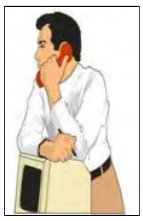
Unidad Didáctica VI.- Generación de informes. Generación de ayuda

CASO.

José está muy contento de cómo están quedando las aplicaciones cliente que en SI Andalucía están realizando con la nueva herramienta de desarrollo Oracle JDeveloper 10g. Pero ve que le faltan algunos detalles para llegar a tener un aspecto profesional y ofrecer a los clientes los requerimientos que demandan. Entre estos detalles están los informes, que no ha visto en ninguna de las aplicaciones que se están desarrollando, y que tiene la certeza que eran requisitos de dichas aplicaciones. Consulta con Víctor el tema de los informes y éste le comenta que efectivamente aún no se han puesto con ello. José le dice que debe ponerse inmediatamente con los informes para la aplicación del taller informático que están desarrollando, ya que los plazos se echan encima. Le propone que le vaya echando un vistazo al problema de la generación de informes de distintos tipos, para así poder incluirlo después en la aplicación del taller. Víctor le dice que no se preocupe, que se pone con ello y en cuanto haya profundizado en el tema se lo comenta para que le indique cuáles son los informes que se deben generar para dicha aplicación.



Víctor, después de mucho buscar en el entorno de desarrollo, no ve la manera de generar informes y debido a la prisa que corre decide pedirle ayuda a Carmen. Víctor le pregunta a Carmen si sabe cómo generar informes "customizados" en JDeveloper. Carmen frunce el ceño y suavemente le dice que "customizar" no es una palabra de la lengua española. Con lo rica que es nuestra lengua no deberíamos echar mano de malas traducciones, pudiendo utilizar los recursos que nos brinda nuestro idioma, que en este caso sería "personalizar". Le pide a Víctor que intente prestar atención a estos detalles ya que suenan fatal y es un error en el que los informáticos caemos muy a menudo.

Dejando de lado la lengua, Carmen le dice a Víctor que es normal que no haya visto cómo se generan informes desde Oracle JDeveloper 10g, ya que con el entorno en sí no se puede generar informes. Víctor se asusta un poco debido a la prisa que corre este tema, pero Carmen le dice que se tranquilice ya que existen soluciones para casi todo. Carmen le comenta a Víctor que cuando estudiaba en el IES Aguadulce, utilizaban una herramienta llamada JasperReport que permitía generar informes de todo tipo en Java de una forma sencilla. Carmen se compromete a explicarle cómo utilizarla y le asegura que en un par de días está generando todo tipo de informes "personalizados".

Efectivamente, como bien dice Carmen, Oracle JDeveloper 10g no tiene incorporada ninguna herramienta para la generación de informes. Pero lo mismo te estás preguntando ¿qué queremos decir cuando hablamos de informe? Según la RAE (Real Academia Española) un informe es, entre otros significados, la **descripción, oral o escrita, de las características y circunstancias de un suceso o asunto**. Para nosotros un informe será un documento que contendrá información sobre el resultado de una o varias consultas a una Base de Datos, como puede ser un listado de empleados, el listado de las facturas pendientes de cobrar ordenadas por cantidad, un gráfico comparativo de ganancias de los años que lleva abierta la empresa, etc.

Los informes generalmente han sido presentados en papel, pero hoy día ha cobrado una gran importancia el informe electrónico, debido al avance de la tecnología y la conciencia de respeto al medio ambiente. El formato más extendido para presentar informes electrónicos es el formato PDF debido a su portabilidad entre sistemas conservando la apariencia. Pero existen muchos más: XLS, HTML, RTF, CSV, XML, etc.

En este apartado vamos a ayudar a Víctor a solucionar su problema de cómo generar informes en Oracle JDeveloper 10g. Para ello primero veremos una herramienta denominada JasperReport que nos va a permitir generar todo tipo de informes y cómo incluir dichos informes en nuestras aplicaciones. Seguidamente detallaremos el uso de otra herramienta llamada iReport que nos va a facilitar la generación de informes en formato entendible por JasperReport. Te preguntarás por qué son necesarias dos herramientas para hacer la misma tarea, pero conforme vayas avanzando por el apartado te darás cuenta que trabajar directamente con JasperReport es bastante tedioso, por lo que nos apoyaremos en iReport para hacernos la vida mucho más fácil.

Autoevaluación

Respecto a la generación de informes en Oracle JDeveloper 10g podemos afirmar...

a) Con Oracle JDeveloper 10g es imposible crear aplicaciones que generen informes
b) Con Oracle JDeveloper 10g podemos generar informes sólo si añadimos alguna librería que sea capaz de hacerlo o los genera nuestra aplicación "a pelo"
c) Oracle JDeveloper 10g incluye una herramienta muy poderosa para la generación de informes llamada Oracle Report
d) Podemos generar informes muy simples y sin gráficos por medio del menú "Report"

Comprobar

Trabajando con JasperReport

Víctor estaría impaciente por saber qué es JasperReport. ¿No te pasa a ti lo mismo? Veamos qué es JasperReport y cómo podemos utilizarlo en Oracle JDeveloper 10g para sacarle el mayor partido posible.

JasperReport es una herramienta que consiste en un poderoso motor para la generación de informes. Está empaqueta en un archivo JAR y puede ser utilizada como una librería, la cuál podemos integrar en Oracle JDeveloper 10g (y en cualquier otro IDE de desarrollo en Java) para desarrollar nuestras aplicaciones. Está escrita totalmente

en Java, su código está abierto y es totalmente gratuita bajo los términos de la licencia GPL.

JasperReport permite generar informes con contenidos muy variados y presentar dichos informes en pantalla, imprimirlos o exportarlos a los formatos PDF, XLS, HTML, RTF, ODT, XML y CSV. Además de ser una herramienta muy potente, tiene la ventaja de que se integra fácilmente con JfreeChart que es otra herramienta para la generación de gráficos de muy diversos tipos, por lo que su poder aumenta. Ya veremos más adelante cómo podemos llevar a cabo dicha integración.

En los siguientes subapartados veremos cómo y dónde podemos conseguir JasperReport. Una vez nos la hallamos descargado, profundizaremos en el funcionamiento de esta librería y veremos qué tenemos que hacer para que nuestras aplicaciones puedan hacer uso de ella.

Autoevaluación

Respecto a la licencia de uso de JasperReport podemos afirmar...

a) JasperReport está sujeto a la misma licencia que Oracle JDeveloper una vez lo integramos en el mismo.
b) Nos permite su uso para fines educativos pero no para uso lucrativo.
c) Permite que la usemos libremente bajo los términos de licencia GPL.
d) Podemos descargar una versión de evaluación válida sólo para 30 días.

Comprobar

Descarga de JasperReport

Como hemos dicho anteriormente la descarga de JasperReport es gratuita y lo único que nos pedirá es que nos registremos. Para este menester, nosotros ya nos hemos registrado por tí (aunque si quieres lo puedes hacer por tu cuenta). El usuario que hemos dado de alta es "**usuarioFPD**" y la contraseña "**claveFPD**". La versión que utilizaremos es la 2.0.2 que a día de hoy es la última estable.

ZONA DE DESCARGA

Si visitas este enlace podrás acceder a la página de descarga de JasperReport para todas las plataformas. Encontrarás una lista con todas las versiones disponibles, pero deberás identificarte para poder bajarte la que desees.

Zona de descarga de JasperReport

Al acceder al enlace encontrarás una página como la que mostramos a continuación y en la que te indicamos dónde identificarte y la versión a descargar. Recuerda que para poder descargarte JasperReport debes identificarte o no te dejará bajártelo y te dirá que no estás autorizado.

Al pinchar sobre la última versión recomendada, accederás a una página en la que al final de la misma puedes encontrar los diferentes ficheros disponibles para descargar. En esta imagen te mostramos esa lista y te indicamos cuál es el fichero a descargar. Elegimos este fichero ya que es el que contiene todas las librerías necesarias, además de la librería de JasperReport ya empaquetada en un fichero JAR.

Si descomprimimos el fichero que nos hemos descargado, veremos que en el mismo hay varios directorios o carpetas. Veamos qué contiene cada una:

1. **build**: Es la librería JasperReport pero sin empaquetar, con todas las clases que ésta incluye.
2. **demo**: Podemos encontrar algunos ejemplos de utilización de la librería. Estos ejemplos están preparados para ser compilados con la herramienta "ant". Puedes inspeccionar el código Java e incluso intentar compilarlos y ejecutarlos.
3. **dist**: Aquí se encuentra realmente la librería empaquetada en un fichero JAR (jasperreport-2.0.2.jar) y algunos ficheros JAR que no utilizaremos. También podemos acceder a la documentación tipo javadoc.
4. **docs**: Referencia rápida en formato XML.
5. **lib**: Diferentes librerías necesarias por JasperReport, como algunas para exportar a distintos formatos, para incluir gráficos, ... Estas librerías también deberemos integrarlas en nuestro IDE si queremos utilizar alguna de esas características. Ya veremos cuáles son.

6. **src**: Ficheros fuente de la librería.

Como ya hemos dicho anteriormente, JasperReport es una librería que nos proporciona el motor para la generación de informes, pero que no incluye nada más (no incluye por ejemplo un interfaz gráfico para generar dichos informes, imprimirlos, ...). Por tanto lo que nos queda hacer con esta librería es saber cómo podemos integrarla en nuestro IDE y saber cuál es su funcionamiento para posteriormente poder utilizarla en nuestras propias aplicaciones. ¡Vamos a ello!

Autoevaluación

Al descargarnos JasperReport podemos encontrar...

a) El fichero JAR correspondiente al motor de generación de informes.
b) Otros ficheros JAR necesarios en nuestras aplicaciones para la generación de informes
c) Una variedad bastante amplia de ejemplos
d) Todas las respuestas anteriores son correctas

Comprobar

Generación de informes y ayuda en línea

Integración de JasperReport en nuestros proyectos.

Te estarás preguntando: ¿Y ahora qué hacemos con todas esas librerías o ficheros JAR? ¿Qué debemos hacer para poder utilizarlas en nuestros proyectos?

Eso es lo que vamos a ver en este subapartado y verás cómo no es nada del otro mundo.

Las librerías que vamos a necesitar para incluir en nuestros proyectos serán las que mostramos a continuación. Hemos añadido una columna especificando el directorio en el que se encuentra cada librería dentro del directorio donde hayamos descomprimido JasperReport.

Nombre	Descripción	Directorio
jasperreports-2.0.2.jar	Contiene el motor de generación de informes.	dist
commons-beanutils-1.7.jar	Agiliza el uso de JavaBeans.	lib
commons-collections-2.1.jar	Extiende las colecciones que ofrece el JDK.	lib
commons-digester-1.7.jar	Ayuda a procesar documentos XML.	lib
commons-logging-1.0.2.jar	Facilita la escritura de log.	lib
itext-1.3.1.jar	Permite generar documentos PDF.	lib
poi-3.0.1-FINAL-20070705.jar	Permite escribir documentos XLS.	lib
jfreechart-1.0.0.jar	Motor para generar todo tipo de gráficos.	lib
jcommon-1.0.0.jar	Librería utilizada por JFreeChart.	lib

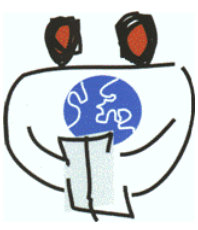
Decir que para generar informes necesitaremos una conexión a una Base de Datos y la librería que contiene los driver de dicha conexión también debemos añadirla a nuestro proyecto. En nuestro caso nos conectamos a una Base de Datos Oracle Express Edition, por lo que necesitamos los driver de conexión a una Base de Datos Oracle. Estos driver los podemos encontrar integrados en nuestro IDE como librería de sistema con el nombre "**Oracle JDBC**" o como fichero JAR que podemos encontrar en el directorio **`<jdev_home>/jdbc/lib/`** con el nombre **`ojdbc14.jar`**.

Una librería es un conjunto de clases (o archivos JAR) que nos proporcionan una funcionalidad concreta. En JDeveloper podemos encontrar tres tipos de librerías:

1. **Librerías de sistema**: Son librerías accesibles por todos los usuarios que utilizan el entorno y nosotros no las podemos crear. Éstas se crean al instalar JDeveloper y por medio de actualizaciones en forma de extensiones.
2. **Librerías de usuario**: Son librerías a las que sólo tiene acceso el usuario que las creó y las puede usar en todos los proyectos que éste realice. Las entradas que componen una librería de usuario tendrán una URL para localizar el archivo que representa dicha entrada. Las entradas de una librería de usuario suelen estar almacenadas en alguna carpeta del espacio de trabajo del usuario.
3. **Librerías de proyecto**: Son librerías que sólo están accesibles por el proyecto en el que están definidas. Las entradas que componen una librería de proyecto suelen estar almacenadas en alguna carpeta dentro del proyecto.

Para **integrar JasperReport en nuestro IDE y poder utilizarlo en nuestra aplicación** deberemos añadir las librerías antes mencionadas a nuestro proyecto y eso lo haremos en cada proyecto en el que necesitemos generar algún tipo de informe. Para hacer esto existen varias opciones:

1. **Añadirlas como una librería de usuario al entorno**: Creamos una nueva librería de usuario (recuerda que eso lo hicimos para instalar nuestros JavaBeans de la unidad 2), para lo cuál elegimos la opción "**Tools | Manage Libraries**". Pulsamos sobre el botón "**New**" para crear una nueva librería. Como nombre le ponemos "**JasperReport-2.0.2**" y elegimos añadirla a la categoría "**User**" y en ella incluiremos todos los ficheros JAR mencionados antes por medio del botón "**Add Entry**", buscándolos en el directorio donde hayamos descomprimido JasperReport y cada uno en la carpeta adecuada. Una vez hecho esto, nuestra librería (que no es más que varios ficheros JAR agrupados) ya está integrada en el IDE pero no en nuestro proyecto. Para ello en las propiedades de nuestro proyecto, a las que podemos acceder eligiendo la opción "**Project Properties**" del menú contextual que aparece al pulsar con el botón derecho del ratón sobre nuestro proyecto, en la entrada "**Libraries**" debemos elegir "**Add Library**" y buscamos la librería recién creada. Puedes ver el proceso en la siguiente demostración, en la que también se añade al proyecto la librería de sistema "**Oracle JDBC**".



Mira cómo añadir las librerías como Librerías de usuario

2. **Añadirlas como una librería de proyecto al entorno**: Primero creamos una carpeta a nivel de proyecto llamada "**lib**" y en ella copiamos todos los archivos JAR necesarios para el correcto funcionamiento de JasperReport. Elegimos la opción "**Project Properties**" del menú contextual que nos aparece al pulsar con el botón derecho del ratón sobre nuestro proyecto, y en la entrada "**Libraries**" elegimos "**New**" para crear una nueva librería de proyecto. Ahora creamos la librería siguiendo el mismo proceso descrito en el punto

anterior, pero con la única diferencia que en vez de añadirla a la categoría "**User**" la añadiremos a la categoría "**Project**" y que las entradas debemos buscarlas todas en la carpeta "**lib**" que hemos creado. Una vez hecho esto, esta librería ya estará integrada en nuestro IDE a nivel de proyecto y añadida a nuestro proyecto. En la siguiente demostración puedes ver el proceso que hemos seguido.

Mira cómo añadir las librerías como librería de proyecto

3. **Añadirlas como ficheros JAR independientes a nuestro proyecto**: Primero creamos una carpeta a nivel de proyecto llamada "**lib**" y en ella copiamos todos los archivos JAR necesarios para el correcto funcionamiento de JasperReport y el fichero " **`ojdb14.jar"`** necesario para conectarnos a Bases de Datos Oracle. Seguidamente en las propiedades de nuestro proyecto, en la entrada "**Libraries**" debemos elegir "**Add Jar / Directory**" e ir añadiendo uno a uno los ficheros JAR necesarios. Esta es la mejor opción si queremos llevarnos nuestra aplicación a otra máquina para abrirla con JDeveloper, ya que todo lo que nuestro proyecto utiliza y todas sus referencias están bajo el directorio de nuestro proyecto, por lo que lo único que debemos hacer es copiar todo el directorio de nuestra aplicación a otra máquina y decirle a JDeveloper que queremos abrir esa aplicación. Esta opción será la que deberás utilizar para entregar la tarea de esta unidad. La siguiente demostración muestra cómo llevar a cabo esta tarea (partiendo de que ya hemos creado la carpeta "**lib**" a nivel de nuestro proyecto) y en ella también añadimos el fichero JAR que contiene los drivers JDBC para conectarse a una Base de datos Oracle (que anteriormente hemos copiado a la carpeta "**lib**" de nuestro proyecto), en vez de utilizar la librería de sistema como mostramos en los dos puntos anteriores.

Mira cómo añadir las librerías como ficheros JAR independientes

Ya estamos preparados para comprender el funcionamiento de JasperReport para posteriormente poder utilizarlo en nuestras aplicaciones, ya que el entorno ya lo tenemos preparado. Como puedes ver, la tarea de integrar librerías en nuestro entorno es bastante sencilla ¿no?

Autoevaluación

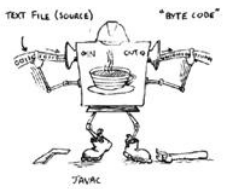
Para integrar JasperReport en el entorno debemos:

a) Descargar un instalador dependiente de la plataforma de trabajo
b) Descargar un instalador que sólo depende del IDE en el que lo queremos integrar.
c) Añadir los ficheros JAR necesarios a nuestro proyecto.
d) Todas las respuestas anteriores son falsas

Comprobar

Generación de informes y ayuda en línea

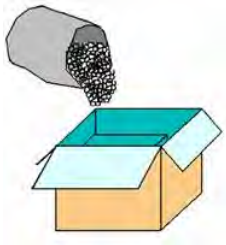
Funcionamiento de JasperReport



Seguramente a estas alturas tendrás bastante claro el funcionamiento del compilador javac: partimos de un archivo en texto plano escrito siguiendo las directrices del lenguaje Java con extensión .java y genera (si no hay errores de compilación) un archivo de bytecodes con extensión .class entendible por la máquina virtual de Java. Te estarás preguntando ¿Y a cuento de qué viene todo esto? ¿Qué tiene esto que ver con JasperReport, que es el punto que estamos tratando? Pues vamos a ver que tiene mucho que ver, ya que el funcionamiento de JasperReport se asemeja bastante a este proceso de compilación.

Efectivamente, el funcionamiento de JasperReport es muy parecido al funcionamiento de cualquier compilador. Veamos cuáles son los pasos seguidos en la generación de un informe, desde que lo diseñamos hasta que lo visualizamos o exportamos al formato deseado:

1. **Diseño del informe**: Diseñamos el informe escribiendo un fichero plano en formato XML (que suele tener la extensión . **`jrxml`**) utilizando las etiquetas y atributos especificados en el diccionario llamado **`jasperreport.dtd`** que puedes encontrar en la siguiente URL:
   http://jasperreports.sourceforge.net/dtds/jasperreport.dtd
   (Si te da problemas al intentar abrirlo desde Internet Explorer, prueba a abrirlo con Firefox, o bien desde la misma ventana del Internet Explorer que te indica el fallo, elige Archivo-Guardar Como, elige el nombre y dónde guardarlo, y ábrelo más tarde con un editor de texto).
   Por medio de XML definimos completamente el informe, especificando dónde estarán situados los distintos objetos que utilicemos en el informe (texto, imágenes, líneas, rectángulos, etc.), el aspecto que tendrán estos objetos, la consulta que haremos a la Base de Datos, cómo realizar ciertos cálculos para mostrar totales, etc.
2. **Compilación del informe**: El informe debe ser compilado antes de que podamos pasarlo al motor de generación de informes. La compilación del informe dará como resultado un **archivo Jasper** (con extensión **`.jasper`**) que aún no será entendible por el motor de generación de informes, ya que le faltarán los datos dinámicos procedentes de parámetros en tiempo de ejecución y de la consulta a nuestra Base de Datos.
3. **Relleno del informe**: Para poder mostrar (o exportar) el archivo Jasper debemos rellenarlo con los datos dinámicos que produzca la consulta a la Base de Datos y con algunos parámetros que se le pueden pasar al archivo Jasper en tiempo de ejecución y que modificarán el aspecto de nuestro informe. Una vez hemos rellenado el archivo Jasper, obtenemos un **archivo Print** el cuál si será entendible por el motor de generación de informes. Este archivo Print lo podremos visualizar en una aplicación que utilice Swing o exportar a diferentes formatos (PDF, HTML, XML, CSV, XLS, RTF, ODT). Estos documentos exportados también podremos visualizarlos en una página JSP, mostrarlos en un servlet, etc...



Como puedes observar, el funcionamiento de JasperReport es bastante simple. En el siguiente apartado veremos qué clases nos ofrece la librería JasperReport para llevar a cabo las distintas tareas que debemos realizar hasta visualizar o exportar nuestros informes.

Autoevaluación

Respecto a los informes generados por JasperReport podemos afirmar:

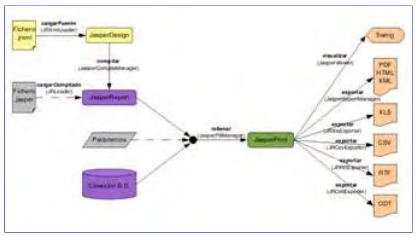
a) Podemos visualizarlos en una aplicación que utilice Swing
b) Podemos exportarlos al formato de Microsoft Word
c) Debemos exportarlos primero a formato PDF
d) Todas son verdaderas

Comprobar

Utilización de JasperReport en nuestras aplicaciones

Ya tenemos nuestro proyecto preparado con las librerías necesarias para generar informes. También tenemos claro las distintas tareas que debemos llevar a cabo para visualizar o exportar los informes que generemos. Sólo nos queda saber por medio de qué clases podemos realizar las distintas tareas y poner en práctica todo esto con un ejemplo que termine de aclararnos todo el proceso.

En la siguiente imagen puedes ver cuál puede ser el proceso desde la generación del archivo .jrxml hasta su visualización o exportación. Las flechas discontinuas significan que ese paso no es necesario, pero que se puede llevar a cabo.



Como puedes observar en la imagen, partimos de un fichero **`.jrxml`**, lo cargamos y obtenemos un objeto **`JasperDesign`**. Al compilar dicho objeto, obtenemos un objeto **`JasperReport`**. Este objeto, junto a los parámetros (si es que hay) y la conexión a la Base de Datos nos permiten rellenar el informe y generar un objeto **`JasperPrint`** el cuál será el que visualizaremos o exportaremos.

También podemos observar en la imagen anterior, que es posible cargar un fichero ya compilado y obtener directamente un objeto **`JasperReport`**. Esto no es necesario si partimos del archivo **`.jrxml`**, por eso está en la imagen con flecha discontinua.

Como puedes ver en la imagen las flechas van etiquetadas con un texto entre paréntesis que indica cuál es la clase que nos permite llevar a cabo la acción que denota. También vemos que las etiquetas empiezan con un texto en negrita que además de describir la acción que estamos realizando, indican el método que realiza dicha acción en una clase que hemos implementado para que puedas comprender mejor todo el proceso.

La clase que hemos implementado se llama **`Informe`**. Te recomendamos encarecidamente que eches un vistazo a la clase y que intentes comprender su comportamiento, ya que en esta clase nos apoyaremos para realizar los ejemplos que te propondremos.

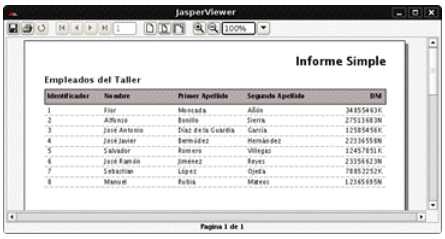
Descarga la clase Informe

Como has podido observar la clase **`Informe`** nos abstrae de la complejidad que pueda conllevar trabajar con informes (aunque es poca, pero un poco tediosa debido a la generación de excepciones que debemos capturar) y hace muy fácil el uso de JasperReport. Por tanto ya podemos llevar a cabo todo el proceso y generar un informe simple para comprobar las bondades de JasperReport ¿no?

Pues no, falta un paso importante y que no hemos mencionado cómo llevar a cabo. Ese paso es el primero de todos, se trata del diseño del informe para generar el fichero **`.jrxml`**. Por ahora te vamos a pedir que hagas un acto de fe y te creas que sabemos generar ese fichero, por lo que para seguir con nuestro ejemplo y poder hacer una prueba te vamos a suministrar un archivo **`.jrxml`** que debes pensar que ya sabes generar. Más adelante veremos que es MUY tedioso generar a mano dicho archivo y que para ello nos apoyaremos en una herramienta gráfica que nos hace mucho más fácil la vida. De todas formas échale un vistazo a dicho archivo y no se te ocurra asustarte. El informe muestra un listado de los empleados del taller "El PC Feliz", mostrando para cada empleado su identificador, nombre, apellidos y DNI.

Descarga el archivo

El resultado de visualizar este informe es el siguiente. Haz clic sobre la imagen para ampliarlo.



Pues ya ha llegado la hora de crear una aplicación que nos permita ver y exportar el informe suministrado. En el siguiente subapartado crearemos una aplicación que utilizaremos como caso de estudio e iremos engordando poco a poco conforme vayamos adquiriendo nuevos conocimientos a lo largo de esta unidad.

> ***PARA SABER MÁS***
>
> ***En el siguiente enlace podrás encontrar un tutorial de cómo crear una página JSP que visualice un informe. La página está en inglés, pero es bastante sencilla su lectura, ya que se acompaña de imágenes explicativas.***
>
> ***Creación de un informe en una página JSP con JasperReport*** [Versión en caché]

Autoevaluación

Respecto al funcionamiento de JasperReport podemos afirmar que...

a) Podemos partir de un informe ya relleno que tiene la extensión .print
b) Es posible partir de un informe ya compilado con extensión .jasper
c) Siempre partimos de un informe fuente con extensión .jrxml
d) Todas las respuestas anteriores son falsas

Comprobar

Creación de nuestro caso de estudio: Comenzando

Para nuestro caso de estudio seguiremos con el ejemplo del Taller Informático "El PC Feliz". Diseñaremos una aplicación cliente que genere el informe simple que hemos visto en el subapartado anterior y que nos habíamos creído que sabíamos generar. Podíamos haber continuado con la aplicación **"GestionTaller"** que habíamos empezado en la unidad anterior, pero hemos decidido hacer una aplicación aparte para que así podamos ver otra forma diferente de llevar a cabo algunas tareas (unas se vieron en la unidad anterior y otras son nuevas).

Hemos creado una aplicación llamada **"PcFeliz"** y en ella hemos creado un proyecto llamado **"Informes"** y en el mismo incluiremos nuestra aplicación cliente que nos muestre el informe simple visto anteriormente (pero que poco a poco nos ofrecerá más tipos de informes). No hemos seguido el patrón MVC para la realización de nuestra aplicación, ya que el modelo es gestionado por la librería JasperReport, por lo que todo estaría en la vista. El diseño de clases si está pensado para utilizar dicho patrón y sería trivial dividir nuestra aplicación en vista y controlador.

La aplicación comenzará lanzando una ventana tipo "splash". Seguidamente nos pedirá que nos identifiquemos con las credenciales de la Base de Datos. Una vez la identificación sea correcta nos aparecerá la interfaz de nuestra aplicación.

En el siguiente enlace te puedes descargar el archivo comprimido perteneciente al directorio a nivel de aplicación, por lo que lo único que debes hacer es descomprimirlo y decirle a JDeveloper que quieres abrir el archivo **PcFeliz.jws** que se encuentra dentro del directorio.

Descarga el archivo

Fíjate que el directorio del proyecto, contiene en el directorio **src** un directorio llamado **recursos** y en él dos directorios llamados **informes** e **imágenes**. En el primero hemos colocado nuestro archivo **.jrxml** y en el segundo está el logo del taller, la imagen de inicio y algunos iconos que hemos utilizado. Si miras el código verás que a la hora de cargar dichos archivos lo hacemos obteniendo su URL en tiempo de ejecución por medio del método **getResource** o **getResourceAsStream**. Esto es muy importante, ya que permitirá que al empaquetar nuestra aplicación en un archivo JAR, los caminos hacia dichos archivos sean válidos y si ejecutamos nuestra aplicación fuera del entorno, también sean caminos válidos.

En verdad los recursos que hemos mencionado anteriormente no deben estar bajo el directorio **src**, sino bajo el directorio **classes** (pero si los situamos en el directorio **src** podemos verlos en el entorno). Para que todo funcione correctamente, JDeveloper en el proceso de compilación copia los ficheros incluidos en el directorio **src** y que estén en la lista de extensiones que debe copiar, al directorio **classes** que es el directorio donde deberían estar situados.

En la siguiente demostración podemos ver cómo incluir la extensión **.jrxml** en las extensiones que debe copiar en el proceso de compilación y también le diremos que la extensión **.jrxml** es un fichero de tipo XML para que lo visualice correctamente en el entorno.

Mira cómo incluir la extensión para que se visualice correctamente

Creación de nuestro caso de estudio: Clases que lo forman

¿Qué nos queda por comentar sobre nuestro caso de estudio? Como imaginas, es importante saber qué clases lo van a componer, y qué utilidad tendrá cada una de ellas.

Por último vamos a comentar las distintas clases que forman nuestro proyecto:

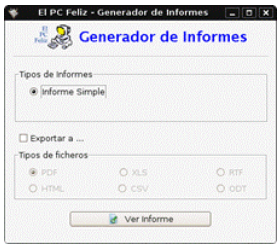
1. **BD**: Esta clase es la encargada de abrir una conexión con la BD con el usuario y contraseña que se pasan como parámetros (**abirConexion**), proporcionar la conexión que hemos abierto (**getConexion**) y cerrar dicha conexión (**cerrarConexion**). Todos los métodos son métodos de clase por lo que no necesitamos una instancia de dicha clase para poder utilizarla.
2. **Utilidades:** Clase auxiliar que tiene un método que centra un contenedor en la pantalla (**centrar**) y otro que hace una pausa (**pausa**) de tantos milisegundos como indique el parámetro.
3. **PoliticaFocos:** Clase que hereda de **FocusTraversalPolicy** y que es necesaria para poder indicar el orden de los componentes que tendrán el foco dentro de una ventana (la propiedad **nextFocusableComponent** está descatalogada (deprecated)). Lo único que hace esta clase es sobrescribir los métodos abstractos de la clase de la que hereda. Tiene un vector de componentes que indica el orden a seguir por el foco. Los métodos devuelven un componente u otro basándose en el orden de este vector. La hemos utilizado en las clases **DialogoLogin** y **GUIGeneradorInformes**.
4. **Informe:** Esta clase ya la vimos en el apartado anterior y es la encargada de abstraernos de la complejidad que puede tener el trabajar con las clases de la librería JasperReport.
5. **DialogoEspera:** Clase que hereda de **JDialog** y que simplemente está compuesta por una etiqueta que nos dice que esperemos con paciencia y una barra de progreso que ponemos en modo indeterminado y que nos servirá para indicar al usuario que se está realizando una operación que puede consumir bastante tiempo.

6. **DialogoInicio:** Clase que muestra una ventana tipo "splash", aunque la hemos implementado

heredando de **JDialog**. Consta de una etiqueta en la que pondremos la imagen y una barra de progreso.
Tiene un método **actualizaEstado** que cambia el progreso de la barra y la etiqueta mostrada. Este método llama al método **invokeLater** de la clase **SwingUtilities**, que nos permite planificar un trabajo (clase que hereda de la clase **Runnable**) para que sea ejecutado por el EDT (Event Dispatch Thread - Hebra de despacho de eventos). El trabajo será la actualización del progreso y de la etiqueta en la barra de progreso.

7. **DialogoLogin**: Dialogo que nos pide el usuario y la contraseña de la BD e intenta conectarse a la misma mostrando el resultado de dicha operación (éxito, error, etc.). En esta clase puedes ver cómo usar el control **JPasswordField**. Mientras se intenta conectar a la BD (tarea que puede tardar su tiempo) mostramos el **DialogoEspera** para indicar que se está realizando dicha operación. Esto ocurrirá como respuesta a un evento (pulsar el botón aceptar o pulsar Enter sobre el campo password). Un programa Swing está compuesto por una sola hebra que es la EDT, por lo que el repintado de dicho diálogo no se llevará a cabo hasta que no haya terminado el manejador de dicho evento. Como la tarea puede tardar, el efecto estético no es el deseado. Para solucionar este problema hemos creado una hebra que ejecuta el trabajo pesado y así el manejador termina inmediatamente, se repinta el **DialogoEspera** y en segundo plano se está realizando el trabajo pesado de validación.

8. **GUIGeneradorInformes**: Es la interfaz de nuestra aplicación. Esta clase es la que lanza el **DialogoInicio** y conforme se va cargando va llamando al método **actualizaEstado** de dicha clase (hacemos pausas para que sea perceptible la carga, aunque esto no es necesario sino que sólo es por estética ya que nuestra interfaz tarda muy poco en cargarse). Cuando termina la carga destruimos dicho diálogo y mostramos el **DialogoLogin**. Si nos validamos correctamente regresaremos a nuestra interfaz (en caso contrario saldremos de la aplicación). Una vez en la interfaz esperaremos a que se pulse el botón Ver (o Ver / Exportar) que nos mostrará (y en su caso exportará al tipo elegido) nuestro informeSimple (posteriormente iremos añadiendo más tipos de informes). Al pulsar el botón se ejecuta el manejador de dicho evento y nos encontramos con el mismo problema que antes, ya que queremos mostrar el **DialogoEspera** y ejecutar el método **verExportarInforme**, por lo que este método lo ejecutaremos en una nueva hebra. El método **verExportarInforme** es el encargado de llevar a cabo todo el proceso de generación de un informe: carga del informe, compilación, relleno, visualización y exportación si es necesaria. Este método hace uso de la clase **Informe**.



9. **GeneradorInformes**: Clase principal que es la encargada de lanzar la interfaz gráfica desde su método **main**.

Ya sabemos como ejecutar nuestra aplicación desde el entorno. Pero ¿y si la queremos ejecutar fuera de éste o en otro ordenador? Lo más cómodo es empaquetar nuestra aplicación en un archivo JAR. En el siguiente apartado veremos cómo podemos hacerlo ya que nuestra aplicación depende de unas librerías necesarias para su ejecución.

Autoevaluación

Nuestro caso de estudio es capaz de:

a) Mostrar una ventana tipo "splash" cuando se está cargando la GUI.
b) Definir su propia política de focos en algunas ventanas o diálogos
c) Indicar que se está realizando una tarea pesada por medio de un diálogo de espera
d) Todas las respuestas anteriores son correctas

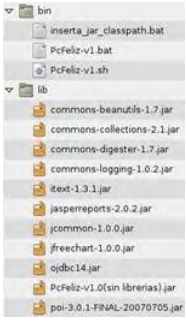
Comprobar

Despliegue de aplicaciones que dependen de librerías (I)

En la unidad 2 ya vimos cómo desplegar nuestras aplicaciones a ficheros JAR y pudimos comprobar lo fácil que era. En dicha unidad desplegamos una aplicación simple y todo funcionó correctamente. Pero si intentaste desplegar la aplicación del factorial que hacía uso del JavaBean que proponíamos, te darías cuenta que al intentar ejecutar el fichero JAR nos daba un error debido a que no encontraba la clase del JavaBean. ¿Por qué ocurre esto? Esto ocurre sencillamente porque el JavaBean era una librería (que JDeveloper incluyó en nuestro proyecto por el mero hecho de arrastrar desde la paleta de componentes el JavaBean), y nuestra aplicación depende de dicha librería para ser ejecutada.

Al igual que en el caso del JavaBean, cuando nuestro proyecto depende de librerías como es el caso de la generación de informes y la conexión a una Base de Datos, debemos tener claro cómo vamos a desplegar nuestra aplicación. Para resolver este problema tenemos dos opciones (partimos de que la Base de Datos está creada y operativa en la máquina en la que despleguemos nuestra aplicación):

1. **Desplegamos la aplicación en un archivo comprimido que debemos descomprimir para luego ejecutar un script que lance nuestra aplicación.** Al descomprimir la aplicación se creará un directorio al que nos referiremos como **<ap_home>**. Debe existir un directorio en el cuál estarán todas las librerías (ficheros JAR) de las que dependa nuestra aplicación (p.e. **<ap_home>/lib**). Debe existir un archivo JAR con nuestra aplicación tal cual explicamos en la unidad 2 (es decir, sin las librerías de las que depende nuestra aplicación), el cual podría estar situado en el directorio donde están situadas las librerías (**<ap_home>/lib**). Debe existir un script que lance nuestra aplicación y que generalmente está situado en el directorio **<ap_home>/bin**. Este script lo único que hará será ejecutar la máquina virtual de Java indicándole donde están las librerías por medio de la opción **"-classpath"**. La estructura de directorios podría ser la que mostramos en la imagen, en la que puedes observar que hay tres script: dos para Windows con extensión **.bat** (uno que lanza la aplicación y otro que es utilizado por el primero) y uno para Linux con extensión **.sh**. También podemos observar como hemos incluido en el directorio **lib** el fichero JAR de nuestra aplicación (**PcFeliz-v1.0(sin librerias).jar**). Ahora simplemente debemos comprimir esa estructura en un fichero que será el que distribuiremos y el usuario simplemente deberá descomprimirlo y ejecutar el script para su plataforma. Te facilitamos el fichero comprimido que desplegaría nuestra aplicación para que lo descomprimas y ejecutes el script adecuado para lanzar la aplicación.



Descarga el fichero comprimido

2. **Desplegamos un sólo archivo JAR que será el que ejecutaremos**. Esta es la opción más elegante, ya que el usuario sólo ve un archivo que puede ejecutar sin realizar ningún paso más. Para ello a la hora de generar el archivo JAR deberemos decirle a JDeveloper que incluya en el mismo las librerías de las que depende nuestra aplicación. Para ello seguimos el mismo proceso que seguimos en la unidad 2 para generar nuestro fichero JAR, pero antes de

terminar debemos añadir un grupo de ficheros por medio del botón "**New**", y elegimos la opción "**Dependency Analysis**". Como nombre le podemos poner "**Dependencias**". En el nuevo grupo creado, en la categoría "**Contributors**" elegimos la pestaña "**Libraries**" y veremos que en ella aparecen las librerías que hemos incluido en nuestro proyecto. Sólo nos queda marcarlas todas y asegurarnos que está elegida la opción "**Include Contents in Output**", que es la opción por defecto. Una vez hecho esto aceptamos y sólo nos queda desplegar el fichero JAR como ya sabemos. Al hacer esto lo que hacemos es incluir en nuestro fichero JAR además de las clases de nuestra aplicación, las clases de las librerías de las que depende nuestra aplicación. Podemos observar cómo nuestro fichero JAR ahora es mucho más grande que antes, ya que contiene las librerías necesarias. El usuario simplemente tiene que ejecutar el archivo mediante la orden: `java -jar PcFeliz-v1.0(con librerias).jar` (en Windows también podemos hacer doble clic sobre el archivo para ejecutarlo). En la siguiente demostración puedes ver cómo hemos creado el fichero JAR (en la demostración lo hemos llamado `PcFeliz-v1.jar`, pero para no liarnos con los nombres hemos decidido renombrarlo a `PcFeliz-v1.0(con librerias).jar`). También te facilitamos el fichero JAR creado (y ya renombrado) para que puedas comprobar cómo puedes ejecutarlo directamente y no da ningún error debido a que no encuentra clases pertenecientes a las librerías.

Creación del archivo JAR

Descarga el archivo

Despliegue de aplicaciones que dependen de librerías (II)

Además del problema de las librerías de las que depende nuestro proyecto, está el problema de los caminos utilizados en nuestro proyecto para referirnos a recursos (imágenes, iconos, ficheros necesarios por nuestra aplicación, etc.). Ya hemos comentado anteriormente que nosotros situábamos los recursos bajo el directorio `src` y que luego al compilar (y haciendo los cambios que comentamos para que incluya el fichero `.jrxml`) se copiaban al directorio `classes`. Al situarlos ahí nos aseguramos que éstos se incluyen en el fichero JAR que creemos.

Para acceder a los recursos hemos utilizado los métodos `getResource` y `getResourceAsStream` que devuelven una URL válida cuando dichos recursos están dentro de un fichero JAR (también es válida cuando los caminos a los que hacemos referencia están en el mismo directorio desde donde ejecutamos nuestra aplicación, como es el caso de la ejecución desde el entorno de JDeveloper).

Ya tenemos claro cómo funciona JasperReport y cómo podemos integrarlo en nuestro entorno de trabajo. También hemos visto cómo podemos crear aplicaciones que hacen uso de JasperReport y cómo desplegar dichas aplicaciones. Sólo nos queda saber cómo generar los ficheros `.jrxml` que darán lugar a los informes propiamente dichos. En el siguiente apartado veremos cómo crear dichos ficheros utilizando un entorno visual y las posibilidades que tenemos a la hora de diseñar nuestros informes.

Autoevaluación

Para integrar las librerías de las que depende nuestro proyecto en un archivo JAR debemos...

a) Utilizar una herramienta que nos permita hacer esto, como puede ser "MakeJAR"
b) No es posible incluir librerías dentro de un fichero JAR
c) Añadir un nuevo grupo de ficheros al perfil de despliegue que permita definir las dependencias de nuestra aplicación.
d) Empaquetar todas las librerías de las que depende nuestra aplicación en un fichero .dll

Comprobar

Diseñando informes con iReport

Ya debes tener claro cómo podemos visualizar y exportar informes utilizando la librería JasperReport. Lo que seguramente no te convence mucho es ese "acto de fe" que te pedimos que hicieses y que consistía en creerte que sabías escribir ficheros `.jrxml` a partir de los cuales generar los informes que necesites. Si echaste un vistazo al fichero `InformeSimple.jrxml` que te proporcionamos en los apartados anteriores y que generaba un informe muy simple con el listado de los empleados del taller informático "**El Pc Feliz**" (es decir, una sentencia SELECT de lo más trivial), su escritura es como hacer "encaje de bolillos" debido a lo complejo que parece.

Pero lo prometido es deuda y como te dijimos, la tarea de diseñar informes se hace mucho más simple si utilizamos la herramienta adecuada. Al igual que apretar un tornillo es casi imposible si lo queremos hacer a mano (aunque siempre hay "brutos" capaces de hacerlo), es una tarea muy simple si utilizamos la herramienta adecuada como puede ser un destornillador y aún más simple si este destornillador es eléctrico. Ocurre exactamente lo mismo a la hora de diseñar informes, ya que si lo queremos hacer a mano la tarea puede llegar a hacerse imposible. Pero si utilizamos iReport la tarea es como apretar un tornillo con un destornillador eléctrico. Pues veamos qué es iReport ¿no te parece?

Podemos decir que iReport es una herramienta muy potente, intuitiva y de muy fácil manejo para diseñar informes de forma visual para JasperReport. Nos permite diseñar informes de lo más complejos utilizando gráficos, imágenes, subinformes y generar el archivo `.jrxml` correspondiente a dicho diseño. Al igual que JasperReport, está escrita íntegramente en Java y su código está abierto y libre bajo licencia GPL. Como ya dijimos, para la generación de gráficos utiliza la librería de código abierto líder en la generación de representaciones gráficas de datos que es JfreeChart. Las fuentes de datos pueden ser cualquier conexión JDBC, modelos de tablas, JavaBean, XML... Nosotros utilizaremos las conexiones JDBC.

En los siguientes apartados veremos cómo descargarnos la herramienta e iremos viendo las posibilidades que nos ofrece. Poco a poco y con muy poco esfuerzo seremos capaces de crear todo tipo de informes. Como al andar se hace camino, como decía el poeta, empecemos a dar los primeros pasos para lograr nuestro objetivo.

Autoevaluación

Respecto a la licencia de uso de iReport podemos afirmar...

a) Nos permite probar una versión que nos permite trabajar durante 5 minutos
b) Nos permite su uso para fines educativos, pero no para uso lucrativo
c) Permite que la usemos libremente bajo los términos de licencia GPL.
d) Podemos descargar una versión de evaluación válida sólo para 30 días.

Comprobar

Descarga e instalación de iReport

Como ya hemos comentado la descarga de iReport es gratuita (incluso podemos bajarnos el código fuente de la misma). A diferencia de JasperReport, para descargarnos iReport no tenemos que identificarnos por lo que tampoco debemos registrarnos para obtener un usuario y una contraseña. La última versión a día de finalización de esta unidad es la 2.0.2.

ZONA DE DESCARGA

Si visitas este enlace podrás acceder a la página de descarga de iReport para todas las plataformas. Encontrarás una lista con todas las versiones disponibles y debes elegir la última.

Zona de descarga de iReport

Al acceder al enlace encontrarás una página en la que al final de la misma puedes encontrar los diferentes ficheros disponibles para descargar. En esta imagen te mostramos esa lista y te indicamos cuál es el fichero a descargar. Al estar íntegramente escrita en Java el fichero a descargar es independiente de la plataforma en la que lo vayamos a utilizar.

Si descomprimes el fichero que nos hemos descargado, verás que en el mismo hay varios directorios o carpetas. Para ejecutar la herramienta deberás ejecutar el script apropiado para tu plataforma: **`iReport.bat`** para Windows o **`iReport.sh`** para Linux al cual debes darle permisos de ejecución a él y a otro script que es llamado por el mismo llamado **`startup.sh`** que se encuentra en el directorio **`bin`**. Estos scripts se encuentran en el directorio donde has descomprimido el archivo que te has descargado. Recuerda que siempre puedes crear un acceso directo en el caso de Windows o un lanzador en el caso de Linux en el escritorio para poder lanzar la aplicación haciendo doble clic sobre el lanzador o acceso directo.

Puedes ver cómo en el directorio donde has descomprimido iReport existe un directorio **`"lib"`** en el que se encuentran los ficheros JAR de las librerías que iReport utiliza. Entre estas librerías vienen algunos driver de conexión a BD, pero no se encuentran los driver de conexión a BD Oracle, por lo que copiaremos el fichero **`ojdbc14.jar`** (que como recordarás lo puedes encontrar en el directorio **`<jdev_home>/jdbc/lib/`**) a este directorio para que iReport pueda conectarse a BD Oracle (esto también lo puedes conseguir modificando el **`CLASSPATH`** de iReport).

Autoevaluación

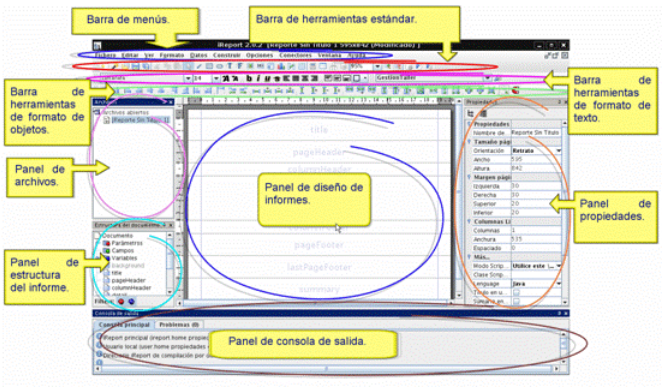
Respecto a la descarga iReport podemos afirmar...

a) Nos descargamos un fichero comprimido válido para todas las plataformas
b) Nos descargamos un fichero comprimido dependiente de nuestra plataforma
c) Sólo podemos descargarnos una versión para Windows ya que la de Linux está aún en fase de desarrollo
d) Nos descargamos un script de instalación dependiente de la plataforma y un fichero comprimido independiente de la plataforma

Comprobar

Familiarizándonos con el entorno (I)

Ya has descargado e instalado iReport y seguro que estás impaciente por lanzarlo, ver el aspecto que tiene el entorno de nuestra herramienta y comprobar si es verdad que es intuitivo y sencillo de utilizar, así como explotar la potencia del mismo. Para ello lo primero que debes hacer es lanzarlo y encontrarás una ventana como la que te mostramos a continuación (aunque nosotros te mostramos el entorno una vez hemos creado un informe nuevo en blanco) y en la que puedes encontrar varios elementos que seguidamente pasamos a comentar.



1. **Barra de menús**: Como en todas las aplicaciones, aquí puedes encontrar todas las funciones que realiza el programa agrupadas por funcionalidad. Poco a poco iremos viendo dichas funcionalidades.
2. **Barra de herramientas estándar**: Contiene atajos a las funciones más comunes a la hora de diseñar informes. Está dividida por funcionalidades: archivo, edición, objetos, ver y construir. La funcionalidad más usada es la referente a objetos, ya que posee todos los objetos que podemos añadir a nuestro informe.

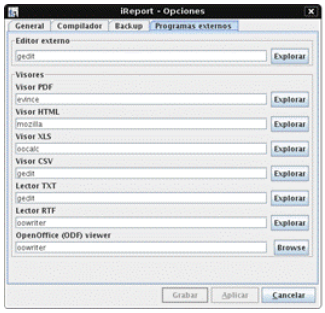
3. **Barra de herramientas de formato de texto**: En esta barra podemos encontrar atajos para formatear el texto asociado a los objetos: tipo de letra, tamaño, atributos, justificación horizontal, justificación vertical y bordes. Al final de esta barra hay otra barra que nos permite manipular las fuentes de datos.

4. **Barra de herramientas de formato de objetos**: Aquí están los atajos para dar formato a los objetos que hemos añadido al informe y también está dividida en funcionalidades: alineación, posición, tamaño, espaciado, ...

5. **Panel de archivos**: En este panel se muestran los ficheros que tenemos actualmente abiertos.
6. **Panel de estructura de informe**: Panel donde podemos visualizar el contenido de nuestro informe de una forma estructurada.
7. **Panel de consola de salida**: Este panel muestra la consola principal de nuestra herramienta donde se nos va informando de las acciones que iReport va realizando. También contiene una pestaña donde se muestran los problemas encontrados en el informe. Si intentamos compilar un informe se abrirá una tercera pestaña detallando el proceso de compilación.
8. **Panel de propiedades**: Aquí podemos ver y cambiar las propiedades del objeto que actualmente esté seleccionado. También podemos acceder a las propiedades de un objeto haciendo doble clic sobre el mismo o pulsando con el botón derecho sobre el objeto y eligiendo la opción "**Propiedades**" del menú contextual que aparece.
9. **Panel de diseño**: Panel en el que podemos ver nuestro informe de una manera más intuitiva y visual al estilo de los editores WYSIWYG (What you see is what you get - Lo que ves es lo que obtienes).

Familiarizándonos con el entorno (II)

Como puedes observar el entorno es bastante amigable y como pronto verás, sencillo de utilizar.



Antes de pasar a ver algunos conceptos básicos necesarios a la hora de diseñar un informe queremos que eches un vistazo a la opción "**Opciones...**" del menú "**Opciones**".

Como ves nos aparece un diálogo dividido en 4 pestañas de las que vamos a comentarte algunas opciones interesantes.

- La primera nos muestra opciones en general del entorno, como por ejemplo el lenguaje (aunque te adelantamos que la traducción al español deja mucho que desear como pronto verás) o la apariencia del entorno entre otras.
- La segunda es referente al compilador y entre las opciones que muestra la más interesante es la etiquetada como "**Usar directorio de informes para las compilaciones**" que si la activamos generará los informes compilados en el mismo directorio donde se encuentren los fuentes. Si esta casilla no está activada nos permite indicar el directorio donde se guardarán los informes compilados.
- La tercera permite configurar los archivos de respaldo (backup) de los informes sobre los que estamos trabajando.
- En la cuarta pestaña debemos indicar los programas externos utilizados para visualizar los distintos tipos de archivos a los que podemos exportar un informe. Si esta pestaña no está configurada podremos exportar informes a los formatos disponibles pero no podremos verlos. En la imagen de comienzo de este apartado puedes ver la configuración que hemos hecho para Linux (en Windows los programas cambiarán).

Otra tarea que debemos llevar a cabo antes de empezar a diseñar nuestros informes es configurar la conexión a la Base de Datos (o a otra fuente de datos, pero nosotros lo haremos a una BD y en concreto a una BD Oracle). Para ello en el menú "**Datos**" elegimos la opción "**Conexiones / Fuentes de datos**". Nos aparecerá un diálogo en el que podemos añadir una nueva conexión pulsando el botón "**Nuevo**" y seguidamente debemos elegir la fuente de datos que en nuestro caso es "**Conexión a Base Datos JDBC**". A continuación nos aparece un cuadro de diálogo en el que debemos configurar nuestra conexión (nombre que queremos poner a la conexión, controlador JDBC, URL JDBC, usuario y contraseña). La configuración que debemos realizar para conectarnos a la BD del taller informático "El Pc Feliz" es la que mostramos en la figura (suponiendo que la BD está en la misma máquina donde estamos trabajando).

Y comentado esto pasemos a ver los conceptos básicos en los que se basa iReport (que son conceptos en los que se basa JasperReport pero que no hemos visto ya que la parte de diseño nos la

saltamos) para diseñar informes.

Autoevaluación

El entorno de iReport está formado por:

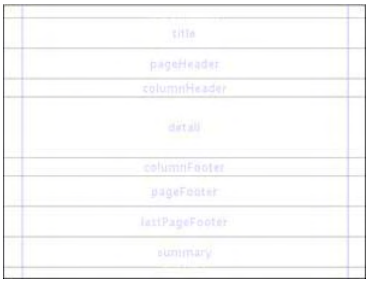
a) Una barra de menús con todas las funciones que nos ofrece iReport
b) Varias barras de herramientas con atajos a las funciones de iReport.
c) Varios paneles en los que se nos muestra información de distinta índole
d) Todas las respuestas anteriores son correctas

Comprobar

Conceptos básicos: Bandas

Ya hemos comentado que iReport es una herramienta muy intuitiva y bastante fácil de utilizar, pero para poder llegar a diseñar informes debemos tener claros algunos conceptos utilizados por esta herramienta (en verdad son conceptos utilizados por JasperReport ya que como hemos dicho iReport diseña informes para JasperReport). En cuanto tengamos claros dichos conceptos seremos capaces de diseñar todo tipo de informes. Pasemos a describir estos conceptos, comenzando por el de banda.

- **Una banda es cada una de las áreas en las que está dividido el diseño de nuestro informe.**
- **Un informe en iReport se diseña a nivel de página** y no podemos añadir elementos fuera del tamaño de la página.
- **Cada banda se comporta de una forma diferente**.



Las bandas en las que se divide una página son las que se muestran en la imagen que mostramos a continuación cuyo aspecto es el que tiene un nuevo informe en blanco.

1. **title**: Se mostrará sólo una vez al principio del informe, tenga las páginas que tenga el mismo. Es utilizada generalmente para poner el título del informe.
2. **pageHeader**: Es la cabecera de cada página. En ella podemos incluir por ejemplo el logo de nuestra organización que queremos que aparezca en todas las páginas.
3. **columnHeader**: Es la cabecera de las columnas. Se utiliza para mostrar los nombres de las columnas de las que constará nuestro informe.
4. **detail**: Es la encargada de mostrar los valores devueltos por una consulta a la Base de Datos (o a otra fuente de datos).
5. **columnFooter**: Aparece al final de cada columna (una vez por página) y suele utilizarse para presentar cálculos intermedios sobre las columnas, como sumas, promedios, ...
6. **pageFooter**: Pie de página, se repite una vez por página. Se suele utilizar para mostrar la paginación de nuestro informe, la fecha, ...
7. **lastPageFooter**: Se muestra sólo una vez al final del informe y se suele utilizar para calcular totales de las columnas mostradas en la banda detail.
8. **summary**: Sólo se repite una vez por informe en la última página del mismo y en ella mostraremos información resumen de nuestro informe. Se suele utilizar para mostrar gráficos o tablas de referencias cruzadas.

El orden de las bandas no se puede cambiar. Generalmente no utilizamos todas las bandas por lo que podemos poner su altura a 0 si queremos que una banda no se muestre y no ocupe espacio. Al usar grupos aparecen nuevas bandas que ya trataremos cuando hablemos de grupos. **Hay una banda especial que no se suele mostrar (su altura está a 0) y es la banda de fondo (background) que nos permite utilizar una imagen de fondo como marca de agua.**

Autoevaluación

Una banda en iReport podemos definirla como...

a) Los márgenes de nuestro informe
b) Una separación que aparece entre cada registro de la BD que presentamos en el informe
c) Cada una de las áreas en las que está divido el diseño de nuestro informe
d) Todas las respuestas anteriores son falsas

Comprobar

Conceptos básicos: Campos

¿Qué es un campo para iReport?

Los campos son los datos que recuperamos de la Base de Datos (o de cualquier otra fuente de datos). Aunque nosotros podemos gestionar (añadir, modificar y borrar) los campos accediendo a la opción "**Campos de Informe**" del menú "**Ver**" o pulsando sobre el icono y eligiendo la pestaña "**Campos**", eso no es lo normal.

Lo normal es que los campos se definan automáticamente al definir nuestra consulta a la Base de Datos. Los campos tomarán el nombre del atributo en la BD, a no ser que utilicemos la cláusula **"AS"** en la consulta. Para referirnos a un campo lo haremos usando la sintaxis **"$F{NOMBRE_CAMPO}"** (el nombre del campo irá en mayúsculas, ya que iReport lo convierte a mayúsculas). El tipo de un campo será una clase adecuada para almacenar el dominio de ese campo en la BD. Por ejemplo, si nosotros hemos definido la siguiente consulta a la Base de Datos de nuestro ejemplo del taller informático **"SELECT idempleado as id, nombre FROM empleados"**, automáticamente se crearán dos campos llamados: **ID** y **NOMBRE**. Para referirnos a ellos lo haremos usando los siguientes identificadores: **$F{ID}** y **$F{NOMBRE}**.

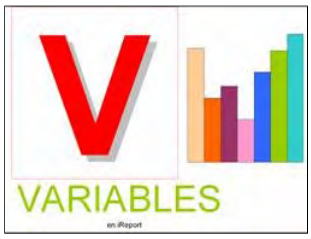
Autoevaluación

Respecto a los campos en iReport podemos afirmar que...

a) Podemos referirnos a ellos por medio de su nombre.
b) Podemos referirnos a ellos por medio de su nombre encerrado entre llaves y anteponiendo "$F".
c) Podemos utilizar ambas nomenclaturas, es decir, tanto por medio de su nombre, como por su nombre encerrado entre llaves y anteponiendo "$F".
d) Todas las respuestas anteriores son correctas

Comprobar

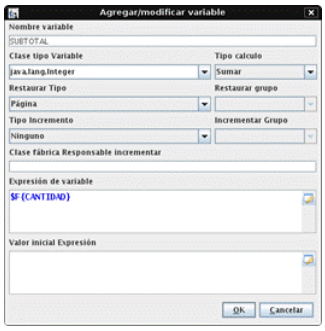
Conceptos básicos: Variables



¿Crees que el concepto de variable será muy distinto a lo que estamos acostumbrados a usar en otras aplicaciones?

Las variables almacenan valores asociados a una expresión o realizan cálculos sobre una expresión. Las expresiones se pueden construir utilizando expresiones java, variables, parámetros o campos y en realidad son expresiones Java por lo que podemos usar cualquier operador, método...

Se suelen utilizar para mostrar valores calculados en tiempo de ejecución en nuestro documento como calcular totales, hacer recuentos... Podemos acceder a las variables de nuestro informe mediante la opción "**Variables de Informe**" del menú "**Ver**" o pulsando sobre el icono y eligiendo la pestaña "**Variables**" del diálogo que nos aparece. Desde ahí podremos añadir, modificar o borrar variables pulsando sobre el botón adecuado. Para añadir una varibale debemos definir:



1. **Nombre variable**: su nombre.
2. **Clase tipo variable**: la clase que almacenará el valor de la variable.
3. **Tipo cálculo**: si queremos utilizar un cálculo predefinido sobre un campo indicaremos el tipo de cálculo a realizar (recuento, suma, promedio, etc.). De lo contrario elegiremos "**Nada**".
4. **Restaurar tipo**: cuándo se inicializará la expresión a su valor inicial (no se inicializa, una vez al principio del informe, al principio de cada página, al principio de cada columna o cuando el grupo especificado en "**Restaurar grupo**" cambie).
5. **Restaurar grupo**: el grupo que debe cambiar para que se inicialice la variable cuando hemos elegido "**Grupo**" en "**Restaurar tipo**".
6. **Tipo incremento**: cuándo se incrementará la variable (con cada registro, al final del informe, al final de cada página, al final de cada columna o cuando el grupo especificado en "**Incrementar grupo**" cambie),
7. **Incrementar grupo**: el grupo que debe cambiar para que se incremente la variable cuando hemos elegido "**Grupo**" en "**Tipo incremento**"
8. **Clase fábrica responsable incrementar**: si queremos utilizar un incremento personalizado indicaremos el nombre de la clase que implementa la interfaz **net.sf.jasperreports.engine.fill.JRIncrementerFactory** encargada de realizar el incremento.
9. **Expresión de variable**: Será la expresión asociada a la variable que se evaluará en tiempo de ejecución y se realizarán los cálculos correspondientes según hallamos indicado en los campos anteriores.
10. **Valor inicial expresión**: Valor que tomará la variable cuando se inicialice.

Para referirnos a una variable "**NOMBRE_VARIABLE**" lo haremos mediante el identificador "**$V{NOMBRE_VARIABLE}**". Por ejemplo, en la imagen que hemos mostrado se define una variable llamada "**SUBTOTAL**" a la que nos referiremos como "**$V{SUBTOTAL}**" y que calcula para cada página la suma del campo "**CANTIDAD**", ya que se inicializa en cada página y el incremento lo realizamos en cada registro y todo ello referido a la expresión "**$F{CANTIDAD}**" que precisamente es como nos referimos al campo "**CANTIDAD**" y que es sobre el que queremos realizar la suma. Al inicializar la variable en cada página lo que estamos calculando es el subtotal para dicha página, a diferencia de si sólo se inicializase al principio del informe, que estaríamos calculando el acumulado.

Autoevaluación

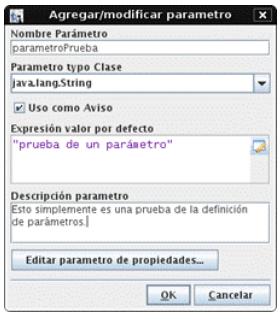
Respecto a las variables en iReport podemos afirmar que...

a) Almacenan valores asociados a una expresión o realizan cálculos sobre una expresión.
b) Son campos que tienen longitud variable.
c) En iReport no existe el concepto de variable, sólo existe en Java.
d) Es un concepto utilizado por JasperReport que aún no podemos utilizar en iReport

Comprobar

Conceptos básicos: Parámetros

¿Qué entendemos por parámetro en iReport?



Un parámetro es un valor que se pasa al informe en tiempo de ejecución.

Recuerda que para rellenar los informes lo hacíamos con la conexión a la BD y los parámetros (que podían no existir). Para poder utilizar un parámetro debemos definirlo para que la compilación se pueda llevar a cabo. Podríamos tener definido un parámetro y a la hora de rellenar el informe no pasarle dicho parámetro, con lo que su valor sería `null`. Podemos acceder a los parámetros de nuestro informe mediante la opción "**Parámetros de Informe**" del menú "**Ver**" o pulsando sobre el icono y eligiendo la pestaña "**Parámetros**" del diálogo que nos aparece. Desde ahí podremos añadir, modificar o borrar parámetros pulsando sobre el botón adecuado.

Para añadir un parámetro debemos definir:

- su nombre,
- la clase a la que pertenece,
- si queremos que iReport nos pregunte por su valor,
- si tiene un valor por defecto (que será utilizado cuando no pasemos dicho parámetro) y
- su descripción si queremos ponérsela.

El principal uso de los parámetros es personalizar nuestros informes mediante valores introducidos en tiempo de ejecución y que suelen ser introducidos por el usuario o calculados (por ejemplo, un informe que muestre los empleados cuyo nombre empiece por una letra que introducirá el usuario en tiempo de ejecución). Para nombrar un parámetro debemos hacerlo utilizando la sintaxis: `$P{nombre_parametro}` (por ejemplo, para referirnos al parámetro de la imagen lo haremos usando el identificador `$P{parametroPrueba}`).

Autoevaluación

Respecto a los parámetros en iReport, podemos afirmar que...

a) Son los distintos argumentos que pasamos a iReport a la hora de ejecutarlo
b) El concepto de parámetro es típico de una función Java, pero no de iReport.
c) Es un valor que se pasa a un informe en tiempo de ejecución
d) Es un concepto muy parecido al de campo.

Comprobar

Conceptos básicos: Objetos

Una vez más, ¿qué son los objetos?

Son los distintos elementos que podemos incluir en nuestro informe.

La visualización de cada objeto vendrá determinada por el comportamiento de la banda en la que lo hayamos situado. (Por ejemplo, no será lo mismo situar un objeto en la banda "**title**" que en la banda "**detail**"). Las propiedades de cada objeto las podemos ver en el panel de propiedades cuando tenemos seleccionado dicho objeto. También podemos pulsar con el botón derecho del ratón y en el menú contextual que nos aparece, elegimos "**Propiedades**", donde nos aparecerá un cuadro de diálogo con las propiedades de dicho objeto agrupadas en pestañas. Los objetos tienen unas propiedades comunes a todos ellos y otras específicas a cada uno de ellos. Las propiedades comunes son muy intuitivas y se refieren a la posición del objeto, su color y algunos detalles más. Hay una propiedad etiquetada como "**Imprimir cuando expresión**" que indica que el objeto se visualizará sólo cuando sea cierta la expresión. Esta expresión espera un objeto `Bolean`, por lo que si queremos utilizarla deberemos poner algo como "`new Boolean($F{CAMPO} != null)`" (por ejemplo), lo importante es que debemos crear el objeto. Los objetos de los que disponemos son los siguientes (adjuntamos su icono en la barra de herramientas estándar):

1. **Línea**: Las líneas se suelen utilizar para separar filas de nuestro informe. Podemos personalizar el trazo con el que la dibujamos.
2. **Rectángulo**: Los rectángulos los utilizamos para resaltar elementos como una cabecera de texto o un total. Podemos personalizar su trazo y el radio con el que se redondean sus esquinas.
3. **Elipse**: Suelen utilizarse para resaltar elementos y podemos personalizar el trazo con el que la dibujamos.
4. **Texto estático**: Es texto que no varía y es utilizado en cualquier sitio donde necesitemos presentar texto invariable como en el título, la cabecera de las columnas, ... Podemos personalizar la fuente, sus atributos, el espaciado, la justificación, la orientación y el borde.
5. **Campo de texto**: Es texto dinámico ya que visualiza una expresión que es evaluada en tiempo de ejecución. Podemos personalizar la fuente, sus atributos, el espaciado, la justificación, la orientación, su clase, la máscara a utilizar en su visualización, el tiempo de evaluación, el borde y si es un vínculo a una URL, a anclas dentro del documento o en otros documentos... Esta última opción no es muy utilizada ya que la mayoría de los informes al final se imprimen (aunque se debería hacer sólo cuando sea imprescindible y así ahorrar papel y cuidar el medio ambiente).
6. **Imagen**: Añade una imagen a nuestro informe, como puede ser el logotipo de nuestra organización. Debemos indicar la expresión para cargar la imagen y podemos personalizar su escala, su justificación, si es enlace y su borde.
7. **Código de Barras**: Añade un código de barras a nuestro informe. Debemos indicar la expresión que queremos codificar y podemos personalizar el tipo de codificación, si queremos comprobar la suma, si queremos ver también el texto, su tamaño, su escala, su justificación, el tiempo de evaluación y el borde.
8. **Marco**: No es más que un elemento en el cuál podemos incluir subelementos. Podemos decidir si tendrá borde y su tipo.
9. **Gráfica**: Es una representación gráfica de un conjunto de datos. Dedicaremos un apartado a este tipo de elementos.
10. **Subinforme**: Define un subinforme dentro del informe actual. A este elemento también dedicaremos un apartado.

11. **Tabla de referencias cruzadas**: Una tabla de referencias cruzadas no es más que un tipo especial de tabla en la que las filas y las columnas son dinámicas. Son usadas para resumir información de una forma compacta con múltiples niveles de agrupamiento tanto para filas como para columnas. Deben ser colocadas en la banda "**summary**". También dedicaremos otro apartado a este elemento.

Decir también que con todos estos elementos podemos usar las distintas opciones de la barra de herramientas de formato de objetos que permitían cambiar la posición, el tamaño y la alineación de los diferentes elementos. Para seleccionar varios elementos podremos ir seleccionando uno tras otro manteniendo la tecla "**mayúsculas**" pulsada. Al seleccionar varios objetos, las operaciones que realicemos serán aplicadas a todos los seleccionados y tendrán como referencia el primer objeto seleccionado (por ejemplo, si seleccionamos dos objetos y realizamos la operación "Misma anchura", el segundo objeto tomará la anchura del primero).

Vistos estos conceptos pasemos a ponerlos en práctica diseñando distintos tipos de informes que luego añadiremos a nuestra aplicación.

Autoevaluación

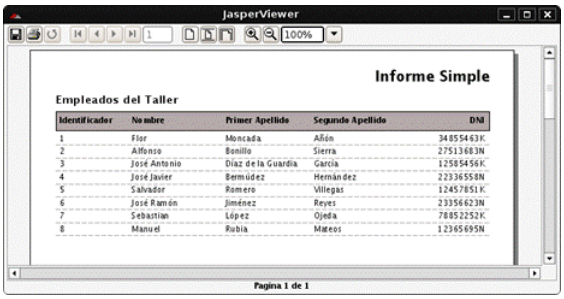
Indica cuál de los siguientes no es un objeto en iReport

a) Curva
b) Línea
c) Rectángulo
d) Elipse

Comprobar

Diseñando nuestro primer informe

Ya hemos visto los conceptos básicos que utiliza iReport, por lo que para ir afianzando dichos conceptos nada mejor que ir poniéndolos en práctica. Recordarás que en nuestro caso de estudio creamos una aplicación que hacía uso de un informe simple que te dijimos que aprenderías a diseñar (ya que en ese momento te lo suministramos nosotros). Pues ha llegado el momento de hacer dicho diseño y verás como todo es muy sencillo. Antes de nada volvamos a examinar el informe: (Haz clic sobre él para ampliarlo)



Lo primero que hemos hecho es crear un nuevo informe en blanco. Seguidamente hemos definido la consulta a la Base de Datos, por medio de la opción "**Consulta de informe**" del menú "**Datos**". La consulta que hemos realizado es "`SELECT idempleado, nombre, apellido1, apellido2, dni FROM empleados`". Al escribir la consulta podemos ver cómo automáticamente se crean los campos asociados a los atributos de nuestra consulta.

En este informe podemos distinguir claramente que se están utilizando sólo cuatro bandas de todas las posibles y que son las siguientes:

1. **title**: En ella hemos añadido un texto estático representando el título de nuestro informe. Lo hemos situado pegado a la esquina superior derecha de la banda, lo hemos puesto en negrita, hemos cambiado su tamaño y lo hemos justificado a la derecha.
2. **pageHeader**: En ella hemos añadido otro texto estático que se repetiría al principio de cada página (aunque nuestro informe actualmente sólo consta de una página debido a la cantidad de datos que posee nuestra BD). Lo hemos situado pegado a la esquina superior izquierda, lo hemos puesto en negrita y hemos cambiado su tamaño.
3. **columHeader**: Como esta banda es la cabecera de las columnas, hemos añadido 5 textos estáticos con el nombre de cada columna. Los textos simplemente los hemos puesto en negrita y sólo el "DNI" lo hemos justificado a la derecha. Hemos cambiado sus posiciones y tamaño para que todo tenga el aspecto deseado. Para resaltar esta cabecera hemos añadido un rectángulo que ocupa toda la banda y que tiene el fondo gris.
4. **detail**: En esta banda hemos añadido 5 campos de texto y la expresión que contiene cada uno es el campo correspondiente. Hemos cambiado sus posiciones y tamaño para que todo tenga el aspecto deseado.

Una vez hecho esto ya tenemos diseñado nuestro informe. Ahora sólo nos queda rellenarlo con los datos de la BD y visualizarlo para ver si su aspecto es el que esperábamos. En el menú "**Construir**" nos hemos asegurado que está seleccionada la opción "**Vista previa en JRViewer**" para que no exporte a ningún formato, sino que simplemente lo visualice con el visor de JasperReport. Seguidamente hemos compilado el informe pulsando sobre el icono . Comprobamos que nuestro informe ha compilado correctamente (después de corregir un pequeño error que nos daba de incompatibilidad de tipos, ya que el campo `IDEMPLEADO` es del tipo `BigDecimal` y en su campo de texto asociado hemos dejado la clase por defecto, que es `String` y debido a que iReport no sabe hacer la conversión, por lo que simplemente cambiando la clase del campo de texto asociado de `String` a `BigDecimal` se corrige el error) mirando en el panel de consola de salida. Ahora ya sólo nos queda ejecutar el informe por medio del icono para visualizarlo y comprobar que efectivamente ese erá el aspecto que deseábamos para él. En la siguiente demostración puedes apreciar los pasos que hemos seguido para diseñar nuestro primer informe.

Diseño del primer informe

Como punto de partida no está mal, pero la verdad es que el aspecto de nuestro informe es un poco "cutre". Ahora vamos a diseñar un

informe más completo (lo que no quiere decir que sea más complejo, como podrás apreciar), en el que utilizaremos algunos de los conceptos adquiridos en los apartados anteriores, para intentar darle a nuestro informe un aspecto más profesional.

Autoevaluación

Respecto al informe simple que acabamos de diseñar podemos afirmar que...

a) No hemos podido utilizar imágenes ya que es imposible incluir imágenes en un informe con iReport
b) No hemos podido cambiar el color de la fuente en las columnas, ya que en iReport si cambiamos el color de una fuente, éste será el color de todas las fuentes que aparezcan en el informe
c) La consulta era tan simple porque en iReport no podemos utilizar consultas que realicen JOIN de varias tablas
d) Todas las respuestas anteriores son falsas

Comprobar

Diseñando un informe más elaborado (I)

Después de ver el informe que acabamos de diseñar, te estarás preguntando: ¿Esa es toda la potencia que nos ofrece iReport para diseñar informes? ¿Para generar un informe tan simple debemos montar toda esa parafernalia? A lo largo de este apartado y siguientes veremos cómo la potencia de iReport es enorme y todo lo que hemos hecho nos sirve para diseñar cualquier tipo de informe y poder utilizarlo en nuestras aplicaciones (bien sea para visualizarlo o para exportarlo a distintos formatos).

A lo largo de los apartados anteriores hemos ido conociendo conceptos útiles para el diseño de informes con iReport. La finalidad de este apartado y los siguientes es poner en práctica dichos conceptos. En este apartado nos dedicaremos a diseñar un informe que haga uso de algunos de estos conceptos y en los siguientes utilizaremos otros más complejos. Para realizar nuestros informes nos basaremos en el caso de estudio del taller informático.

Para poner en práctica estos conceptos diseñaremos un informe que:

- contendrá una imagen,
- hará uso de un parámetro que restringirá la consulta a la BD,
- utilizará variables para realizar distintos cálculos,
- incluirá expresiones que se evaluarán en tiempo de ejecución y
- usará la visualización condicional.

Echemos un vistazo al aspecto del informe que queremos diseñar para luego ver los distintos elementos que incluye y cómo hemos llegado al resultado final.

Podemos observar cómo el informe muestra un desglose de la mano de obra asociada a una reparación dada, que será un parámetro de dicho informe. Este parámetro lo hemos definido con el nombre **`ID_REPARACION`** (su nombre podría haber sido otro) y su tipo es **`BigDecimal`**, que es el tipo del campo **`id_reparacion`** de la BD y así evitar incompatibilidades de tipos. Este parámetro restringirá la consulta a la BD, que es la siguiente:

```
select
        a.nombre_accion, a.horas,
        c.preciohora
from reparacion_acciones ra, acciones a, categoria c
where
        ra.id_accion=a.id_accion and
        a.idcategoria=c.idcategoria and
        ra.id_reparacion = $P{ID_REPARACION}
```

Vemos que de la consulta automáticamente se generan tres campos cuyo nombre será: **`NOMBRE_ACCION`**, **`HORAS`** y **`PRECIOHORA`**. Estos campos serán los que mostremos en la banda **detail**. Nuestro informe también muestra en dicha banda el precio total de cada acción como resultado de multiplicar el número de horas por el precio de la hora para dicha acción. Para realizar dicho cálculo utilizamos una variable llamada **`PRECIO_ACCION`** que lo único que hace es la multiplicación de ambos campos.

Diseñando un informe más elaborado (II)

¿Qué otros elementos tenemos que seguir añadiendo al informe para mejorar su aspecto? Es el momento de detallar todo lo que éste incluye:

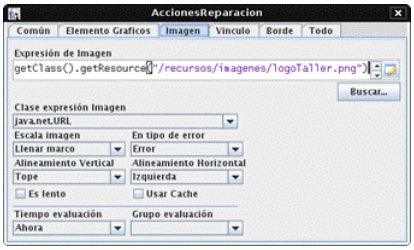
- En la banda **title** mostramos:
  - el título del informe y
  - una cabecera con información del taller informático.

El título está compuesto por:

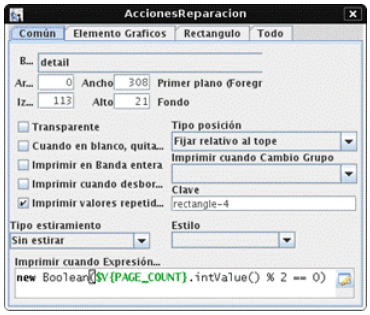
- un texto estático,
- un campo de texto en el que visualizamos el parámetro del informe,
- un rectángulo con los bordes redondeados.

Para la cabecera utilizamos:

- otro rectángulo,
- cinco textos estáticos con información del taller y
- una imagen que muestra el logotipo.



Para el logotipo podríamos haber utilizado como tipo un `String` cuyo valor sería el camino hacia el archivo correspondiente. Pero si lo hacemos así, cuando visualicemos el informe desde nuestro programa, que estará empaquetado en un archivo JAR, éste no podrá ser encontrado. Para ello hemos utilizado como tipo una `URL` y como expresión el valor devuelto por el método `getResource` que ya conocemos (para que se visualice correctamente desde iReport hemos copiado el logotipo al directorio desde donde estamos ejecutando el informe incluyendo la jerarquía que utilizaremos en nuestro programa `/recursos/informes/logoTaller.png`).



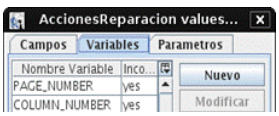
- La banda **columnHeader** está compuesta de:
  - otro rectángulo y
  - cuatro textos estáticos indicando el título de las columnas que vamos a mostrar en la banda detail que ya hemos comentado anteriormente.
- La banda **detail** contiene:
  - un rectángulo, con el color de fondo igual al de primer plano para evitar que se muestre el borde y cuya visualización es condicional, ya que sólo se imprime en las filas pares. Para ello hemos utilizado una expresión condicional que hace uso de una variable definida por iReport, `$V{PAGE_COUNT}`, que almacena el valor de los registros procesados. Si como tiempo de evaluación utilizamos "ahora" nos estará indicando el número de la fila que estamos procesando.
- La banda **pageFooter** está compuesta de:
  - otro rectángulo redondeado
  - y tres campos de texto.

En el primero de los campos visualizaremos la fecha actual por medio de la expresión `new Date()` y utilizando un patrón de fecha adecuado. Los otros dos los utilizamos para mostrar la paginación del informe. En ambos se utiliza la misma variable definida por iReport `$V{PAGE_NUMBER}`, que nos mostrará la página actual si su tiempo de evaluación es "**ahora**" o el número total de páginas si su tiempo de evaluación es "**Reporte**".

Diseñando un informe más elaborado (III)

Continuamos explicándote las distintas bandas que debemos usar para definir el informe.

- Por último tenemos la banda **summary** en la que mostramos información resumen del informe. Está compuesta por:
  - otro rectángulo redondeado,
  - tres textos estáticos indicando la naturaleza de la información que mostraremos y
  - tres campos de texto que mostrarán la información resumen haciendo uso de variables.



Para el cálculo del total utilizamos una variable que realizará la suma de la variable **$V{PRECIO_ACCION}**. Para el precio medio por hora utilizamos una variable **$V{MEDIA_HORA}** que es el cociente entre **$V{TOTAL}** y **$V{NUM_HORAS}**, que es una variable auxiliar calculada como la suma del campo **$F{HORAS}**. El precio medio por acción **$V{MEDIA_ACCION}** es el cociente entre la variable **$V{TOTAL}** y **$V{NUM_ACCIONES}**, que es otra variable auxiliar calculada como el recuento de cualquiera de los campos, lo que nos dará el número de registros procesados (también podíamos haber utilizado la variable **$V{PAGE_COUNT}**).

En los siguientes enlaces puedes encontrar el fichero fuente del informe y una demostración que nos muestra las propiedades más importantes del informe.

Descarga el fichero fuente del informe

Mira las propiedades más importantes del informe

Puedes apreciar cómo el aspecto que hemos conseguido para nuestro informe está mucho más elaborado y que la tarea no ha sido nada del otro mundo, una vez que tenemos claros todos los conceptos. En el siguiente apartado veremos qué son los grupos y cómo podemos utilizarlos en nuestros informes.

Autoevaluación

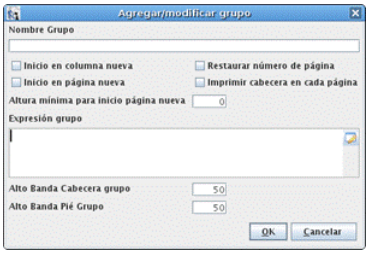
Al utilizar una imagen en nuestro informe:

a) Debemos indicar su camino absoluto para poder cargarla.
b) Podemos indicar su camino absoluto o el relativo, según nos interese
c) No podemos cargarlo en tiempo de ejecución ya que las imágenes van incrustadas en el informe
d) Podemos indicar su camino por medio de un String o de una URL, según nos convenga.

Comprobar

Informes con grupos

Como habrás podido apreciar, cualquier campo que introduzcamos en la banda **detail** será repetido en cada fila del informe. Pero si queremos mostrar información agrupada por algún campo, éste no podremos añadirlo a la banda **detail** para que sólo se muestre una vez como cabecera de la información agrupada por ejemplo. En cualquier otra banda sólo se mostrará una vez por página o por informe dependiendo del comportamiento de la banda en la que lo añadamos. Entonces parece que no podemos diseñar informes en los que podamos agrupar información. Pues no es así, ya que **iReport nos proporciona la funcionalidad de grupos para poder llevar a cabo este tipo de informes**. Veamos cómo podemos hacer uso de la misma.



Cuando un informe necesita mostrar información agrupada debemos utilizar los grupos. Al definir un grupo se crean dos nuevas bandas en nuestro informe; una para la cabecera del grupo y otra para el pie del grupo. Un grupo es definido por una expresión. Para cada registro procesado, JasperReport evalúa la expresión y si su valor cambia entonces el valor del grupo también cambia. Generalmente agruparemos por campos de una consulta, por lo que debemos tener en cuenta el orden de los datos de dicha consulta, ya que si nuestra consulta contiene los datos para un campo de (X Y X Y) JasperReport creará cuatro grupos cuando nosotros esperábamos dos. Para ello nuestra consulta debe estar ordenada por medio de la cláusula **ORDER BY**. Cuando queremos agregar un grupo sólo debemos dar obligatoriamente su nombre, con lo que lo único que conseguiremos será que se añadan las dos bandas que hemos mencionado, ya que nuestro grupo no tendrá ninguna funcionalidad. Otros campos que podemos definir son (como podemos apreciar en la imagen anterior) la expresión, si queremos que la cabecera del grupo se imprima en una nueva columna, en una nueva página, si se debe forzar a imprimir la cabecera en las páginas nuevas, su altura, su anchura, ...

Para mostrar cómo podemos utilizar los grupos lo mejor será crear un informe que haga uso de los mismos.

Vamos a **diseñar un informe** que nos muestre los clientes del taller informático agrupados por provincias. El aspecto del informe será el que mostramos en la figura de la derecha. Como puedes apreciar se nos muestra el identificador, el nombre y la localidad de los clientes del taller informático agrupados por provincia. La consulta que hemos utilizado es:

```
SELECT
        c.idcliente, c.nombre, c.poblacion, p.provincia
FROM
        clientes c, provincias p WHERE c.codprovincia = p.codprovincia ORDER BY p.provincia, c.idcliente
```

Como ves hemos ordenado la consulta para que los grupos se creen correctamente. Luego simplemente hemos creado un nuevo grupo cuyo nombre es **Provincia** y la expresión del grupo es el campo **$F{PROVINCIA}** y este campo es el que hemos incluido en la banda **ProvinciasHeader**. En los siguientes enlaces podrás encontrar el fichero fuente del informe y una demostración en la que puedes ver cómo

hemos realizado el diseño.

Aspecto final del informe

Fichero fuente del informe

Presentación sobre cómo se ha hecho el informe

Ahora que hemos visto lo fácil que resulta agrupar información dentro de un informe, pasemos a ver cómo utilizar gráficas en ellos y así dotarlos de un aspecto más profesional.

Autoevaluación

Para definir un grupo en iReport debemos...

a) Indicar como mínimo el nombre del grupo
b) Primero añadir una nueva banda donde situaremos el grupo.
c) Utilizar un subinforme para mostrar los detalles del grupo
d) En iReport no podemos definir grupos

Comprobar

Informes con gráficas

¿Te acuerdas? Ya comentamos al principio que JasperReport permitía utilizar gráficas y que para ello hacía uso de una librería libre llamada JFreeChart.

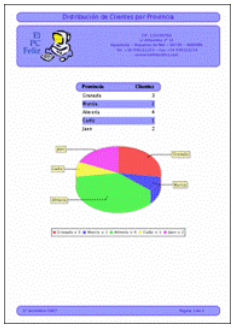
Efectivamente, entre las librerías que añadimos a nuestra aplicación estaba una llamada **`jfreechart-1.0.0.jar`**, que es la responsable de la generación de gráficas en nuestros informes. Esta librería nos permite hacer muchas clases de gráficas como puedes ver en la imagen que acompaña a este texto. En este apartado veremos **cómo crear una gráfica sencilla que sólo tendrá una serie de datos**. Los demás tipos son igual de fácil de manejar, aunque no nos detendremos en los mismos, ya que su explicación sobrepasa el propósito de esta unidad.

JFreeChart también puede ser utilizado directamente desde nuestras aplicaciones en Java, para ello simplemente deberemos añadir el fichero JAR a nuestro proyecto y utilizar el API que nos proporciona para generar las gráficas que deseemos.

PARA SABER MÁS

En el siguiente enlace podrás encontrar un tutorial de cómo utilizar JfreeChart desde Oracle JDeveloper para añadir gráficas a tus aplicaciones.

Creación de gráficas en nuestras aplicaciones Java [Versión en caché]



Para ilustrar cómo utilizar las gráficas en nuestros informes **vamos a crear un informe que mostrará para cada provincia el número de clientes que tenemos en la misma**. Con estos datos construiremos una gráfica del tipo tarta 3D. Este tipo de gráfica sólo contiene una serie de datos. El aspecto que tendrá nuestro informe es el que te mostramos en la imagen adyacente. Puedes ver el aspecto final del informe con más detalle mediante el siguiente enlace.

Descarga el informe

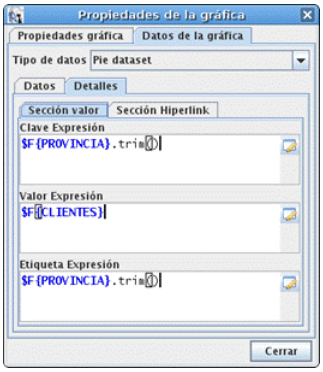
La consulta que realiza este informe es bastante sencilla:

```
SELECT p.provincia, count(*) as Clientes
FROM clientes c, provincias p
```

```
WHERE p.codprovincia = c.codprovincia

GROUP BY p.provincia.
```

Como puedes ver, devuelve para cada provincia el número de clientes y estos datos serán los que utilicemos para rellenar nuestra gráfica. Para añadir una gráfica a nuestro informe debemos situarla en la banda **summary** del mismo, lo haremos por medio del icono y se nos preguntará por el tipo de gráfica que queremos crear. En nuestro caso elegimos "Pie 3D" para crear una gráfica del tipo tarta en 3 dimensiones.



Una vez añadida la gráfica, debemos decirle de dónde tiene que coger los datos para representar la gráfica. Para ello accederemos a la opción "**Opciones de diseño**" del menú contextual que nos aparece al pulsar con el botón derecho del ratón sobre la gráfica. Nos aparece una ventana con dos pestañas: en la primera etiquetada como "**Propiedades gráfica**" podemos modificar propiedades generales de la gráfica como colores, fuentes, leyenda... En la segunda etiquetada como "**Datos de la gráfica**" contiene otras dos pestañas de las cuales nos interesa la segunda, etiquetada como "**Detalles**", en la que podremos definir los datos que formarán la gráfica.

Para ello debemos decir cuál será la clave (en nuestro caso la categoría de cada trozo de la tarta), el valor (o la amplitud de cada trozo de la tarta) y la etiqueta asociada a cada categoría. En la imagen que te mostramos puedes ver los valores que hemos dado a la gráfica de nuestro informe.

En los siguientes enlaces podrás encontrar el fichero fuente del informe y una demostración en la que puedes ver cómo hemos realizado el diseño.

Fichero fuente del informe

Presentación sobre la elaboración del informe

Como has podido apreciar la inclusión de gráficas en nuestros informes es una tarea prácticamente trivial y el aspecto de nuestro informe mejora bastante. En el siguiente apartado veremos cómo utilizar tablas de referencias cruzadas en nuestros informes.

Autoevaluación

Para utilizar gráficas en iReport debemos...

a) Pagar una licencia por el uso de la librería JFreeChart
b) Indicarle a iReport los datos que deberá utilizar para rellenar el informe y el tipo de gráfica
c) Utilizar una fuente de datos especial que sólo es utilizada para rellenar gráficas.
d) En iReport no podemos utilizar gráficas

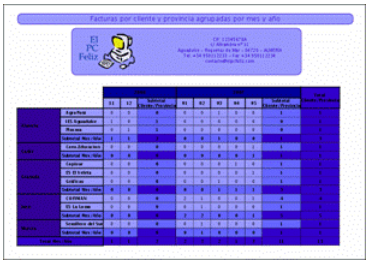
Comprobar

Informes con tablas de referencias cruzadas

Un recurso muy utilizado para mostrar información de una forma compacta y legible es el uso de tablas de referencias cruzadas. Quizás te estarás preguntando: ¿qué es una tabla de referencias cruzadas?

Pues no es más que un tipo especial de tabla en el que las filas y las columnas son dinámicas y que se suelen utilizar para resumir información (recuentos, sumas ...) con diferentes niveles de agrupamiento tanto en filas como en columnas. Parece una definición compleja, pero ya verás como en cuanto veas un ejemplo sabes de lo que estamos hablando.

En el informe de la imagen hemos incluido una tabla de referencias cruzadas en la que mostramos el número de facturas emitidas. Las filas están agrupadas por provincia y cliente. Las columnas están agrupadas por años y meses. Cada celda individual muestra el número de facturas emitidas para un cliente dado en una fecha, que pertenecerá a una provincia. También se han añadido celdas en las que se calcula la suma por cliente, provincia, mes y año. Finalmente hay un cálculo global del número de facturas. Se han utilizado diferentes colores para diferenciar las celdas que realizan cálculos de las demás.



Empezaremos por examinar la consulta que realizamos a la BD para poder rellenar la tabla del informe:

```
SELECT p.provincia, c.nombre, to_char(f.fecha,'mm') mes, to_char(f.fecha,'yyyy') año, count(*)

FROM facturas f,clientes c, provincias p

WHERE f.idcliente = c.idcliente and p.codprovincia = c.codprovincia GROUP BY p.provincia, c.nombre, to_char(f.fecha,'mm
```

Como puedes ver la consulta nos devuelve exactamente el número de facturas agrupadas por provincia, nombre de cliente, mes y año. En ningún momento realizamos los cálculos por grupos en la consulta, ya que eso lo hará nuestra tabla por sí sola.

Para añadir la tabla de referencias cruzadas a nuestro informe lo haremos por medio del icono y deberemos situarla, al igual que los gráficos, en la banda **summary**, ya que se trata de información de tipo resumen. Una vez situada nos aparecerá un asistente que nos guiará en los diferentes pasos a seguir:

1. **Paso 1**: Conjunto de datos a utilizar, en el que generalmente dejaremos el que viene por defecto, que son los datos de nuestro informe.
2. **Paso 2**: Agrupamiento sobre filas. Para nuestro ejemplo elegimos como primer grupo el campo **`$F{PROVINCIA}`** y como segundo el campo **`$F{CLIENTE}`**.
3. **Paso 3**: Agrupamiento sobre columnas. En nuestro caso elegimos como primer grupo el campo **`$F{AÑO}`** y como segundo el campo **`$F{MES}`**.
4. **Paso 4**: Definición de los detalles, donde indicaremos cómo rellenar las celdas individuales y si queremos realizar algún cálculo sobre las mismas. Para nuestro ejemplo indicaremos como detalle el campo **`$F{COUNT(*)}`** y como cálculo la función suma.
5. **Paso 5**: En este último paso indicamos si queremos que se añadan los totales de grupo para filas, para columnas y si queremos visualizar la rejilla de la tabla. Para nuestro ejemplo dejaremos marcadas las tres opciones.

Con estos pasos tan simples hemos terminado de definir la tabla de referencias cruzadas. En nuestro informe principal nos aparece una pestaña etiquetada como "**crosstab-1**" en la que podemos acceder al diseño de las celdas individuales. Nosotros hemos jugado un poco con el diseño de las celdas para darle un poco de color a nuestra tabla. En los siguientes enlaces podrás encontrar el fichero fuente del informe y una demostración en la que puedes ver cómo hemos realizado el diseño.

Fichero fuente del informe

Presentación sobre la creación del informe

Como todo lo que venimos haciendo con iReport ha sido bastante fácil y los resultados obtenidos bastante buenos. Seguro que a estas alturas has perdido el miedo a la generación de informes ¿no?

Veamos el último "giro de tuerca" en la generación de informes, que es la utilización de subinformes.

Autoevaluación

Para utilizar tablas de referencias cruzadas en iReport debemos...

a) Mezclar dos informes distintos, que serán los que generen la tabla
b) Indicarle a iReport los campos por los que agruparemos, la expresión utilizada como detalle y los tipos de funciones para los totales, si es que queremos utilizarlos
c) Utilizar como fuente de datos un Jtable
d) En iReport no podemos utilizar tablas de referencias cruzadas

Comprobar

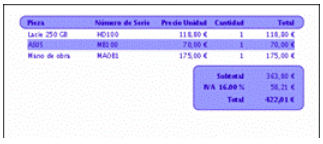
Subinformes: Empezando

¿Crees que ya has visto todo lo que tenías que ver sobre informes? Pues la verdad es que no, ya que te queda por ver una de las funcionalidades más potentes que ofrece iReport que es **la inclusión de subinformes en un informe principal**.

Esta funcionalidad es utilizada para la generación de informes maestro - detalle, ya que sin ella algunos informes no se podrían llevar a cabo o las consultas que se utilizarían serían muy complejas y poco optimizadas.

Un subinforme no es más que un informe ya diseñado que podemos incluir en otro informe principal. Generalmente un informe que va a actuar como subinforme suele tener parámetros para restringir la consulta a la BD. El informe principal o maestro le pasará información al informe detalle por medio de estos parámetros. Lo mejor será pasar a ver un ejemplo.

Queremos mostrar una factura dada. Para ello en la factura visualizaremos datos de la factura como su número, la fecha y la reparación asociada a la misma. También mostrará datos del cliente como su nombre, CIF, dirección, ... Todo esto lo hará el informe maestro o principal.



Una factura debe mostrar las líneas de factura asociadas a dicha factura. Para mostrar las líneas de detalle de la factura utilizaremos un subinforme. Por último queremos desglosar la mano de obra de la reparación asociada a la factura y para ello utilizaremos otro subinforme. Veamos primero cómo hemos diseñado los subinformes para pasar a ver cómo incluirlos en el informe maestro y cómo pasarle los parámetros adecuados.

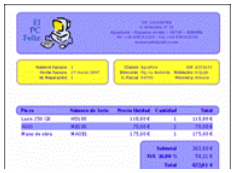
Para mostrar las líneas de detalle hemos diseñado un informe muy simple (ya que la idea es incluirlo como subinforme en otro), que no tendrá la cabecera del taller informático ni tampoco paginación. El aspecto es el que muestra la imagen de la izquierda. El informe acepta dos parámetros: **`$P{CODFACTURA}`** indicando el código de la factura de la que queremos mostrar las líneas de detalle y **`$P{IVA}`** indicando el iva aplicable. La consulta es bastante sencilla:

```
 SELECT ld.numero_serie, ld.cantidad, ld.preciounitario, p.numero_serie, nombre
FROM lineasdetalle ld, piezas p
WHERE ld.numero_serie = p.numero_serie and ld.codfactura = $P{CODFACTURA}.
```

En el siguiente enlace puedes encontrar el código fuente del informe.

Aspecto del informe

Fichero fuente del informe



Como comprobarás, para mostrar el desglose de la mano de obra hemos utilizado un informe muy parecido al que elaboramos en un apartado anterior, pero al que le hemos quitado la cabecera del taller informático y la paginación, pero la consulta a la BD es la misma y sigue utilizando el mismo parámetro que utilizaba. Te

incluimos el código fuente del mismo.

Descarga el código fuente

El informe que representará la factura tendrá el aspecto que puedes observar en la imagen de la derecha. El informe maestro tendrá un parámetro llamado **$P{CODFACTURA}** indicando la factura a visualizar. La consulta que realizaremos a la BD será:

```
 SELECT f.fecha, f.iva, f.id_reparacion, c.nombre, c.cif, c.direccion, c.codpostal, c.poblacion, p.provincia
FROM facturas f, clientes c, provincias p
WHERE f.idcliente = c.idcliente and c.codprovincia = p.codprovincia and f.codfactura = $P{CODFACTURA}
```

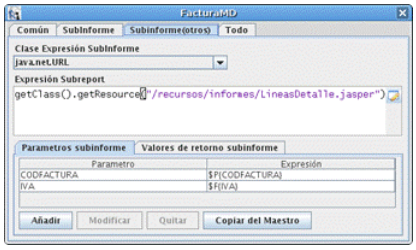
Como puedes observar, dos de los campos que seleccionamos son el iva y el id_reparacion. El campo **$F{IVA}**, junto a **$P{CODFACTURA}** se pasarán al subinforme LíneasDetalle. El campo **$F{ID_REPARACION}** se pasará al subinforme AccionesDetalle. Para incluir los dos subinformes hemos creado dos grupos sin ninguna funcionalidad y en la banda cabecera de cada uno de ellos es donde hemos añadido el subinforme.

Aspecto del informe

Subinformes: Pasos a dar

Para añadir un subinforme debemos utilizar el icono  . Nos aparecerá un asistente que nos guiará por una serie de pasos:

1. **Paso 1**: **Elección del subinforme**, donde debemos indicar el subinforme compilado (.jasper) que vamos a utilizar.
2. **Paso 2**: **Fuente de datos utilizada para rellenar el subinforme**, que generalmente será la misma que la del informe principal o maestro.
3. **Paso 3**: **Qué valores pasaremos a los parámetros del subinforme**.
4. **Paso 4**: **Expresión utilizada para llamar al subinforme**, donde sólo nos permite utilizar una variable para el directorio y concatenarle el nombre del subinforme o utilizar una ruta estática. A nosotros no nos interesa ninguna de estas dos opciones, ya que como en el caso de las imágenes nos interesa llamar al subinforme con un camino evaluado en tiempo de ejecución para que sea válido cuando nuestra aplicación esté empaquetada en un archivo JAR, por ejemplo. Así que elegimos la segunda opción y luego mediante las propiedades del subinforme cambiamos el tipo de expresión a URL y en la expresión utilizamos el método getResource.



En los siguientes enlaces puedes encontrar el código fuente del informe maestro (recuerda que para que te funcione correctamente debes tener compilados los dos subinformes y colocados en el directorio adecuado) y una demostración en la que puedes seguir los diferentes pasos utilizados para la creación de la factura maestro - detalle que hemos utilizado como ejemplo.

Fichero fuente del informe

Presentación sobre la elaboración del informe

Con los subinformes vemos que las posibilidades para la creación de informes con iReport es casi ilimitada y la limitación está en nuestra imaginación y destreza. Para acabar con los informes veremos cómo añadir a nuestra aplicación "**GeneradorInformes**" la posibilidad de gestionar todos los informes que hemos utilizado en los apartados anteriores. Para ello crearemos una nueva versión de la misma que etiquetaremos como **v2.0**, ya que la **v1.0** sólo mostraba el informe simple. Podrás comprobar que los cambios que hemos tenido que introducir son mínimos. Pasemos a verlos.

Autoevaluación

Un subinforme en iReport podemos definirlo como...

a) Simplemente un informe que se incluye en otro
b) Son informes que utilizan como consulta a la BD una consulta anidada
c) Son informes que hemos realizado con una herramienta externa a iReport
d) En iReport no existen los subinformes

Comprobar

Incluyendo todos los informes en nuestra aplicación

¿Qué pasos hemos dado para incluir todos esos informes anteriores en la aplicación?

Nada complicado, en realidad. Para que nuestra aplicación pueda mostrar y exportar los informes

que hemos creado en los apartados anteriores, lo primero que hemos hecho ha sido añadirlos al directorio `src/recursos/informes` de nuestro proyecto. Hemos añadido los ficheros `.jrxml` de los informes vistos y los `.jasper` de los subinformes.

Como recordarás Oracle JDeveloper al compilar copiaba los ficheros del directorio `src` al directorio `classes`. Sólo se copiaban aquellos que estaban contenidos en su lista de extensiones a copiar y que nosotros podíamos modificar. **Ya modificamos esta lista para añadir la extensión `.jrxml` y ahora también la hemos modificado para añadir la extensión `.jasper`, ya que los ficheros con dicha extensión también deben ser copiados para que los informes que incluyen subinformes funcionen correctamente.**

Hemos modificado la interfaz gráfica de nuestra aplicación para que podamos elegir el informe que queremos exportar y/o visualizar, ya que la primera versión sólo nos permitía hacerlo con un informe simple. Para ello hemos añadido cinco radio botones más al panel de informes para que al pulsar el botón de visualización, la acción se realice sobre el informe elegido. Estos radio botones también los hemos añadido a la política de focos para que el orden que seguimos al pulsar el tabulador sea el deseado.

También hemos añadido un `JFileChooser` para elegir el fichero de salida en caso de que queramos exportar el informe. Como recordarás en la primera versión de la aplicación no se nos preguntaba por el nombre del fichero al que queríamos exportar y se generaba directamente en el directorio desde donde estábamos ejecutando la aplicación.

En la primera versión no necesitábamos pasar ningún parámetro al informe simple, pero en esta segunda versión tenemos informes que necesitan que el usuario decida el parámetro que queremos pasar al informe, como por ejemplo en la factura, que debemos indicarle el código de dicha factura. Para ello hemos utilizado un `JOptionPane` que requieren una entrada del usuario. Para que el usuario sólo pueda elegir las facturas existentes (o las reparaciones en el caso del otro informe que necesita parámetros) hemos añadido el método adecuado a la clase BD que nos devuelve los códigos de la factura (o los identificadores de las reparaciones) existentes en la BD.

Como ves, los cambios han sido mínimos y sin embargo la potencia de nuestra aplicación ha aumentado notablemente. Los principales cambios han sido la **adición de los informes fuente**, con lo que ha recaído todo el esfuerzo en el diseño de dichos informes.

¡Importante!

Nosotros hemos utilizado los informes fuente `.jrxml` en nuestra aplicación, ya que como estamos aprendiendo es muy probable que tengamos que modificarlos frecuentemente hasta conseguir el aspecto deseado. **En una aplicación ya terminada lo normal sería utilizar los informes ya compilados `.jasper` y así evitamos la pérdida de tiempo que conlleva la compilación del informe.**

Lo mejor será que eches un vistazo al código fuente de la segunda versión de nuestra aplicación. También te adjuntamos el fichero JAR de la misma por si quieres ejecutarla directamente antes de ponerte a examinar el código.

Código fuente de la aplicación

Fichero JAR para ejecutar la aplicación

¡Pues hemos acabado!

Hemos terminado de ver cómo utilizar los informes en nuestras aplicaciones y cómo diseñar los mismos. Habrás podido comprobar que la tarea ha sido bastante simple, ya que para incluir los informes en tus aplicaciones puedes reutilizar el código de la aplicación que hemos ido desarrollando. La parte del diseño de los mismos tampoco es gran cosa aunque en el mismo entra nuestra imaginación para darle un aspecto elegante. En el resto de esta unidad trataremos sobre cómo dotar a una aplicación de un **sistema de ayuda en línea,** que es algo que cualquier aplicación que se precie no puede obviar.

Autoevaluación

Para pedir al usuario que introduzca el valor de los parámetros hemos utilizado la clase...

a) JOptionPane.
b) JDialog
c) JFileChooser
d) La entrada estándar

Comprobar

CASO.

__Víctor__ está muy contento con el resultado de los informes que ha generado para la aplicación del taller informático y sabe que no hubiera sido posible sin la ayuda de __Carmen__. Además __José__ lo ha felicitado por el trabajo realizado y dentro de los plazos acordados. Todo esto le ha subido la moral bastante.

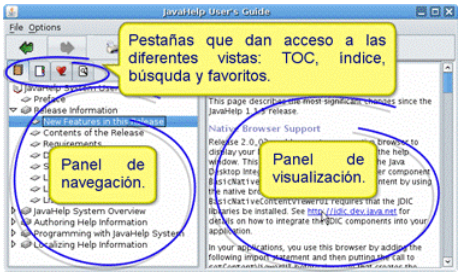
Debido a esto, __Víctor__ está dispuesto a agradecer a sus compañeros y sus compañeras la ayuda prestada y la confianza depositada en él. Para ello ha decidido ser proactivo, ya que sabe que esa es una virtud muy valorada hoy día en cualquier empresa. Para demostrar esta proactividad está dispuesto a adelantarse e incluir un sistema de ayuda en línea en la aplicación del taller informático y así dotarla de un aspecto más profesional.

__Víctor__ no tiene ni idea de cómo poder llevar a cabo esta tarea, pero está convencido de que es capaz de utilizar los recursos que brinda Internet para documentarse. Confía plenamente en sus posibilidades. Sabe que le va a costar un poco más trabajo que si fuese guiado por __Carmen__, pero quiere darle esta sorpresa y es consciente de que no siempre estará __Carmen__ para sacarle las castañas del fuego. Así que tiene un arduo trabajo de documentación hasta encontrar la herramienta adecuada. Una vez encontrada la herramienta deberá formarse en la misma hasta dominarla para poder desarrollar dicho sistema de ayuda. Lo mejor de todo es que tiene ganas e ilusión y esa es una de las mejores ayudas de las que podemos disponer a la hora de enfrentarnos a un trabajo.

Generación de Ayuda en línea: ¿Qué elementos incluye?

¿Cuántas veces habrás tenido que recurrir a la ayuda de una aplicación para saber cómo llevar a cabo una tarea?

Seguramente muchas. Incluso cuando esa tarea la has realizado con anterioridad, pero hace mucho tiempo que no utilizas dicha aplicación, no recuerdas bien cómo realizarla. Otras veces te habrá pasado que te enfrentas a una nueva aplicación que te acabas de bajar y cuando quieres acceder a la ayuda descubres que no contiene esta funcionalidad, con lo que piensas "Uh, me está dando mala espina esta aplicación".



Pues sí, la ayuda es una funcionalidad deseable en cualquier aplicación que se precie.

Como ya sabrás, un sistema de ayuda no es más que un visor que nos permite navegar de una forma sencilla a través de un conjunto de información estructurada que generalmente nos ayuda a comprender el funcionamiento de una aplicación. El visor suele estar compuesto de dos paneles:

- uno para la **navegación** y
- otro para la **visualización** del elemento seleccionado en el panel anterior.

Normalmente un sistema de ayuda consta de **varias vistas para el panel de navegación**:

- Una de ellas permite **navegar a través de una tabla** de contenidos
- Otra vista suele ser una **navegación a través de un índice**.
- Otra permite **realizar búsquedas** en los contenidos.
- Y una última que permite que **agreguemos los contenidos** más útiles para nosotros como **favoritos**.

Además de permitirnos navegar por dichos contenidos, también ofrece la posibilidad de imprimir los contenidos que deseemos.

Si nosotros queremos proporcionar a nuestra aplicación de un sistema de ayuda, nos va a constar mucho trabajo si lo que tratamos es implementarla desde cero. Para eso existen librerías que nos facilitan la tarea y hacen que se convierta en algo casi trivial. En este apartado trataremos de echar una mano a Víctor en la tarea que se ha propuesto. Por tanto veremos cómo podemos crear un sistema de ayuda y cómo podemos integrarlo en nuestras aplicaciones Java con muy poco esfuerzo.

Autoevaluación

Un sistema de ayuda, ¿qué elementos suele contener?

a) Un panel de navegación
b) Un panel de visualización.
c) Una barra de herramientas.
d) Todas las respuestas anteriores son correctas

Comprobar

Generación de Ayuda en línea: JavaHelp

La primera tarea que tenía Víctor era encontrar alguna herramienta que le sirviese para generar sistemas de ayuda. Nosotros hemos buceado un poco por Internet, para intentar ayudar a Víctor y rápidamente hemos descubierto que la herramienta más idónea para generar sistemas de ayuda en nuestras aplicaciones Java es una librería llamada JavaHelp.

JavaHelp no es la única librería para la visualización de sistemas de ayuda, ya que existen otras muchas. Sin ir más lejos, para Java existe otra librería desarrollada por Oracle y que se basa en JavaHelp, con algunas modificaciones. Esta librería se llama OHJ (Oracle Help for Java) y su aspecto es similar al que tiene la ayuda de Oracle JDeveloper, que está hecha con esta herramienta. También existen otros sistemas de ayudas para otros lenguajes o para otras plataformas, como puede ser WinHelp para Windows.

PARA SABER MÁS

En el siguiente enlace podrás encontrar información sobre las tecnologías que ofrece Oracle para la creación de sistemas de ayuda para Java.

Tecnologías de Oracle para la creación de sistemas de ayuda

JavaHelp es un paquete de software desarrollado por Sun Microsystems, cuyo código está abierto y podemos

descargarnos libremente. Está escrito íntegramente en Java usando JFC, por lo que es independiente de la plataforma.
Además permite crear sistemas de ayuda no sólo para aplicaciones cliente, sino que también permite utilizar los sistemas de ayuda en entornos de red, en applets, en páginas JSP, ... El paquete contiene las librerías necesarias, así como documentación y ejemplos. Para configurar el sistema de ayuda de JavaHelp necesitamos configurar unos cuantos ficheros XML (que ya veremos) y escribir los contenidos en sí en formato HTML.

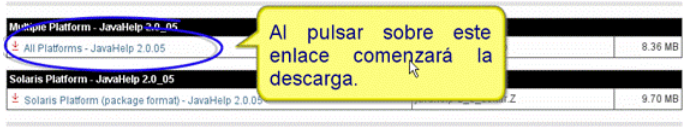
ZONA DE DESCARGA

Si visitas este enlace podrás acceder a la página de descarga de JavaHelp para todas las plataformas. Encontrarás la última versión (que a día de hoy es la 2.0_05). También podremos acceder a versiones anteriores, aunque eso no nos interesa, y a la documentación, que también está incluida en el paquete.

Zona de descarga de JavaHelp

En este enlace puedes encontrar un botón en el que puedes acceder a la descarga de la última versión.

Si pulsamos sobre dicho botón accedemos a otra página, en la que después de aceptar la licencia, nos saldrá una lista de los archivos que podemos descargar y nosotros elegimos la versión multiplataforma.



Una vez realizada la descarga tendremos un archivo llamado **`javahelp-2.0_05.zip`**. Este archivo debemos descomprimirlo y en él encontraremos varias carpetas:

1. **Directorio raíz**: en él podremos encontrar los términos de la licencia, un fichero **`README`** con los últimos errores que corrige la versión y el fichero **`src.jar`** con los fuentes.
2. **javahelp**: Este directorio contiene a la vez otros dos directorios. El directorio **`lib`** es el que realmente contiene la librería del sistema de ayuda. Hay varios archivos JAR que contienen más o menos funcionalidades, nosotros utilizaremos el fichero **`jhall.jar`**, que contiene todas las funcionalidades. El directorio **`bin`** contiene las utilidades para crear bases de datos de búsqueda (**`jhindexer`**) y para realizar búsquedas (**`jhsearch`**).
3. **doc**: Directorio que contiene la documentación de la librería. Podemos encontrar la guía de usuario en PDF y empaquetada como un sistema de ayuda. Te recomendamos que la tengas a mano ya que te será de gran ayuda (valga la redundancia).
4. **demos**: Directorio que a su vez contiene otros directorios con ejemplos de cómo utilizar el sistema de ayuda en nuestras aplicaciones o de cómo crear sistemas de ayuda. Hay una demo que después veremos cómo usarla. Se encuentra en el directorio **`bin`**, se llama **`hsviewer.jar`** y permite visualizar sistemas de ayuda, por lo que nos será de utilidad cuando empecemos a crear el nuestro propio.

Ya tenemos descargada la librería que nos ayudará en la creación de sistemas de ayuda. En los siguientes apartados veremos cómo podemos crear un sistema de ayuda para nuestro caso de estudio del generador de informes del taller informático, para posteriormente incluirlo en dicha aplicación.

Autoevaluación

JavaHelp es una librería desarrollada por:

a) Sun Microsystems
b) Oracle
c) Microsoft
d) Ninguna de las respuestas anteriores es correcta

Comprobar

Creando un sistema de ayuda para nuestro caso de estudio

Imagínate si tuvieses que crear un sistema de ayuda a partir de cero. La tarea sería bastante ardua y las modificaciones que deseáramos realizar también serían costosas. Lo bueno que tienen las librerías para la visualización de sistemas de ayuda es que nos hacen la tarea muy sencilla y las modificaciones a introducir también son muy rápidas. En este apartado veremos cuáles son los pasos necesarios para crear un sistema de ayuda usando JavaHelp para posteriormente ver cómo introducirlo en nuestra aplicación.

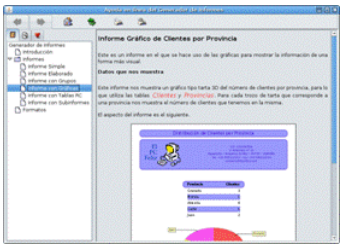
Ya hemos comentado que crear un sistema de ayuda con JavaHelp es bastante sencillo y que se trata de configurar unos ficheros XML y escribir los contenidos en formato HTML. Enumeraremos las tareas que debemos realizar para crear un sistema de ayuda para pasar a describir cada una de ellas:

1. Crear una estructura de directorios adecuada para almacenar el sistema de ayuda.
2. Crear los contenidos en sí, es decir, las diferentes páginas o temas (topics) de que constará nuestra ayuda en formato HTML.
3. Crear el fichero principal de configuración (Helpset), que describirá el aspecto de nuestro sistema de ayuda y dirá dónde encontrar los ficheros que definen cada vista.
4. Crear el fichero mapa, encargado de asignar identificadores a cada tema de ayuda o elemento utilizado en cualquier otro fichero de configuración.

5. Crear el fichero de tabla de contenidos que será el que definirá la vista de tabla de contenidos.
6. Crear el fichero de índice que será el que definirá la vista de índice alfabético. Su creación y definición es exactamente igual que el anterior, por lo que no nos detendremos en el mismo y si necesitas más información no dudes en consultar la guía de usuario.
7. Crear la base de datos de búsqueda si vamos a utilizar la vista de búsqueda.

Como ves son pasos muy bien definidos que iremos desgranando en los siguientes subapartados. Pronto comprobarás que crear un sistema de ayuda es una tarea más fácil de lo que podrías imaginar. Para ello vamos a seguir los pasos que hemos seguido para construir el sistema de ayuda para nuestro caso de estudio del generador de informes para el taller informático, cuyo aspecto es el que te mostramos.

Haz clic sobre la imagen siguiente para verla con claridad.



Autoevaluación

Los temas de ayuda son ficheros escritos en formato:

a) XML
b) PDF
c) HTML
d) JHM

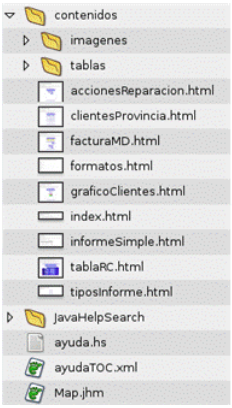
Comprobar

Creación de la estructura de directorios

Aunque este es un paso trivial, es conveniente comentarlo.

Como hemos visto, el sistema de ayuda consta de diferentes ficheros y dependiendo de la extensión de nuestra ayuda éstos pueden llegar a ser muchos. Por tanto es deseable tener bien organizado todo el sistema.

- Lo primero que haremos será crear un directorio, que en nuestro caso llamaremos **`ayuda`**.
- En él colocaremos los ficheros de configuración (que seguidamente vamos a ver cómo construir).
- Dentro de este directorio crearemos otro llamado **`contenidos`**, en el que iremos colocando los temas de ayuda. Nosotros hemos creado dos dentro de éste:
  - uno llamado **`tablas`**, en el que hemos situado los temas referentes a las tablas de la BD y
  - otro llamado **`imágenes`**, en el que hemos almacenado las imágenes que necesitaremos.
- Si nuestro sistema de ayuda fuese más grande, sería conveniente ir creando subdirectorios dentro de este directorio y así tener agrupados jerárquicamente nuestros temas de ayuda.
- Otro directorio llamado **`JavaHelpSearch`**, en el que está la base de datos de búsqueda, que ya veremos cómo crearla. Como es de esperar, este árbol lo situaremos luego bajo el directorio recursos de nuestra aplicación, pero no adelantemos acontecimientos.



En la imagen que acompaña al texto de este apartado puedes ver el árbol de nuestro sistema de ayuda.

Autoevaluación

El árbol de directorios...

a) Puede tener tantos niveles como necesitemos para estructurar bien la información
b) No puede tener más de dos niveles
c) Puede tener los niveles indicados en el fichero de configuración principal
d) Todas son falsas

Comprobar

Creación de los temas de ayuda (topics)

¿Qué son, "físicamente", los temas de ayuda? ¿Cómo los creamos?

Los temas de ayuda no son más que páginas HTML que serán visualizadas en el panel de visualización del sistema de ayuda. Debemos crear tantas páginas HTML como temas tenga el sistema de ayuda.

Siempre debemos utilizar referencias relativas y nunca debemos utilizar referencias absolutas. Al utilizar referencias, debemos utilizar como separador el carácter "`/`" que funcionará en todas las plataformas ya que si utilizamos como

separador el carácter "\", nuestro sistema sólo funcionará correctamente en plataformas Windows.

También es posible crear un enlace en un tema de ayuda que se abra en otra ventana (secondarywindow) o en una ventana emergente (popup). La diferencia es que la primera deberemos cerrarla por medio del aspa que aparece en su barra de título y la segunda no tiene barra de título y desaparecerá en cuanto pulsemos sobre ella o en otro lugar del sistema de ayuda. Por ejemplo, en el sistema de ayuda que hemos compuesto para el taller informático hemos utilizado enlaces con ventanas emergentes (popup) para mostrar las tablas de la BD (que son otro tema de ayuda, pero al que sólo accedemos desde estos enlaces y no desde la tabla de contenidos). Para ello debemos utilizar la etiqueta **<object>** de HTML pasando los parámetros adecuados. Puedes ver todos los parámetros que podemos pasar en la guía de usuario, aunque como puedes comprobar son todos muy descriptivos. Por ejemplo, para crear un enlace a la tabla empleados que se abra en una ventana emergente hemos utilizado el siguiente código HTML:

```
<object classid="java:com.sun.java.help.impl.JHSecondaryViewer">
      <param name="content" value="tablas/tablaEmpleados.html">
      <param name="viewerActivator" value="javax.help.LinkLabel">
      <param name="viewerStyle" value="javax.help.Popup">
      <param name="viewerSize" value="500,230">
      <param name="text" value="Empleados">
      <param name="textFontFamily" value="SansSerif">
      <param name="textFontSize" value="x-large">
      <param name="textFontWeight" value="plain">
      <param name="textFontStyle" value="italic">
      <param name="textColor" value="red">
</object>
```

PARA SABER MÁS
En el siguiente enlace podrás encontrar un tutorial de HTML, por si tus conocimientos sobre este lenguaje flaquean.
Tutorial de HTML 4.0

Autoevaluación

Las referencias que debemos utilizar en los temas de ayuda deben ser...

a) Relativas
b) Absolutas
c) Estrictas.
d) Es indiferente

Comprobar

Creación del fichero principal de configuración

El fichero principal de configuración está escrito en lenguaje XML y tiene la extensión. **hs**, pero no te asustes ya que el número de etiquetas es muy pequeño y nada tiene que ver con los ficheros XML que utilizábamos para diseñar informes.

En este fichero es donde definimos el comportamiento y aspecto que tendrá nuestro sistema de ayuda. En él ponemos título al sistema de ayuda, decimos dónde encontrar el fichero mapa y definimos las vistas que queremos que aparezcan. También indicamos los botones que tendrá la barra de herramientas y el tamaño y posición de la ventana en la que mostramos el sistema de ayuda. En la siguiente imagen puedes observar el fichero de configuración que hemos creado para el sistema de ayuda del taller informático y las secciones en las que está dividido. Como puedes observar su sintaxis es muy sencilla, pero para más detalles deberás consultar la guía de usuario de JavaHelp.

Haz clic sobre la siguiente imagen para ampliarla.

Autoevaluación

En el apartado de presentación del fichero principal de configuración definimos...

a) El tipo de letra del mismo

b) Los elementos de la barra de herramientas
c) Los elementos de la barra de menú
d) Todas las respuestas anteriores son verdaderas

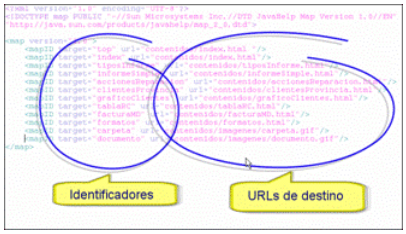
Comprobar

Creación del fichero mapa

Este es otro fichero XML con extensión **.jhm**, pero cuya sintaxis es aún más sencilla que la del anterior. En este fichero asignamos identificadores a cada uno de los temas de nuestro sistema de ayuda. También podemos asignar identificadores a otros elementos que utilicemos en algún otro fichero de configuración, como pueden ser las imágenes que usemos como iconos en la vista de tabla de contenidos.

En la siguiente imagen puedes comprobar la asignación de identificadores que hemos hecho para el sistema de ayuda del taller informático. Puedes ver cómo hemos asignado identificadores a todos los temas de ayuda y también a algunas imágenes que vamos a usar en la vista de tabla de contenidos.

Haz clic sobre ella para ampliarla.



Autoevaluación

El fichero mapa, por convención tiene la extensión...

a) .xml
b) .map
c) .jhm
d) Todas las respuestas anteriores son falsas

Comprobar

Creación del fichero de tabla de contenidos

El fichero de tabla de contenidos (TOC) es otro fichero XML, y seguro que te pregutas para qué sirve.

Define la vista de tabla de contenidos y suele tener extensión **.xml**. Aunque éste no es obligatorio, no se concibe un sistema de ayuda sin la vista tabla de contenidos.

En este fichero lo único que hacemos es estructurar jerárquicamente los temas de ayuda dándole un texto a mostrar en la vista tabla de contenidos. La estructura jerárquica la conseguimos anidando etiquetas **<tocitem>** unas dentro de otras. Para cada entrada podemos definir un icono que aparecerá a la izquierda del texto descriptivo. En la siguiente imagen puedes comprobar el diseño de este fichero para nuestro sistema de ayuda del taller informático. Haz clic sobre ella para ampliarla.

Autoevaluación

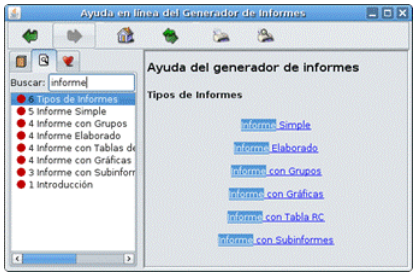
El fichero de definición de la tabla de contenidos suele tener la extensión...

a) .xml
b) .toc
c) .html
d) .jhtoc

Comprobar

Creación de la base de datos de búsqueda

Cuando en nuestro sistema de ayuda mostramos la vista de búsqueda, nos aparece una caja de texto en la que podemos introducir texto a buscar dentro de todos los temas de ayuda. El resultado de la búsqueda nos indicará los temas en los que ha encontrado el texto y el número de incidencias del mismo en cada uno.



Lo primero que hay que hacer es generar la base de datos con los índices para que dicha búsqueda sea posible.

Como recordarás cuando vimos el árbol de directorios del sistema de ayuda para el taller informático, había un directorio llamado **`JavaHelpSearch`** que correspondía a esta base de datos. Este directorio está compuesto por seis ficheros y es generado automáticamente por medio de una utilidad incluida en el paquete JavaHelp.

Dentro del directorio donde descomprimimos JavaHelp (**`<path_JavaHelp>`** a partir de ahora) existe una utilidad llamada **`jhindexer.jar`** (está situada en el directorio **`<path_JavaHelp>/javahelp/bin`**). **Esta utilidad es la que nos permite crear dicha base de datos**. Para ello sólo tenemos que ejecutarla en el directorio raíz de nuestro sistema de ayuda y pasarle como argumentos los directorios que queremos indexar, que serán los que contengan los temas de ayuda. En nuestro caso nos situaremos en el directorio **`ayuda`** y ejecutamos el comando: **`java -jar <path_JavaHelp/javahelp/bin/jhindexer.jar contenidos`**. Una vez ejecutado el comando veremos cómo aparece el directorio **`JavaHelpSearch`**.

Autoevaluación

Respecto a la base de datos de búsqueda podemos afirmar que...

a) No es necesaria para nada, ya que si no está definida se crea en tiempo de ejecución
b) Debemos crearla mediante la utilidad jhsearch
c) Debemos crearla mediante la utilidad jhindexer
d) Debemos crearla mediante la utilidad makeBDSearch

Comprobar

Visualizando el resultado

Llegados a este punto ya tenemos creado nuestro sistema de ayuda para el generador de informes del taller informático. En el siguiente enlace podrás descargarlo y echar un vistazo a todos los ficheros de los que hemos venido hablando.

Descarga el archivo de ayuda

Seguramente estás impaciente por ver el aspecto que tiene el mismo y no puedes esperar a integrarlo en la aplicación. Además, generalmente cuando estamos creando nuestros sistemas de ayuda debemos ir probándolos antes de integrarlos, para alcanzar el aspecto deseado. Para este cometido el paquete JavaHelp nos ofrece un visor de sistemas de ayuda que nos viene muy bien en estos casos.

En el directorio **`<path_JavaHelp>/demos/bin`** existe una aplicación de ejemplo llamada **`hsviewer.jar`** que precisamente realiza este cometido.

Para poder visualizar el sistema de ayuda que acabamos de crear debes descomprimir el fichero **`ayuda.zip`** en un directorio (**`<path_ayuda>`** en adelante) y desde un terminal ejecutar el comando: **`java -jar <path_JavaHelp>/demos/bin/hsviewer.jar -helpset <path_ayuda>/ayuda/ayuda.hs`**.

En el siguiente apartado veremos cómo integrar este sistema de ayuda en la aplicación de generación de informes para el taller informático. Verás que esta tarea es realmente simple.

Autoevaluación

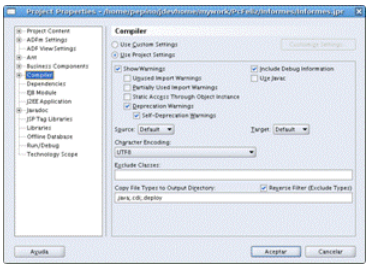
Para visualizar nuestro sistema de ayuda...

a) Debemos integrarlo en nuestra aplicación
b) Sólo podemos visualizarlo en plataformas Linux con la utilidad viewHelp
c) Podemos visualizarlo en todas las plataformas con la utilidad viewHelp
d) Todas las respuestas anteriores son falsas

Comprobar

Integrando el sistema de ayuda en nuestra aplicación

Ya tenemos nuestro sistema de ayuda creado y listo para integrarlo en nuestra aplicación. Vamos a ver cómo utilizar el API que nos ofrece la librería JavaHelp para poder utilizar este sistema de ayuda recién creado.



- Lo primero que debemos hacer es incluir el fichero JAR **`jhall.jar`** en nuestro proyecto como ya sabemos hacer.
- Una vez hecho esto nuestra aplicación ya será capaz de utilizar dicha API.
- Seguidamente hemos colocado el directorio **`ayuda`** que contiene todo nuestro sistema de ayuda en el directorio **`src/recursos`** de nuestro proyecto, que es donde estamos colocando todos los recursos necesarios para nuestra aplicación.
- Debido a que el sistema de ayuda contiene ficheros con diferentes extensiones y la base de datos de búsqueda tiene ficheros que no tienen extensión, en vez de añadir todas estas extensiones a la lista de ficheros que JDeveloper debe copiar al directorio de salida al compilar, hemos optado por la opción inversa. Esto es, decimos qué extensiones no queremos que copie y marcamos la casilla "**Reverse Filter (Exclude Types)**", con lo que conseguimos que se copien todos los ficheros excepto los que tengan las extensiones especificadas en la lista.

Hemos añadido a nuestra aplicación una barra de menú, en la que hemos colocado dos menús.

- Uno llamado **`Fichero`** que sólo contiene la opción **`Salir`**.
- Otro llamado **`Ayuda`** que contiene
  - la opción **`Ayuda`**, que lanzará nuestro visor del sistema de ayuda y
  - otra llamada **`¿Qué es ...?`** , que al elegirla cambiará el puntero del ratón añadiéndole una interrogación y cuando pulsemos sobre un componente de nuestra aplicación nos mostrará el tema de ayuda asociado al mismo.

Hemos añadido un nuevo método a la clase **`GUIGeneradorInformes`** llamado **`crearSistemaAyuda`** que es el encargado de realizar todas estas tareas. Su código es el que puedes encontrar en el siguiente cuadro, y que vamos a comentar a continuación.

```
/**Creamos el Sistema de Ayuda de nuestra apliación*/
private void crearSistemaAyuda() {
        HelpSet hs = null;
        HelpBroker hb;
      // Buscamos el fichero .hs
        String helpHS = "recursos/ayuda/ayuda.hs";
        ClassLoader cl = this.getClass().getClassLoader();
        try {
            URL hsURL = HelpSet.findHelpSet(cl, helpHS);
            hs = new HelpSet(cl, hsURL);
        } catch (Exception e) {
            System.out.println("HelpSet " + e.getMessage());
        }
      // Creamos el objeto HelpBroker
        hb = hs.createHelpBroker();
      //Habilitamos la tecla F1 para nuestra GUI
        hb.enableHelpKey(getRootPane(), "index", hs);
      //Definimos la ayuda contextual
        hb.enableHelp(jRB_InformeSimple, "informeSimple", hs);
        hb.enableHelp(jRB_AccionesReparacion,"accionesReparacion", hs);
        hb.enableHelp(jRB_ClientesProvincia, "clientesProvincia", hs);
        hb.enableHelp(jRB_GraficoClientes, "graficoClientes", hs);
        hb.enableHelp(jRB_TablaRC, "tablaRC", hs);
        hb.enableHelp(jRB_FacturaMD, "facturaMD", hs);
```

```
            hb.enableHelp(jRB_Pdf, "formatos", hs);
            hb.enableHelp(jRB_Xls, "formatos", hs);
            hb.enableHelp(jRB_Csv, "formatos", hs);
            hb.enableHelp(jRB_Html, "formatos", hs);
            hb.enableHelp(jRB_Rtf, "formatos", hs);
            hb.enableHelp(jRB_Odt, "formatos", hs);
            hb.enableHelp(jCB_Exportar, "formatos", hs);
          //Lanzamos la ayuda al pulsar sobre el Menú Ayuda
            hb.enableHelpOnButton(jMI_Ayuda, "index", hs);
          //Lanzamos la ayuda después de elegir el componente sobre el que
          //queremos ayuda
            jMI_QueEs.addActionListener(new CSH.DisplayHelpAfterTracking(hb));
    }
```

Como puedes observar, lo primero que debemos hacer es crear un nuevo objeto **HelpSet**, pasándole la URL al fichero de configuración principal (HelpSet) y un objeto **ClassLoader**. La URL la obtenemos mediante un método estático de la clase **HelpSet** llamado **findHelpSet**. Seguidamente debemos crear un objeto **HelpBroker** que es el encargado de gestionar el visor del sistema de ayuda. Esto lo hacemos mediante el método **createHelpBroker** de la clase **HelpSet**. Una vez hemos creado dicho objeto debemos decirle cómo debe comportarse cuando se pulse la tecla F1. Para ello utilizamos los métodos **enableHelpKey** y **enableHelp** para decir qué tema de ayuda se debe mostrar al pulsar dicha tecla sobre el componente que pasamos como argumento. Como ves, para indicar el tema a mostrar utilizamos una cadena con el identificador del tema que definimos en el fichero mapa. Por último hemos añadido un **ActionListener** al elemento de menú **jMI_QueEs** encargado de mostrar un tema de ayuda asociado a un componente después de pulsar sobre él. En los siguientes enlaces podrás descargarte el fichero JAR con la tercera versión del generador de informes totalmente funcional y también el proyecto para JDeveloper de la misma.

Descarga el fichero jar

Descarga aquí el proyecto

Como has podido observar, integrar el sistema de ayuda en nuestra aplicación nos ha supuesto muy poco esfuerzo gracias a la librería JavaHelp. Esperamos que esta unidad te haya sido de utilidad y que la asimilación de los conceptos expuestos en la misma haya sido una tarea amena.

Autoevaluación

Para integrar la ayuda en nuestra aplicación lo primero que debemos crear es...

a) Un obeto HelpSet
b) Un objeto Help.
c) Un objeto HelpBroker
d) Todas las respuestas anteriores son falsas

Comprobar

En el siguiente enlace encontrarás un resumen visual del tema. No tienes más que hacer clic sobre la imagen para que la presentación funcione.

>