Java simple con la clase principal HelloWorld de antes:

```
apply plugin: 'java'
repositories {
    mavenCentral()
}
```

```
jar {
    baseName = 'gradleExample'
    version = '0.0.1-SNAPSHOT'
}

dependencies {
    testImplementation 'junit:junit:4.12'
}
```

Podemos compilar el código ejecutando:

gradle classes

En esencia, Gradle proporciona intencionalmente muy poca funcionalidad. Los plugins agregan todas las funciones útiles. En nuestro ejemplo, estábamos usando un complemento de Java que nos permite compilar código Java y otras características valiosas.

Gradle dio a sus pasos de compilación el nombre de "tasks", a diferencia de los "targets" de Ant o las "phases" de Maven. Con Maven, usamos Apache Maven Dependency Plugin, y es un objetivo específico copiar las dependencias a un directorio específico. Con Gradle, podemos hacer lo mismo usando tareas:

```
task copyDependencies(type: Copy) {
   from configurations.compile
   into 'dependencies'
}
```

Podemos ejecutar esta tarea ejecutando:

gradle copyDependencies

Ampliaremos nuetros conocimientos siguiendo el curso de OpenWebinars: Curso de Gradle 6.0 al que tenemos acceso con nuestra cuenta del IES: https://openwebinars.net/cursos/gradle-6-0/.

12. Instalación de aplicaciones desde un servidor.

Caso práctico

Carlos está comprobando los requisitos que solicitan los servidores de Internet que distribuyen aplicaciones. También baraja la posibilidad de crear un servidor propio de BK Programación, donde alojen y distribuyan las aplicaciones desarrolladas por la empresa.

Cuando se implementa un paquete software para su distribución, existe la posibilidad de alojarlo en un servidor web para que sea accesible a un conjunto de usuarios. Estos usuarios pueden instalar directamente el paquete en su ordenador, simplemente haciendo click en un hipervínculo. Esta forma de distribuir software es muy común en distribuciones Linux como Ubuntu.

Las principales ventajas de los servidores de aplicaciones es la centralización y la disminución de la complejidad del desarrollo de aplicaciones, ya que éstas no necesitan ser programadas. Basta con ensamblarlas desde bloques y ponerlas en el servidor a disposición de quien las necesite.

Para poder instalar aplicaciones desde un servidor, lo único que el usuario realiza es la pulsación sobre el hipervínculo donde aparece el paquete que quiere instalar. Un ejemplo sería la instalación de firefox en ubuntu.

Para poder realizar esta instalación, existe **apturl**. AptUrl es un miniprograma gráfico para instalar programas desde el repositorio donde se encuentran. Para poder distribuir aplicaciones utilizando esta herramienta, se edita una página web, donde se suele dar una descripción del programa a instalar, y se añade el enlace siguiente:

```
<a href="apt:paquete">Nombre del paquete a instalar</a>
```

Si queremos distribuir múltiples paquetes en el mismo enlace, la sintaxis sería la siguiente:

```
<a href="apt:paquete1, paquete2, paquet3"> Nombre de la aplicación </a>
```

Un ejemplo de instalación de esta forma, sería:

```
apturl apt:pidgin,pidgin-plugin-pack
```

Donde se instalaría la aplicación Pidgin y Pidgin Plugin Pack.

13. Descarga y ejecución de aplicaciones ubicadas en servidores web.

Caso práctico

Ana está investigando las características que debieran tener las aplicaciones desarrolladas por BK Programación, para poder ser ubicadas en servidores de Internet, y desde allí, que los posibles clientes las puedan descargar e instalar.

Se puede presentar otra posibilidad, y es que la distribución se produzca a través de un fichero que tenga empaquetada y comprimida la aplicación. Se pueda dar el caso de la distribución a través de fichero iso, los cuales nos van a obligar a crear un CD o DVD con el software de instalación de la aplicación, o bien utilizar programas para montar la imagen ISO y poder instalar el software.

Si el fichero descargado es un ejecutable, bastará con la ejecución de ese fichero para que se produzca su instalación. Este sería el caso de la descarga de un paquete autoinstalable .exe en Windows o un sh en Linux.

Otro tipo de distribución, muy común en Linux, es la posibilidad de descargar desde Internet paquetes deb o rpm. Esto paquetes contienen aplicaciones instalables en nuestra distribución, deb si estamos en Ubuntu. Para poder instalar estos paquetes se puede usar el gestor de paquetes de Ubuntu, bien entorno gráfico, o bien en entorno de consola.

Otra forma de distribuir software a través de web, sería el empaquetado en paquete jar o archivos comprimidos zip o rar. En el caso de la primera forma de distribución, lo único que necesitamos para instalar el paquete, sería disponer de una máquina virtual Java. En el caso de distribuir los paquetes comprimidos, necesitamos en el ordenador cliente un programa para descomprimir compatible. En todos los casos, hay que tener en cuenta que el software que se descarga y se quiere instalar, esté diseñado para nuestro sistema operativo y sea compatible con nuestra arquitectura.

13.1. Implantación de Aplicaciones JavaFX en el navegador (Java Web Start)

El software de Java Web Start permitía descargar y ejecutar aplicaciones Java desde un servidor Web. El software de Java Web Start: permite activar las aplicaciones con un simple clic, garantiza que se está ejecutando la última versión de la aplicación y elimina complejos procedimientos de instalación o actualización.

Java Web Start está incluido en Java Runtime Environment (JRE) desde la versión Java 5.0. Esto significa que al instalar Java, Java Web Start se instalará automáticamente. El software de Java Web Start se inicia automáticamente, cuando una aplicación Java que utiliza tecnología Java Web Start se descarga por primera vez. El software de Java Web Start almacena en caché toda la aplicación de forma local en el ordenador. Por lo tanto, los siguientes inicios son casi instantáneos, ya que todos los recursos necesarios ya están disponibles localmente. Cada vez que se inicie la aplicación, el componente de software de Java Web Start comprueba el sitio web de la aplicación para comprobar si existe una nueva versión y, si es así, la descarga e inicia automáticamente.

Java Web Start fue desarrollado por Sun (ahora Oracle) y se eliminó en la versión Java SE 11. Para implementar aplicaciones JavaFX en el navegador la plataforma dispone de la herramienta **Deployment Toolkit**. En el siguiente enlace de Oracle se realiza una descripción general de la API, información sobre los métodos de devolución de llamada y ejemplos típicos de uso. La forma recomendada de insertar una aplicación JavaFX en una página web o iniciarla desde un navegador web es utilizar la biblioteca Deployment Toolkit.
Enlace: https://docs.oracle.com/javafx/2/deployment/deployment_toolkit.htm.

Deployment Toolkit proporciona una API de JavaScript para simplificar la implementación web de las aplicaciones JavaFX y mejorar la experiencia del usuario final para que la aplicación se inicie.

Además de proporcionar funcionalidad para agregar etiquetas HTML necesarias para insertar contenido JavaFX en la página web, Deployment Toolkit también hace lo siguiente:

- ✓ Detecta si el entorno de usuario es compatible
- ✓ Ofertas para instalar el JavaFX Runtime si es necesario
- ✓ Proporciona comentarios visuales al usuario mientras se carga la aplicación
- ✓ Informes al usuario en caso de error inesperado
- ✓ Proporciona otras API auxiliares para el desarrollador, que se pueden usar para simplificar la implementación y la integración con la página web

La forma recomendada de utilizar Deployment Toolkit es importar el archivo Deployment Toolkit JavaScript desde una ubicación compartida en http://java.com/js/dtjava.js.

De esta forma, la aplicación siempre utiliza la última forma recomendada de integración en una página web. Si no puedes usar la ubicación compartida, puedes incluir los archivos necesarios con tu aplicación utilizando la opción includeDT en la tarea Ant.<fx: deploy>.

Hay que tener en cuenta que para la mayoría de los casos de uso simple, la mayoría si no todo el código necesario para la implementación es generado por las herramientas de empaquetado JavaFX (NetBeans las usa en aplicaciones JavaFX) y puede usar devoluciones de llamada para ajustar aún más el comportamiento predeterminado. Sin embargo, si se requiriese más flexibilidad, se puede usar las API de Deployment Toolkit directamente.

Sin embargo problemas de seguridad en el ámbito de la ciberseguridad de los plugin Java necesarios para la ejecución de Java Web Start, hacen que actualmente los navegadores, Firefox, Chrome, etc. abandonen los plugin, por lo que no es viable esta solución.

13.2. Implantación de Aplicaciones JavaFX en el navegador (Jpro)

Jpro technologies AG salió de JavaOne en San Francisco al ganar el Premio Duke's Choice Technology. Jpro, "Es una herramienta con la que se puede ejecutar programas Java en el navegador sin el plugin Java… https://www.jpro.one/.

Al usar JPro se permite que la aplicación Java o página web Java se ejecute en el navegador. No importa si el navegador se ejecuta en una computadora de escritorio o en un dispositivo móvil. Puede subirse a la nube, ejecutarlo en un servidor o se puede ejecutar en localhost (para desarrollo). Por supuesto, no se requiere ningún plugin.

Se usa el mismo código para dispositivos **móviles**. El código Java que se utiliza para los navegadores puede instalarse como una aplicación móvil nativa. Se puede ejecutar el código en el navegador del dispositivo o como una aplicación nativa. Android e iOS son compatibles.

Y puede ejecutarse el mismo código Java en cualquiera de los sistemas operativos comunes de su escritorio. Básicamente, puede ejecutarlo donde sea que se ejecute una JVM.

Con JPro, la aplicación se ejecuta de forma remota, es decir, en la nube o en algún otro entorno de servidor. Se puede alojar en las instalaciones o en algún lugar de forma remota. Pero, para algunos casos de uso, puede ser interesante saber que con JPro también existe la opción de NO usar el navegador estándar como IU. En su lugar, podría se usar un JPro Renderer preinstalado.

Tal vez se prefiera que la aplicación se ejecute de forma centralizada, para facilitar la administración, tener una escala infinita, etc., pero la aplicación no debería tener las limitaciones que requiere un navegador, como restricciones de Sandboxing, por ejemplo…de interés la biblioteca de diseño de material JavaFX: JFoenix

Para implantar la aplicación JavaFX en la web usando Jpro usaremos **Gradle** como herramienta de construcción de proyectos (https://gradle.org). Gradle, es una herramienta que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben utilizando Groovy o Kotlin DSL (Domain Specific Language). JPro proporciona un plugin para Gradle, que le permite iniciar fácilmente JPro desde un proyecto existente, lo que obviamente es bastante práctico durante el proceso de desarrollo.

El procedimiento pasa por crear un proyecto Gradle (lo podemos hacer con Netbeans que ya incluye un plugin) y asegurar que la clase principal del proyecto extienda de JProApplication, y que el archivo index.html, el jpro.conf y el archivo gradle.build (script de compilación) estén configurados correctamente para iniciar la aplicación…estos son los cambios:

En el archivo de la clase principal de la aplicación extenderá de JproApplication en lugar de Application… src\main\java**Main.java**

```
public class Main extends JProApplication {
```

Entre los archivos del proyecto…. el archivo **gradle.build** de construcción del proyecto contendrá:

```
/**
 ****************** Script Configuration ******************/
buildscript {
  repositories {
    jcenter()

    maven {
      url "https://sandec.bintray.com/repo"
    }
  }

  dependencies {
    classpath 'com.sandec.jpro:jpro-plugin-gradle:2019.2.6'
  }
}

/**
 ****************** Java Configuration ******************/
apply plugin: 'java'
apply plugin: 'application'
apply plugin: 'org.openjfx.javafxplugin'

//Para proyecto java que requiere JPA el sourceCompatibility será 1.8
compileJava {
  sourceCompatibility = 11
  targetCompatibility = 11
}

repositories {
  jcenter()
}

javafx {
    version = '11'
    modules = [ 'javafx.base', 'javafx.graphics', 'javafx.controls',
'javafx.fxml', 'javafx.media', 'javafx.web' ]
}
/**
 * App Main Class */
application {
      mainClassName = 'appsondeojpro.Main'
}
```

```
/**
 ****************** jpro Configuration ******************/
apply plugin: 'com.sandec.jpro'

/**
 * jpro settings*/
jpro {
  // for debugging
  // JVMArgs << '-
agentlib:jdwp=transport=dt_socket,server=n,address=5006,suspend=y'

  JVMArgs << '-Xmx1000m'

  //jpro server port
  port = 8080

}
```

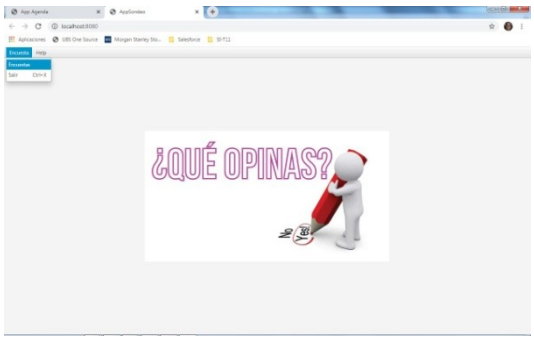
En el archive jpro.conf incluiremos la referencia a la clase principal…\src\main\resources**jpro.conf**:

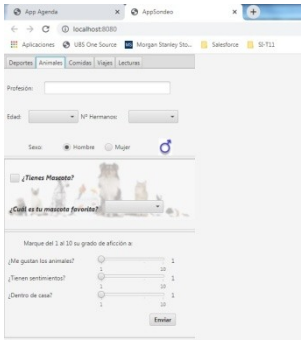
```
//jpro apps
jpro.applications {
  "appsondeojpro" = appsondeojpro.Main
}
```

En el archive index.html ….\src\main\resources\jpro\html**index.html**:

```
<!DOCTYPE html>

<html>
    <head>
        <title>AppSondeos</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0,
    user-scalable=no">
        <link rel="stylesheet" type="text/css" href="/jpro/css/jpro-
    fullscreen.css">
        <link rel="stylesheet" type="text/css" href="/jpro/css/jpro.css">
        <script src="/jpro/js/jpro.js" type="text/javascript"></script>
    </head>
    <body>
          <jpro-app href="/app/appsondeojpro" fullscreen="true"></jpro-app>
    </body>
</html>
```





RESPONDE

8. En una instalación desde un servidor web (JavaWebStart):
   a) El usuario siempre instala directamente la aplicación sin tener que guardarla antes.
   b) La instalación siempre sigue el mismo procedimiento, independientemente del tipo de fichero a descargar.
   c) La aplicación se instala automáticamente sólo si se trata de un archivo ejecutable.

Anexo I: Firma digital en fichero JAR.

En este punto, vamos a resumir cómo utilizar las herramientas proporcionadas en el Kit de Desarrollo Java ™ para firmar y verificar los archivos JAR:

La plataforma Java ™ permite firmar digitalmente los archivos JAR. Firmamos digitalmente un archivo por la misma razón que podemos firmar un documento en papel con pluma y tinta, para que los lectores sepan quién escribió el documento, o al menos que el documento tiene nuestra aprobación.

Cuando se firma una carta, por ejemplo, todos los que reconocen su firma pueden confirmar quién escribió la carta. Del mismo modo cuando se firma digitalmente un archivo, cualquier persona que "reconoce" su firma digital sabe que el archivo procede de dicha persona. El proceso de "reconocimiento" de las firmas electrónicas se llama verificación.

La posibilidad de firmar y verificar archivos es una parte importante de la arquitectura de seguridad de la plataforma Java. La seguridad es controlada por la política de seguridad que esté en vigor en tiempo de ejecución. Puede configurar la directiva para conceder privilegios de seguridad para los applets y aplicaciones. Por ejemplo, puede conceder permiso a un applet para realizar operaciones normalmente prohibidas, tales como leer y escribir archivos locales o ejecutar programas locales ejecutables. Si has descargado un código firmado por una entidad de confianza, puede utilizar este hecho como un criterio para decidir cuál de los permisos de seguridad asignará al código.

Una vez que tú (o tu navegador) ha comprobado que un applet es de una fuente confiable, tú puedes relajar las restricciones de seguridad para que el applet realice operaciones que normalmente estarían prohibidas. Un applet de confianza puede tener libertades como se especifica en el archivo de política en vigor.

La plataforma Java permite a la firma y la verificación mediante el uso de números especiales llamados claves pública y privada. Las claves públicas y claves privadas vienen en pares, y desempeñan papeles complementarios.

La clave privada es la "pluma" electrónica con la que se puede firmar un archivo. Como su nombre indica, la clave privada es conocida sólo por tí, para que nadie más puede "falsificar" tu firma.
Un archivo firmado con su clave privada sólo puede ser comprobada por la correspondiente clave pública.

Las claves pública y privada por sí solas, sin embargo, no son suficientes para verificar realmente la firma. Incluso si has verificado que un archivo firmado contiene un par de claves correspondientes

entre sí, aún necesitas alguna manera de confirmar que la clave pública en realidad proviene de la persona de la que la firma pretende provenir.

Se requiere un elemento más, por lo tanto, para hacer la firma y verificación. Este elemento adicional es el certificado que el firmante incluye en un fichero JAR firmado. Un certificado es una declaración firmada digitalmente de una autoridad de certificación reconocida que indica que es dueño de una clave pública determinada. Una autoridad de certificación son las entidades (generalmente empresas especializadas en seguridad digital) que son de confianza en toda la industria para firmar y emitir certificados para las claves y sus propietarios.

En el caso de los archivos JAR firmados, el certificado indica que es dueño de la clave pública contenida en el archivo JAR.

Al firmar un fichero JAR su clave pública se coloca dentro del archivo, junto con un certificado asociado para que sea de fácil acceso para su uso por cualquier persona que desee verificar su firma.

Para resumir la firma digital:
- ✓ El firmante firma el archivo JAR utilizando una clave privada.
- ✓ La clave pública correspondiente se coloca en el archivo JAR, junto con su certificado, por lo que está disponible para su uso por cualquier persona que quiera verificar la firma.

Resúmenes y el fichero de firmas.
Al firmar un fichero JAR, cada fichero del archivo tiene una entrada de resumen en el archivo de manifiesto. He aquí un ejemplo de lo que tal entrada podría ser:

```
Name: test/classes/ClassOne.class
SHA1-Digest: TD1GZt8G11dXY2p4olSZPc5Rj64=
```

Los valores de resumen son representaciones hash o codificadas de los contenidos de los archivos tal como estaban en el momento de la firma. El resumen de un archivo cambiará si y sólo si el propio fichero cambia.

Cuando se firma un archivo JAR se genera automáticamente un archivo de firma y se coloca en el directorio META-INF del archivo JAR, el mismo directorio que contiene el archivo de manifiesto. Los archivos de firmas tienen nombres de archivo con una extensión. SF. He aquí un ejemplo del contenido de un archivo de firma:

```
Signature-Version: 1.0
SHA1-Digest-Manifest: h1yS+K9T7DyHtZrtI+LxvgqaMYM=
Created-By: 1.6.0 (Sun Microsystems Inc.)
Name: test/classes/ClassOne.class
SHA1-Digest: fcav7ShIG6i86xPepmitOVo4vWY=
Name: test/classes/ClassTwo.class
SHA1-Digest: xrQem9snnPhLySDiZyclMlsFdtM=
Name: test/images/ImageOne.gif
SHA1-Digest: kdHbE7kL9ZHLgK7akHttYV4XIa0=
```

```
Name: test/images/ImageTwo.gif
SHA1-Digest: mF0D5zpk68R4oaxEqoS9Q7nhm60=
```

Como se puede ver, el archivo de firma contiene entradas de resumen para los archivos del archivo comprimido que se parecen a las entradas de valor de síntesis, en el manifiesto. Sin embargo, mientras que los valores de resumen en el manifiesto se calculan a partir de los propios archivos, los valores de resumen en el archivo de firma se calculan a partir de las entradas correspondientes en el manifiesto. Los archivos de firma también contienen un valor de resumen para el manifiesto completo (ver la cabecera SHA1-Digest-Manifest en el ejemplo anterior).

Cuando un fichero JAR firmado está siendo verificado, los resúmenes de cada uno de los archivos se vuelven a calcular y comparar con los resúmenes, registrada en el manifiesto para asegurar que el contenido del archivo JAR no han cambiado desde que se firmó. Como una comprobación adicional, los valores de resumen para el archivo de manifiesto en sí se vuelven a calcular y se compara con los valores registrados en el archivo de firma.
Puede obtener información adicional sobre los archivos de firma en la página Formato de manifiesto de la documentación del JDK ™.

El archivo de bloque de firma.
Además de los archivos de firmas, se coloca automáticamente un archivo de bloque de firma en el directorio META-INF cuando se firma un archivo JAR. A diferencia del archivo de manifiesto o el archivo de firma, los archivos de bloques de firmas no son legibles.

El archivo de bloque de firma contiene dos elementos esenciales para la verificación: La firma digital para el archivo JAR que se generó con la clave privada del firmante, El certificado que contiene la clave pública del firmante, para ser utilizado por cualquier persona que desea verificar el archivo JAR firmado.

Los nombres de los archivos de bloques de firmas suele tener una extensión DSA lo que indica que fueron creados por el algoritmo predeterminado de firma digital. Otras extensiones de nombre de archivo son posibles si las claves asociadas con algún algoritmo estándar de otros se utilizan para la firma.

Firmar archivos JAR.
Para firmar un fichero JAR, primero debes tener una **clave privada**. Las claves **privadas** y sus correspondientes claves **públicas** se almacenan en bases de datos protegidas. Un keystore puede contener las claves de muchos firmantes potenciales. Cada clave del almacén de claves puede ser identificado por un alias que suele ser el nombre del firmante y que posee la clave.
Se puede crear una firma usando la herramienta **keytool** del jdk que encontramos en (en el caso de Windows): %PROGRAMFILES%\Java\Jdk1.8.0_211\bin\

```
keytool -genkey -alias profesora -keystore almacen -validity 150 -v
```

Al crearse nos solicitará una contraseña para almacenar con seguridad y la información personal que nos identifica, Nombre, Departamento, Empresa, Localidad, Provincia, País. Con esto ya dispondríamos de una firma en nuestro almacén aunque no está certificada por una entidad

certificadora como podía ser la FNMT¿?, por lo que esta firma no tiene validez actualmente ya que cualquiera la puede manipular.

Ya con la firma, (en nuestro caso auto-firma) solo queda usar la herramienta de verificación, el comando **jarsigner**, por lo que nos referiremos a ella como "Jarsigner".

La forma básica del comando para la firma de un archivo JAR es

```
jarsigner -keystore almacen -signedjar sApp.jar App.jar profesora -verbose
```

En este comando: App.jar es la ruta del archivo JAR a firmar, alias es el alias que identifica la clave privada que se utiliza para firmar el archivo JAR, y el certificado asociado de la tecla.

La herramienta Jarsigner pedirá la contraseña para el almacén de claves y alias.

En esta forma básica del comando se supone que el almacén de claves a utilizar está en un archivo llamado keystore en su directorio personal. Se va a crear la firma y los archivos de bloques de firmas con nombres x.SF y x.DSA respectivamente, donde x son las primeras ocho letras del alias, todas en mayúsculas. Este comando básico sobrescribe el archivo JAR original con el fichero JAR firmado.

En el siguiente enlace puedes acceder al tutorial de Oracle donde se muestra cómo utilizar herramientas como keytool, jarsigner, PolicyTool y jar para empaquetar archivos en archivos JAR (Java ARchive) para su posterior firma con la herramienta jarsigner: https://docs.oracle.com/javase/tutorial/security/toolsign/index.html.