UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

“ESTUDIO DE LA TECNOLOGÍA DE JAVAFX SCRIPT PARA EL DESARROLLO DE APLICACIONES ENRIQUECIDAS DE INTERNET”

T E S I S

P A R A O B T E N E R E L T Í T U L O D E

I N G E N I E R O E N C O M P U T A C I Ó N

P R E S E N T A :

J U L I Á N C A S T I L L O S O T O

ASESOR: M. EN C. JESÚS HERNÁNDEZ CABRERA

SEPTIEMBRE DE 2010

AGRADECIMIENTOS

Les dedico este trabajo a mis padres y hermanas
que tanto me han apoyado y que siempre han estado conmigo.

Le doy gracias también a la Universidad
que me ha permitido alcanzar tantas metas y llegar a donde estoy.

También agradezco a mis amigos y compañeros de generación con los
que he trabajado durante la carrera y he hecho amistades para siempre.

Estudio de la Tecnología de JavaFX Script para el Desarrollo de Aplicaciones Enriquecidas de Internet.

CAPÍTULO 1 Introducción a las Tecnologías de Internet y Java

CAPÍTULO 2 La Tecnología JavaFX

CAPÍTULO 3 Sintaxis de JavaFX Script

INTRODUCCIÓN

Actualmente el desarrollo y crecimiento de Internet se ha dado de una manera exponencial, donde los usuarios que acceden a la red cada vez demandan más aplicaciones y contenidos dinámicos e interactivos como video, audio, animaciones, etc.

Han surgido nuevas tecnologías enfocadas al desarrollo de contenidos gráficos, a lo que se le ha denominado últimamente como RIA (Aplicaciones de Internet Enriquecidas) siendo el siguiente salto para la Web y la tendencia que han tomado varias empresas para implementar estas tecnologías.

El lenguaje HTML usado para crear páginas web se ha venido combinando con otras varias tecnologías para poder generar todos estos contenidos y efectos visuales, otra ventaja de estas nuevas tecnologías es que no generan tanta carga e intercambio de datos entre un cliente y el servidor ya que toda la aplicación se carga la primera vez y se puede interactuar más con el usuario; así solo se produce comunicación cuando se necesitan datos externos.

Como parte de esta evolución de nuevas aplicaciones Web, Sun Microsystems ha anunciado una nueva familia de tecnologías llamada JavaFX para mostrarse como una propuesta ante la demanda de este tipo de aplicaciones. Apenas se lanzó la primera versión de estos productos y se compone básicamente de dos partes:

- JavaFX Mobile .- Es una completa y nueva plataforma que servirá para la construcción de aplicaciones para dispositivos portátiles y solo se encuentra disponible mediante licencias para los fabricantes de teléfonos móviles y aquellas empresas que también deseen desarrollar nuevo software para dispositivos portátiles.
- JavaFX Script .- Es un nuevo lenguaje declarativo que esta basado principalmente en la programación orientada a objetos y esta enfocado a la creación de interfaces gráficas complejas y robustas, ya sean aplicaciones de escritorio o que se ejecuten dentro de un navegador Web. Una de sus principales fortalezas es que tiene una alta integración con el lenguaje de programación JAVA, lo cual permite explotar el API

de JAVA y así poder construir aplicaciones robustas y al mismo tiempo tener una capa o vista rica en recursos multimedia con una alta integración a la funcionalidad de la aplicación.

También se prevé que en futuras versiones se lancen nuevos productos aparte de estos dos paquetes como aplicaciones y servicios para la televisión, discos Blu-ray y otras plataformas.

El trabajo que se desarrollará en esta tesis es una investigación sobre este nuevo lenguaje de programación (JavaFX Script). Durante el primer capítulo se revisarán las diferentes tecnologías de internet, la plataforma Java, las características de las RIA y otras tecnologías similares a JavaFX.

En el segundo capítulo se revisará la arquitectura JavaFX y se mencionarán las diferentes plataformas y dispositivos donde las aplicaciones JavaFX pueden ejecutarse. Por último se mencionaran las herramientas que existen para generar código con JavaFX Script tanto para desarrolladores y diseñadores.

Durante el tercer capítulo se revisará y se analizará la estructura y sintaxis del lenguaje JavaFX Script, mediante algunos ejemplos se mostrará la forma de usar y declarar objetos, variables, funciones, clases y diversos tipos de expresiones.

En el cuarto capítulo se verá cómo construir interfaces gráficas, dibujando figuras, usando imágenes para aplicar algunos efectos de luz, degradados de color, eventos de mouse sobre gráficos y un ejemplo de animación.

El quinto capítulo está centrado en el desarrollo de una aplicación tipo RIA usando y aplicando varios de los temas y conceptos vistos principalmente en los capítulos tres y cuatro. De esta manera se podrá descubrir parte del potencial de este nuevo lenguaje de programación.

CAPÍTULO 1

INTRODUCCIÓN A LAS TECNOLOGÍAS DE INTERNET Y JAVA

1.1 La Web (World Wide Web) e Internet

Internet la podemos definir como una gran red lógica mundial, que esta compuesta por un gran conjunto de redes interconectadas entre si mediante el uso de una serie de protocolos TCP/IP. Todo empezó como un proyecto patrocinado por el Departamento de Defensa de Estados Unidos en la década de los 60's y que fue desarrollado principalmente en el MIT (Instituto Tecnológico de Massachussets), más tarde en 1969 fue establecida la primera red entre universidades llamada *ARPANET* la cual siguió creciendo en los años siguientes. Tiempo después surgieron nuevas redes y protocolos de comunicación, naciendo así lo que ahora conocemos como Internet.

Otro hecho no menos importante ha sido la introducción de la *World Wide Web* a principios de los años 90's, que a grandes rasgos es un servicio o un sistema de documentos de hipertexto entrelazados y que se pueden acceder mediante Internet; así los usuarios de Internet pueden localizar páginas web y ver información, además de diferentes contenidos multimedia de cualquier tipo y tema.

Todo esto ha significado un gigantesco avance en las telecomunicaciones y ha propiciado la globalización de los medios, por ejemplo, ahora desde casi cualquier computadora con acceso a Internet una persona puede comunicarse e intercambiar información con otra desde cualquier punto del planeta. Hoy en día se puede tener acceso a un gran cúmulo de información y conocimientos, además que muchas empresas, universidades, gobiernos, personas, etc. están apostando por esta nueva forma de comunicación por el gran alcance que tiene y potencial a futuro.

Esta tendencia ha venido a dar paso al uso de aplicaciones remotas, por lo tanto ahora se pueden tener aplicaciones que pueden ser accedidas por millones de computadoras en todo el mundo, esto ha dado lugar al surgimiento de nuevas tecnologías y lenguajes de programación para su desarrollo.

1.2 El lenguaje de programación Java

1.2.1 Introducción

Java es un lenguaje de programación Orientado a Objetos, ha sido tal su impacto y aceptación que podemos verlo hoy en día en una amplia gama de aplicaciones: De escritorio, aplicaciones para teléfonos celulares y dispositivos móviles, grandes aplicaciones empresariales y en servidores web.

El lenguaje de programación Java nació como un proyecto de Sun Microsystems en 1991; cuando Sun patrocinó un proyecto interno con el fin de crear aplicaciones para electrodomésticos inteligentes, era un lenguaje basado en C++ que al principio James Gosling su creador le llamó OAK. Poco después se le cambió el nombre a Java, y en 1993

con el surgimiento y auge de la World Wide Web, Sun vislumbró el gran potencial que tenia Java con la web y apoyó firmemente su desarrollo y mejora haciendo su anuncio oficial en mayo de 1995, poco después en 1996 se liberó la primera versión: JDK (Java Development Kit) 1.0, actualmente esta disponible la versión: Java SE 6 que fue nombrada así para diferenciarla de las otras plataformas de Java disponibles Java EE para aplicaciones empresariales y Java ME para aplicaciones portátiles.

El JDK es un software que provee las herramientas necesarias para el desarrollo de programas en Java e incluye el entorno de ejecución conocido como JRE (Java Runtime Enviroment) que se compone la Máquina Virtual de Java (que veremos más adelante) y librerías para el desarrollo de software, el compilador de Java, ejemplos de código y una amplia API (Aplication Program Interface) que es código ya escrito y reutilizable para el desarrollo de programas Java.

Las principales características de Java son: la capacidad de que sus programas se pueden ejecutar desde cualquier computadora que tenga la Máquina Virtual de Java, por lo cual, el código Java es independiente a la plataforma donde se este ejecutando y al Sistema Operativo, tiene un entorno más seguro de ejecución, es de distribución libre y se pueden ejecutar pequeñas aplicaciones dentro de los navegadores llamadas applets.

JavaFX comparte y toma muchas características de Java, también pueden interactuar entre sí combinando código, por eso es importante saber sobre este lenguaje.

1.2.2 Características de Java

A continuación mencionaremos las características de Java las cuales lo convierten en un lenguaje bastante robusto y exitoso:

Es un lenguaje *Orientado a Objetos*, este es un paradigma de programación o un enfoque sobre como estructurar y modelar la construcción de aplicaciones, tal como lo indica su nombre se trata de buscar cómo resolver un problema o necesidad basándose en modelar e identificar entidades u objetos; estos objetos tienen propiedades y

comportamientos que nos ayudan a modelar sistemas. Los objetos son creados a partir de archivos llamados clases que es donde se definen y donde están sus atributos y métodos, también se usan diferentes técnicas que permiten establecer relaciones especiales entre objetos como son: la herencia, el polimorfismo, el encapsulamiento, abstracción. La Programación Orientada a Objetos permite un desarrollo más rápido, modular y funcional.

Java es un lenguaje *Portable*, gracias a su Máquina Virtual los códigos de java pueden ejecutarse en varios Sistemas Operativos y diferentes arquitecturas que tengan instalada la Máquina Virtual, esto quiere decir que una vez escrito un programa, puede ejecutarse en cualquier dispositivo (tal como afirma Sun). Por este motivo cuenta con un gran éxito es aplicaciones y servicios Web.

También se dice que Java es un *Lenguaje Dinámico*, ya que las clases y librerías se cargan en cuanto se les necesite. Por otro lado hace un manejo automático de punteros, así se trata de abstraer esta lógica. Los punteros son variables que hacen referencia a una región de memoria, con este manejo automático el programador no tiene que preocuparse por el uso de la memoria a la hora de trabajar con variables y otras estructuras de datos.

Java es un *Lenguaje Seguro*, fue desarrollado pensando en aplicaciones web, se le añadieron capas de seguridad para poder evitar la ejecución de códigos maliciosos que frecuentemente circulan por Internet, para así evitar daños en los sistemas. También se lleva a cabo un proceso de verificación en la Máquina Virtual de los *bytecodes* que son generaros por el compilador.

Es un *Lenguaje Eficiente* en el uso de los recursos, ya que cuando un programa se encuentra en tiempo de ejecución solo crece lo necesario, enlazando los tipos de datos y estructuras dinámicamente durante la ejecución.

Java cuenta con un *Recolector de Basura*, este es un mecanismo que se ejecuta como una subrutina en el momento en que se esta ejecutando un programa y sirve para liberar memoria. Al momento de que se crean objetos en la ejecución de un programa estos tienen un espacio asignado en la memoria, el recolector de basura tiene la función de inspeccionar estos espacios de memoria y decidir cuales son todavía útiles, cuando un

objeto que reside en la memoria ya no tienen ninguna referencia hacia él se convierte en candidato para que el recolector de basura lo elimine; el recolector se ejecuta durante ciertos intervalos de tiempo en forma automática sin intervención del programador, por lo tanto esto ocurre de un forma transparente para el programador. Este proceso puede consumir cierto tiempo de procesamiento al ser invocado, por consecuencia el único inconveniente con este mecanismo es poder determinar en que momento debe ser ejecutado.

Otra característica de Java es la capacidad de llevar a cabo simultáneamente varias tareas lo que de conoce como *Multi-Threaded* o *hilos múltiples*. Un hilo también conocido como *proceso ligero*, lo podemos definir como un pequeño proceso que en conjunto con otros hilos forma parte de un proceso más grande el cual es un programa que se encuentra en ejecución; los hilos tienen la característica de que comparten recursos con otros más como la memoria, el tiempo de procesamiento, acceso a datos o el acceso a algún archivo que se encuentre abierto. Para poder realizar la ejecución concurrente de hilos, estos poseen varios estados: listo, en ejecución y bloqueado; de esta manera existe un ciclo en el cual los hilos pueden convivir hasta que el proceso termina, a continuación mencionamos como se lleva a cabo:

- *Creación*.- Cuando se crea un nuevo proceso este genera un hilo para este proceso, inmediatamente pasa al estado de *Listo*.
- *Listo*.- En este estado el hilo es candidato a ejecutarse y solo esta en espera para que otro hilo finalice su ejecución o entre al estado de *Bloqueo*.
- *Ejecución*.- El hilo se encuentra ejecutándose y dispone de todos los recursos mientras no finalice su ejecución o pase al estado de *Bloqueo*.
- *Bloqueo*.- Ocurre cuando un hilo que se esta ejecutando se detiene y permanece suspendido; esto puede deberse porque hay otro hilo con una prioridad mayor y esta solicitando su ejecución, o puede haber pasado que se haya terminado su tiempo para ejecutarse. Para salir del estado de bloqueo se requiere que otros hilos notifiquen que han dejado de ocupar los recursos y así el hilo bloqueado puede cambiar al estado de *Listo* otra vez.
- *Finalización*.- Es cuando el hilo termina por completo su ejecución.

Esta característica de java es vital en las aplicaciones web, por ejemplo al tener una aplicación corriendo en un servidor, la aplicación debe ser capaz de atender las peticiones que hacen varios clientes a la vez y que pueden suceder al mismo tiempo. Las tecnologías de Java para Web implementan este manejo de concurrencias.

1.2.3 Ejecución y Compilación

Para ejecutar un programa en Java generalmente pasa por las siguientes fases antes de poder ser ejecutado:

1. En primer lugar el programa es creado en un editor o una interfaz de desarrollo y es salvado en un dispositivo de almacenamiento como un disco y se guarda con la extensión *.java*.
2. El código fuente pasa por un proceso de compilación, y genera un código de bytes o *bytecode* que después será interpretado al momento de la ejecución, es guardado en disco con la extensión *.class*.
3. Los archivos *.class* pasan por un proceso de carga, lo que quiere decir que los *bytecodes* son almacenados en la memoria.
4. Al momento que se van cargando los *bytecodes*, se verifica su estructura así como el que cumplan con los criterios de seguridad de java.
5. Finalmente los *bytecodes* son interpretados en la *Máquina Virtual de Java*, los que posteriormente son traducidos a código máquina nativo donde son ejecutados.

Veremos como funciona un compilador y más adelante abordaremos el tema de la *Máquina Virtual de Java*, que es la parte medular de toda la tecnología Java y JavaFX.

Un Compilador es un programa que analiza y traduce el código de un lenguaje en otro código intermedio, el resultado de la traducción es lo que se conoce como lenguaje objeto. Generalmente se genera un código máquina específico, pero en el caso de los compiladores Java el resultado es un código llamado *bytecode* que después será

interpretado en la máquina virtual, el compilador de Java que viene incluido en el JDK se llama “javac”. El proceso que sigue un compilador es el siguiente:

1. Análisis Léxico.- es esta fase se analiza carácter por carácter del código fuente, verificando que sean validos y los agrupa en secuencias de caracteres llamadas *tokens*, que son caracteres con relación entre sí, de tal forma que se identifican palabras reservadas del lenguaje y otras que no como nombres de constantes y sus valores.
2. Análisis Sintáctico.- Se verifica el orden en que son introducidos los *tokens*, verificando así la sintaxis de a cuerdo al orden permitido por el lenguaje. También se verifica el orden jerárquico de los operadores, dando como resultado un árbol sintáctico.
3. Análisis Semántico.- Se comprueba que haya una coherencia de las instrucciones, y que los tipo de datos relacionados a un operador sean posibles y compatibles entre sí. Esto genera un árbol semántico.
4. Generación de Código intermedio.- Se realiza una representación intermedia del programa fuente, la cual debe ser fácil de traducir y producir al programa objeto.
5. Optimización de Código.- En esta fase se optimiza el código para mejorar el tiempo de ejecución del programa; sin embargo, los compiladores de Java no realizan en este momento la optimización, utilizan otra técnica llamada *Ejecución en Tiempo de Compilación (JIT)* que realiza la JRE, esta técnica la mencionaremos más adelante.
6. Generación de Código.- Finalmente se genera el código objeto.

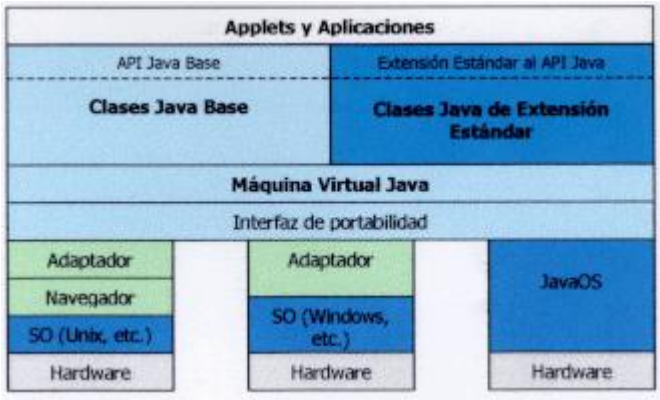
El proceso de compilación se apoya en una tabla de símbolos, que es una estructura en donde se almacenan todos los identificadores encontrados en las fases de análisis, también hay un manejo de errores para cada fase de la compilación.

1.2.4 La Máquina Virtual de Java (JVM)

Una máquina virtual es un software creado con el objetivo de simular el comportamiento de una máquina real que corre generalmente sobre un Sistema Operativo. Las máquinas virtuales son una excelente opción para la portabilidad, seguridad y abstracción del software que se ejecuta sobre ellas.

La *Máquina Virtual de Java (JVM)* es el software donde se interpretan y ejecutan los *bytecodes* generaros en la fase de compilación, por eso es importante saber como esta estructurada ya que es el núcleo de toda la Tecnología Java.

La JVM implementa una capa de software que oculta los detalles de la plataforma donde esta instalada, para llevar a cabo la portabilidad de Java se cuenta con dos capas más: un adaptador que es dependiente del Sistema Operativo y es la parte que se debe de escribir para que la Máquina Virtual pueda acceder y interactuar con el Sistema Operativo, una interfaz de portabilidad que si es independiente al Sistema Operativo. Estas dos capas son el soporte para que pueda ejecutarse la API de Java, las aplicaciones y applets.

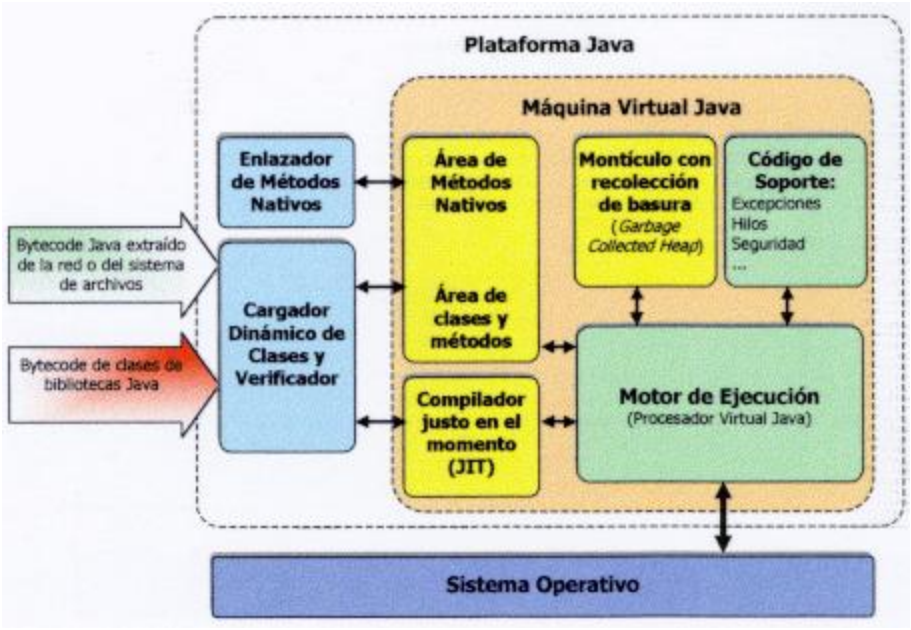


El código intermedio Java "*bytecode"* solo puede ejecutarse sobre la JVM, por lo tanto, no se ejecuta directamente sobre el procesador del dispositivo físico, esto lo hace en un procesador virtual de la JVM.

La Plataforma Java o también conocida como Sistema en Tiempo de Ejecución esta integrada por varios componentes, aquí se reciben los *bytecodes* donde son tratados y verificados para que se ejecuten en el Sistema Operativo nativo. Los componentes de un Sistema en Tiempo de Ejecución son:

- Motor de ejecución.- Aquí se ejecutan los *bytecodes* por medio de un procesador virtual; se utiliza una técnica llamada JIT "generación de código justo en el momento" que consiste en una compilación de los métodos contenidos en el *bytecode* a código nativo, de esta manera las instrucciones se traducen sólo la primera vez y pueden utilizarse varias veces a lo largo de la ejecución del programa, mejorando la velocidad y rendimiento.
- Manejador de memoria.- Es la parte encargada de administrar la memoria que es asignada para los objetos que se crean, estructuras y otros tipos de datos, aquí se ejecuta el mecanismo del recolector de basura.
- Manejador de errores y excepciones.- Su función es poder atrapar errores y excepciones, o lanzar las excepciones que se generan durante la ejecución de un programa, una excepción es un comportamiento extraño o inesperado del programa que puede ser tratado para que continué la ejecución. Generalmente las excepciones tienen asociado un manejador que puede ser un código o una notificación, si no hay asociado un manejador se notifica el error y termina la ejecución del programa.
- Soporte de métodos nativos.- Puede ser que los *bytecodes* requieran para su ejecución métodos o librerías que no son propias de Java, en este caso el Sistema en Tiempo de Ejecución ofrece el código necesario para cargar dinámicamente estos métodos nativos al momento que se requiera y adecuarlos al formato de la Máquina Virtual de Java.
- Interfaz multihilos.- Es la encargada de proporcionar el soporte para la ejecución concurrente de hilos.
- Cargador de Clases.- Se leen los archivos .class y coloca los códigos de *bytecodes* en la memoria, después se verifica la validez y estructura de los códigos bajo las normas de seguridad de Java.

- Administrador de seguridad.- Lleva a cabo la verificación de las clases que han sido cargadas, una de sus tareas es tratar de evitar la ejecución de códigos maliciosos. También tiene mecanismos para restringir el acceso a los recursos del sistema.



1.2.5 Plataformas de Java: Java SE, Java EE y Java ME

A partir de la liberación del primer JDK de Java han surgido y evolucionado varias tecnologías. El rápido crecimiento de Java como lenguaje lo ha llevado a extenderse a un muy variado tipo de aplicaciones, como aplicaciones de escritorio, interfaces gráficas, aplicaciones web, grandes aplicaciones empresariales, aplicaciones móviles, etc. Esto ha llevado al desarrollo de tres plataformas: Java SE, Java EE y Java ME.

Java SE (Java Edición Estándar). Es la especificación estándar de Java para desarrollar aplicaciones de propósito general. Lo conforman la Máquina Virtual de Java y librerías estándar.

Java EE (Java Edición Empresarial). El primer SDK de la versión empresarial se libero en el 2001, esta orientado hacia el desarrollo aplicaciones empresariales donde se busca una estructura modular, distribuida y escalable. Esta plataforma tiene su base en la edición estándar y agrega nuevas librerías o API's para servicios como: conectividad con bases de datos, servicios de mensajería, servicios web, invocación remota de métodos, etc. Este tipo de aplicaciones suelen ser ejecutadas en equipos de grandes capacidades a través de servidores de aplicaciones. Un servidor de aplicaciones es un software que ejecuta la mayor parte de un aplicación dentro del modelo Cliente-Servidor, lleva a cabo la lógica de negocio y la manipulación o acceso a datos, de esta manera brinda servicios de la aplicación a las computadoras usuario.

Java ME (Java Edición Micro). Es la especificación para el desarrollo de juegos y aplicaciones en dispositivos móviles como celulares o PDA (Asistentes Personales Digitales). A pesar de la rápida evolución de estos dispositivos, tienen muy limitados sus recursos como son la memoria y la capacidad de procesamiento, por lo tanto, se ha optimizado el entorno de ejecución donde corren sus aplicaciones minimizando el consumo de recursos y creando así una pequeña máquina virtual y librerías especializadas.

1.3 Aplicaciones Web

Una aplicación es un programa que esta diseñado para que un usuario pueda realizar alguna tarea determinada, una aplicación actúa como una interfaz entre el usuario y la computadora. Las aplicaciones Web son aquellas en las cuales el cliente interactúa con un servidor web a través de Internet mediante un navegador, para solicitar un recurso.

1.3.1 Arquitectura Cliente-Servidor

Muchas aplicaciones Web se basan en la arquitectura Cliente-Servidor, este modelo en un principio exigía que los clientes soportaran ciertas tecnologías para interactuar con las aplicaciones del servidor; sin embargo, ahora con las aplicaciones basadas en la Web,

para el cliente sólo es necesario tener un navegador que interprete HTML y sea capaz de entender otros lenguajes interpretados (Script), de esta manera es mas fácil el acceso a miles clientes sin la necesidad de instalar algún software en específico y sólo es necesario tener actualizado nuestro navegador.

Los intérpretes son programas que realizan una traducción de un lenguaje de alto nivel a uno de bajo nivel (código máquina), esta traducción se hace instrucción por instrucción al momento de la ejecución y a medida que se va requiriendo.

El navegador es un programa que actúa como una interfaz para el usuario o cliente y es el encargado de realizar las peticiones a un Servidor Web, también es capaz de interpretar la respuesta de este mediante el protocolo HTTP (Hipertext Transfer Protocol), y así el navegador muestra la información y recursos en forma de documentos de hipertexto o páginas HTML.

Por otro lado, el Servidor Web consiste en un programa corriendo sobre una máquina, el cuál implementa el protocolo HTTP para la transferencia de hipertextos o páginas Web, por lo tanto, su función es atender las peticiones de los clientes en cualquier momento, dependiendo la petición que se realice se devolverá un documento HTML estático o se ejecutará un programa para generar a lo que se le conoce como una página dinámica. De esta manera tenemos aplicaciones corriendo en los dos lados y podemos distinguirlas como sigue:

- Aplicaciones del lado del cliente. Estas suelen ser lenguajes interpretados como es el caso de JavaScript que nos ayuda para validaciones y generar algunas partes dinámicas dentro de la página, también puede haber pequeñas aplicaciones ejecutándose en los propios navegadores como es el caso de los applets de Java y otros lenguajes que pueden soportados con la ayuda de algunos plugins.
- Aplicaciones del lado del servidor. El servidor al poder ser un equipo de mayores capacidades puede manejar procesos más pesados y así ejecutar el grueso de la aplicación como el procesamiento de los datos y gestión de

bases de datos, de esta manera solo se devuelven al usuario las páginas con la información ya procesada.

Algunos de los lenguajes utilizados para el desarrollo de Aplicaciones Web son: CGI, PHP, Java, Perl, .NET.

1.3.2 Modelo Vista Controlador

Es un patrón de diseño de software que fue introducido por primera vez en 1979 y que recientemente se ha estado utilizando mucho para la construcción de aplicaciones entre ellas las Web, el objetivo de este patrón de diseño es poder estructurar y separar los componentes de una aplicación básicamente en tres partes manejables: la vista o interfaz de usuario, el modelo y el controlador. A continuación las describiremos:

- *Vista*. Es la parte que actúa como la interfaz para el usuario, es donde se muestran los datos e información procesada por el servidor, y es también en donde el usuario puede interactuar con la aplicación.
- *Modelo*. Esta representa y encapsula la lógica de negoció de la aplicación, es donde normalmente se llevan a cabo los procesos para devolver algún tipo de información, también se puede tener acceso a un Sistema de Base de Datos.
- *Controlador*. Actúa como un manejador entre las peticiones que hace el usuario a través de la vista y las delega si es necesario a capa del modelo, por lo tanto cumple con la tarea de manejar el flujo de la aplicación.

En el caso de las aplicaciones web el usuario genera solicitudes HTTP que son atendidas por controlador y este decide el flujo a seguir, en donde se puede devolver una respuesta inmediata a la vista o también pueden haber manejadores que estén ligados a un modelo donde se llevan a cabo los procesos internos del negocio y así la información es devuelta usualmente del controlador a la vista.

De esta manera la aplicación es más manejable y fácil de mantener ya que podríamos enfocarnos en una capa en particular. Existen otros patrones del diseño del software pero este es uno de los más usados actualmente y ha tenido una gran aceptación, incluso al grado de que han surgido muchos Frameworks de varios lenguajes de programación basados en este patrón y que mencionaremos a continuación.

1.3.3 Frameworks

Dentro del diseño y desarrollo de aplicaciones existe también el concepto de framework, que es una estructura de software que nos brinda una base o esqueleto con componentes personalizables e intercambiables para poder desarrollar una aplicación. En otro punto de vista es como una pequeña aplicación genérica a la cual le podemos ir agregando piezas y también configurarlas según las necesidades de la aplicación a desarrollar.

El objetivo de los frameworks es ofrecer una estructura para el desarrollo de software más rápida y bajo una metodología, reutilizando código ya existente. En relación con las aplicaciones web los frameworks tienen componentes reutilizables como archivos de configuración, descriptores y código fuente.

Pueden estar orientados hacia el desarrollo de algún modulo en especifico, como puede ser la parte la vista, soporte para interactuar con una base de datos, la parte del controlador y la navegación, etc. También pueden seguir ciertas metodologías o enfoques para desarrollar aplicaciones como pueden ser algunos paradigmas de la programación: orientada a objetos, aspectos, eventos, etc.

1.4 Las RIA (Aplicaciones de Internet Enriquecidas)

Es un nuevo término que es aplicado para aquellas aplicaciones Web, móviles y de escritorio de nueva generación y que vienen en camino, combinan las características de las

tradicionales aplicaciones web y otras aplicaciones de escritorio; con el objetivo de poder combinar e integrar nuevas capacidades multimedia con entornos Web sin la necesidad de utilizar programas externos a la aplicación, de esta manera se descarga toda la aplicación de una sola vez, por eso se dice que son enriquecidas. La ventaja de esto es que el usuario tiene una mayor interacción con la aplicación sin tener la necesidad de una comunicación tan continua con el Servidor Web como es el caso de las tradicionales aplicaciones Web.

1.4.1 Historia de las RIA

El término fue introducido por Macromedia (ahora fusionada con Adobe) en el 2002 y a partir de ahí ha tenido diferentes denominaciones. Anteriormente en las clásicas aplicaciones Web basadas en el modelo Cliente-Servidor el cliente solo podía interactuar con páginas HTML estáticas y si se solicitaba o requería nueva información se generaban nuevas peticiones hacia el servidor para obtener el recurso, lo cual provocaba una alta tasa de intercambio de datos o información.

Poco tiempo después el lenguaje HTML se empezó a combinar con otros lenguajes interpretados como JavaScript, ActionScript, VBScript, etc., para añadir mayor capacidad de interacción entre el usuario y la aplicación, a lo que se le conoce como DHTML o Dynamic HTML.

Recientemente al mejorar los servicios de Internet, las comunicaciones y al aumentar el número de usuarios que se conectan a Internet, las grandes empresas como Microsoft, SUN, Adobe, etc. están apoyando fuertemente al desarrollo de las RIA, ya que los usuarios cada vez demandan mejores y más complejas aplicaciones con mejores experiencias y efectos visuales. De tal manera que se busca poder hacer más independiente la capa de la vista y optar por una comunicación asíncrona con el servidor.

1.4.2 Características y Ventajas

Las clásicas aplicaciones Web suelen correr sobre un navegador lo cual limita y complica el desarrollo de aplicaciones más complejas, las RIA cuentan con las siguientes características:

- ✓ Son aplicaciones ricas, lo cual quiere decir que tienen capacidades de audio, video, voz, etc., así como mejores animaciones y efectos visuales.
- ✓ Generalmente no necesitan de otros programas externos a la aplicación para poder desplegar su contenido, como reproductores de audio o video, ya que cuentan con sus propias herramientas.
- ✓ Cuando se descarga la aplicación se hace de forma completa, de esta manera existe una mayor interacción con el usuario, también existe una mayor capacidad para que el cliente realice sus tareas.
- ✓ Las aplicaciones son más eficientes e inteligentes, permitiendo hacer cosas más complejas de lo que puede hacer el HTML, como controles gráficos deslizables para manipular, cambiar y calcular datos, sin la necesidad de enviar datos al servidor para su procesamiento.
- ✓ Son independientes al navegador y al Sistema Operativo, son multiplataforma, evitan el problema de la compatibilidad de diferentes navegadores mediante el uso de Frameworks.
- ✓ No requieren de una instalación y solo necesitan estar actualizados sus entornos de ejecución para poder ejecutarse.
- ✓ La comunicación con el Servidor Web se lleva a cabo de una forma asíncrona, mejorando la carga que se hace sobre la red, siendo así aplicaciones más eficientes.
- ✓ Son más seguras ya que es menos probable que sean afectados por algún virus informático.

1.4.3 Desventajas

A pesar de las ventajas de las RIA y sus capacidades graficas, tienen algunos inconvenientes para poder ser soportadas y ejecutadas. A continuación listamos algunos:

- Al ejecutarse en sus propios entornos como es el caso de las máquinas virtuales tienen restringido el acceso a los recursos del Sistema Operativo.
- En algunos casos los usuarios tienen desactivada la ejecución para los lenguajes interpretados en sus navegadores, por tal motivo las RIA pudieran no funcionar correctamente.
- Los motores de búsqueda de páginas Web no son capaces de indexar el contenido de estas aplicaciones.
- Dependen del ancho de banda de la red, al ser aplicaciones en algunas veces más pesadas que una aplicación clásica algunos usuarios con módems Dial-UP no podrán visualizarlas o tener una buena interacción con la aplicación.
- En algunos casos los clientes pueden modificar la estructura y comportamiento de la RIA pudiendo provocar que falle, se pude perder la integridad entre lo que es el HTML y la RIA.

1.5 Tecnologías para Desarrollar RIA

Actualmente existen varios frameworks y herramientas para el desarrollo de aplicaciones tipo RIA, que están basados en varios lenguajes y diferentes plataformas, como ejemplo tenemos a Silverlight de Microsoft, Adobe Flex y Adobe AIR, AJAX, OpenLaszlo, Bindows y la familia de tecnologías JavaFX. A continuación mencionaremos estas tecnologías.

1.5.1 AJAX

AJAX es la abreviación de JavaScript asíncrono y XML, es un término que es aplicado a un conjunto de tecnologías que ayudan al desarrollo de aplicaciones RIA y que son ejecutadas del lado del cliente a través de un navegador web. Combina las siguientes tecnologías:

- HTML y CSS (hojas de estilo).
- DOM (Modelo de Objetos para la representación de documentos). Este es un estándar basado en Objetos que establece las reglas para la visualización y representación de documentos HTML y XML.
- XMLHttpRequest. Es un objeto usado para la comunicación asíncrona con el servidor Web, de tal manera que funciona como interfaz para realizar las peticiones hacia el servidor.
- XML. Este es el formato en que los datos son transmitidos al cliente.

Las aplicaciones AJAX pueden ejecutarse casi en cualquier navegador y básicamente dependen del soporte del navegador en donde se ejecuta. El uso AJAX permite una mayor interacción con el usuario, velocidad en ejecución y un menor tráfico de datos con el servidor.

1.5.2 Silverlight

Esta es la propuesta por parte de Microsoft, básicamente funciona como un complemento para navegadores web. Ofrece un entorno para desarrollar aplicaciones RIA parecidas a Flash y esta basado en la plataforma de Windows, por lo tanto solo esta disponible para algunos navegadores y para los sistemas operativos Windows y Mac OS aunque también existe Moonligh que es alternativa para sistemas Linux y de distribución libre.

Microsoft Expression Blend es la herramienta usada para programar RIA con Silverlight, el modo de programación es mediante el uso de JavaScript para el acceso a datos y objetos, e XAML (Lenguaje Extensible de Formato para Aplicaciones) que es un lenguaje declarativo basado en XML para la creación de interfaces gráficas.

Estas aplicaciones también pueden ser cargadas dinámicamente como en el caso de AJAX y pueden almacenar en el equipo donde se ejecutan otro tipo de recursos, otra ventaja es que su contenido es mejor indexado por buscadores a diferencia de lo que sucede con Flash.

1.5.3 Adobe Flex

Es un framework que agrupa un conjunto de tecnologías para desarrollar RIA basadas en la plataforma Adobe Flash. El SDK de Flex es de distribución libre y es de código abierto, también los desarrolladores pueden utilizar un IDE basado en eclipse llamado Flex Builder para desarrollar este tipo de aplicaciones.

Flex utiliza una biblioteca de clases escritas en ActionScript 3.0 y también el lenguaje MXML (Multimedia eXtensible Markup Language) un lenguaje basado en XML para definir los componentes de las interfaces para el usuario, aunque también MXML es utilizado para definir enlaces de datos entre componentes gráficos y acceso a la parte del servidor.

ActionScript 3.0 es un lenguaje orientado a objetos y usado para la animación de aplicaciones web, es ejecutado sobre la máquina virtual AVM2. Utiliza una API XML basada en la especificación de ECMAScript lo cual optimiza el desarrollo de aplicaciones que usan XML y reduce el código.

Las etiquetas que son definidas en MXML corresponden a ciertas clases y propiedades de ActionScript y pueden ser compiladas en archivos SWF que es el formato de archivos gráficos vectoriales usados por Flash, y de esta manera poder ser desplegadas como cualquier archivo de Flash.

1.5.4 Adobe AIR

AIR (Adobe Integrated Runtime) es otra propuesta por parte de Adobe para el desarrollo de RIA; consiste básicamente en un entorno de ejecución que puede combinar varias tecnologías como HTML, JavaScript, Flash y Flex, además de reutilizar el código de estas. Su objetivo principal es desarrollar aplicaciones enriquecidas para que sean ejecutadas como aplicaciones de escritorio.

Estas aplicaciones no requieren de ser ejecutadas en un navegador web, esta es una ventaja ya que escapa a las limitantes del propio navegador y puede disponer de los recursos locales del sistema además no requiere de estar conectada a Internet para funcionar; pero por otra parte tiene la desventaja que debe ser empaquetada y debe firmarse digitalmente para que pueda ser instalada en los equipos donde se vaya a ejecutar.

1.5.5 OpenLaszlo

Es una plataforma de código abierto para el desarrollo de RIA, este tipo de aplicaciones pueden correr de dos diferentes formas: SOLO (Standalone OpenLaszlo Output), y mediante el servidor de OpenLaszlo.

En el modo SOLO las aplicaciones pueden correr desde cualquier servidor Web y son programadas en JavaScript y con el lenguaje LZX que es un muy similar al MXML de Adobe y XAML de Microsoft. Estas aplicaciones pueden desplegarse como HTML dinámico o también pueden ser compiladas a un archivo binario SWF que puede cargarse en una página de forma estática.

Cuando las aplicaciones corren en el Servidor OpenLaszlo, estas son compiladas y regresadas al navegador de forma dinámica como un Servlet de Java, esto da soporte a aplicaciones que requieren integración y persistencia de datos al momento de la ejecución de la aplicación.

La ventaja de estas aplicaciones es que se pueden desplegar en los navegadores web que tengan el reproductor de Flash, además son aplicaciones robustas y escalables ya que pueden implementarse con la plataforma Java EE.

1.5.6 Bindows

Es un framework para desarrollar RIA y aplicaciones similares a las de AJAX pero con la apariencia de Windows. Este framework se basa en el desarrollo de HTML dinámico mediante el uso de JavaScript como lenguaje de programación y el uso de unas plantillas denominadas ADF (Application Description Files) basadas en XML y que sirven para la construcción de las interfaces graficas del usuario.

Este tipo de aplicaciones no requieren de la instalación de ningún control, complemento o ActiveX para funcionar, y solo se ejecutan del lado del cliente, así que solo es necesario descargar la aplicación del servidor.

El SDK de Bindows es de distribución libre; sin embargo, para el desarrollo de aplicaciones comerciales hay que pagar una licencia.

1.5.7 JavaFX

JavaFX es un conjunto de varias tecnologías de Sun Microsystems para el desarrollo de RIA, a diferencia de otras tecnologías se puede implementar en diferentes dispositivos que contengan el ambiente de ejecución de java, por lo tanto esta orientado hacia las interfaces altamente animadas. Actualmente este tipo de aplicaciones pueden ser desplegadas en dispositivos móviles, navegadores web y aplicaciones de escritorio.

JavaFX incorpora dos tecnologías: JavaFX Script y JavaFX Mobile. JavaFX Script es un lenguaje declarativo, orientado a objetos y que es compilado en códigos de bytes entendibles para la Máquina Virtual de Java para su ejecución. JavaFX Mobile es la

plataforma para desarrollar aplicaciones para dispositivos móviles o portátiles, estas aplicaciones son integradas en Java ME y son capaces de ejecutarse sobre varios Sistemas Operativos.

JavaFX Script es el lenguaje usado tanto para desarrollar aplicaciones móviles, de escritorio y web, también puede importar código de Java aumentado así la robustez de este tipo de aplicaciones. Toma ventaja de la portabilidad y seguridad de Java por lo tanto es multiplataforma.

El SDK de JavaFX es de distribución libre y después será de código abierto, también se han planeado lanzamientos para otro tipo de plataformas como servicios de televisión, discos Blu-Ray y consolas de videojuegos.

CAPÍTULO 2

LA TECNOLOGÍA JAVAFX

2.1 Introducción a JavaFX

La tecnología JavaFX nació como un proyecto llamado F3 (Form Follows Function) en una compañía de software llamada *SeeBeyond*, el desarrollo de este lenguaje fue emprendido y encabezado por Chris Olivier quién se desempeñaba como ingeniero en software. Tiempo después Sun Microsystems absorbió a la compañía. Al poco tiempo en mayo del 2007 y gracias al interés de Sun en el lenguaje F3, se realizó la conferencia anual de Sun llamada JavaOne [1]en donde se anunció a JavaFX como la nueva tecnología por parte de Sun para el desarrollo de las RIA, es así como en diciembre del 2008 se libera la primera versión del SDK de JavaFX, esta primera versión incluye dos tecnologías: JavaFX Script y JavaFX Mobile.

[1] **JavaOne** es una conferencia anual (desde 1996) puesta por Sun Microsystems para discutir las tecnologías de Java, sobre todo entre los desarrolladores de Java. JavaOne se sostiene en el centro de Moscone en San Francisco, California generalmente de abril a junio y funciona típicamente de domingo a viernes.

El proyecto desde entonces ha sido apoyado fuertemente por Sun y por toda la comunidad de Java incluyendo a James Gosling el padre de Java. Con el advenimiento de las Aplicaciones de Internet Enriquecidas (RIA) se han puesto grandes expectativas en JavaFX para competir contra otras tecnologías ya existentes en el mercado desde hace tiempo como es el caso de Silverlight por parte de Microsoft y Flex de Adobe principalmente.

La idea principal de Chris Olivier era desarrollar un lenguaje basado en la plataforma de Java y que el código desarrollado por el nuevo lenguaje pudiera ser interpretado y ejecutado sobre la Máquina Virtual de Java, y de esta manera poder explotar todos los beneficios de portabilidad de las aplicaciones y tecnologías Java. Esto le brinda a las aplicaciones desarrolladas en JavaFX una gran ventaja al poder estar presentes en una gran variedad de dispositivos que tengan instalada la Máquina Virtual de Java, desde dispositivos móviles, computadoras de escritorio hasta Servidores.

2.2 Descripción y Arquitectura

La tecnología JavaFX puede ser aplicada en muchos campos y plataformas. En el desarrollo de RIA existen ya varias tecnologías sin embargo JavaFX tiene un gran potencial a futuro ya que no solo esta enfocado a aplicaciones web o de escritorio, también tiene soporte para aplicaciones de dispositivos móviles y tiene contemplado dar soporte a otras plataformas en el futuro.

Otra gran ventaja es la capacidad de poder integrar código de JavaFX Script y código Java, lo cual potencializa el alcance de las RIA, por ejemplo puede integrarse acceso a Base de Datos, servicios web y en general operaciones que requieren mayor grado de complejidad.

2.2.1 Características de JavaFX

La plataforma JavaFX esta integrada por dos tecnologías: JavaFX Script y JavaFX Mobile. JavaFX Script es el nuevo lenguaje para el desarrollo de interfaces gráficas y que puede ser ejecutado en cualquier dispositivo que cuente con la Máquina Virtual de Java; en cambio JavaFX Mobile es un completo sistema de software que ya esta disponible para los fabricantes de dispositivos móviles y que servirá como punto de partida para que esta empresas puedan desarrollar sus propias aplicaciones y que a la vez estos dispositivos tengan soporte para aplicaciones desarrolladas en Java ME y JavaFX.

JavaFX provee muchas funcionalidades y una gran integración con todas las tecnologías Java existentes, así que en un futuro podremos ver desde pequeñas aplicaciones para celulares hasta robustas aplicaciones empresariales con una interfaz gráfica en JavaFX.

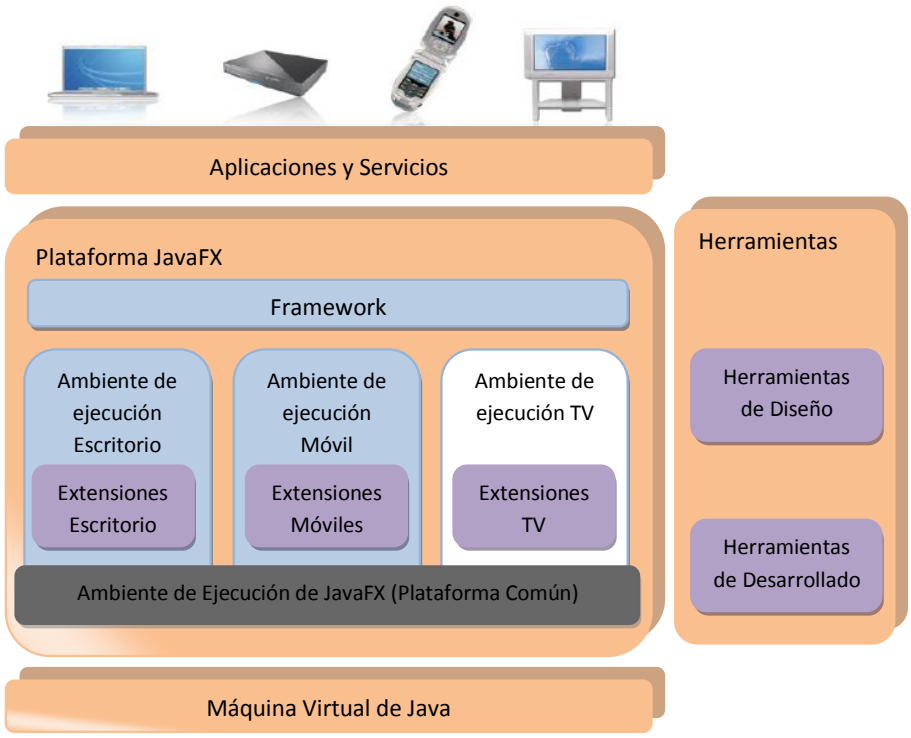
El primer SDK de JavaFX liberado esta conformado por los siguientes componentes:

- El ambiente de ejecución de JavaFX.
- La API de JavaFX
- El compilador del lenguaje JavaFX Script, llamado *javafxc*.
- Documentación de la API de JavaFX.
- Un emulador para aplicaciones móviles, por el momento solo tiene soporte para Windows.
- Ejemplos de aplicaciones con JavaFX Script.

Sun Mycrosystems tiene proyectado en un futuro el lanzamiento de más productos y soporte a más plataformas y/o Sistemas Operativos. Por lo mientras continuaremos más adelante con JavaFX Mobile y JavaFX Script.

2.2.2 Arquitectura de JavaFX

Las aplicaciones desarrolladas con JavaFX Script pueden ser ejecutadas a través de diferentes arquitecturas y/o Sistemas Operativos gracias a la Máquina Virtual de Java. En el siguiente diagrama podemos observar esta arquitectura.



Las aplicaciones pueden ejecutarse sobre una diferente y amplia variedad de dispositivos como puede ser el caso de una aplicación que se ejecuta tanto en un navegador web o como un programa en el escritorio como en un celular o cualquier dispositivo portátil.

El código escrito con JavaFX Script puede combinarse con código Java, JSP o HTML; el compilador de JavaFX (*javafxc*) convierte el código fuente (archivos con la extensión *.fx*) en *Bytecodes* que son interpretados por la Máquina Virtual de Java, estos *Bytecodes* también pueden contener código Java ya que JavaFX Script puede importar cualquier clase de Java para su uso.

La plataforma de JavaFX comparte muchas funcionalidades en su API que pueden utilizarse dentro de los diferentes dispositivos donde se ejecutan estas aplicaciones, sin embargo, a veces se puede requerir de ciertas API´s especificas para la ejecución de alguna aplicación en un determinado ambiente de ejecución que por el momento son para aplicaciones de escritorio y web, móviles, y más adelante también las habrá para servicios de televisión y soporte a discos Blu-Ray.

Una característica de JavaFX es que tanto desarrolladores como diseñadores pueden trabajar en conjunto en el desarrollo de RIA's. Ya están disponibles varios plugins tanto para IDE's como herramientas de diseño por citar algunos ejemplos tenemos: Eclipse, Netbeans, y Adobe Photoshop, etc. Que dan soporte a JavaFX y de los cuales hablaremos adelante.

2.3 JavaFX Mobile

JavaFX Mobile es considerado como un completo sistema de software para dispositivos móviles, el corazón de este Sistema esta constituido por un núcleo o kernel de Linux y otras tecnologías de Java que corren sobre este pequeño Sistema Operativo. Esta tecnología esta enfocada a que los fabricantes de teléfonos y dispositivos portátiles desarrollen sus aplicaciones e interfaces graficas sobre la plataforma Java.

Al día de hoy los fabricantes de celulares se encuentran trabajando con JavaFX Mobile. Esta tecnología esta disponible para fabricantes y proveedores de telecomunicaciones mediante licencias para Fabricantes de Equipos Originales (OEM).

Como se puede ver en el diagrama las funciones tanto las funciones básicas del teléfono como las aplicaciones estarán potenciadas por Java y JavaFx. Así que podemos observar tres capas.

Una capa de Frameworks y API's potenciadas con Java para la administración de aplicaciones, librerías propias del sistema, un motor de gráficos; Frameworks para seguridad, telefonía, multimedia y por supuesto la Máquina Virtual de Java que es donde correrían las aplicaciones Java.

La siguiente capa estaría conformada por las aplicaciones tal como son las de mensajería, para navegación en internet, reproductor de medios y otras aplicaciones como pueden ser juegos.

Y por último JavaFX vendría a enriquecer todo el aspecto gráfico para las aplicaciones Java y las que también son propias del teléfono.

Esta iniciativa por parte de Sun Microsystems viene como opción para solucionar la problemática actual que tienen los dispositivos móviles y celulares de compatibilidad, ya que Java garantiza que si una aplicación se ejecuta en un aparato, también lo hará sin problemas en cualquier otro. Teniendo en cuenta que ya existen millones de teléfonos con soporte a Java es un buen punto de partida para impulsar el desarrollo con JavaFX.

Otra ventaja que significa implementar el sistema JavaFX Mobile es la posibilidad de tener aplicaciones capaces de ejecutar tanto en dispositivos móviles y en diferentes entornos como el web, aplicaciones de escritorio y otros tipos de dispositivos como computadoras y televisión.

2.4 JavaFX Script

JavaFX Script es el lenguaje usado para desarrollar RIA, este lenguaje es completamente nuevo que tiene como antecesor a F3, y como características principales podemos mencionar las siguientes:

- Es un lenguaje declarativo y orientado a objetos, JavaFX Script difiere de otras tecnologías por ser un lenguaje puramente declarativo, por ejemplo Flex y Silverlight utilizan un lenguaje de Script como ASP o JavaScript, y un modelo de plantillas mediante lenguajes basados en XML para crear objetos gráficos.
- JavaFX Script permite utilizar imágenes o si lo preferimos también las podemos dibujar objetos desde cero con este lenguaje.
- Su potencial es muy grande al poder interactuar directamente con el lenguaje Java y las API's que ya existen como SWING, Java2D y Java 3D, dándoles una mejor vista.
- Permite un desarrollo más rápido tanto para desarrolladores de software como diseñadores.
- Es un lenguaje que es compilado a *bytecodes* para su ejecución en la Máquina Virtual de Java.
- Permite el enlace directo de los datos que se están utilizando con los componentes gráficos, tiene un manejo para los eventos generados por el usuario y disparadores.
- Provee funciones y efectos para el manejo de gráficos y video.
- Da soporte para el acceso de información en internet como son los servicios web y fuentes web como RSS y Atom, por ejemplos noticias al ultimo momento, estado del tiempo, mapas y otros servicios que ofrecen otras empresas como Google y Yahoo.
- Es posible acceder a los sistemas de archivos del Sistema, así como también impresoras o escáneres, desde aplicaciones en JavaFX.

2.5 Herramientas para el Desarrollo de JavaFX

Aunque JavaFX Script es un lenguaje relativamente nuevo y ya existen varias herramientas para desarrollar aplicaciones e interfaces gráficas. Estas herramientas están enfocadas a un determinado grupo desarrolladores o diseñadores.

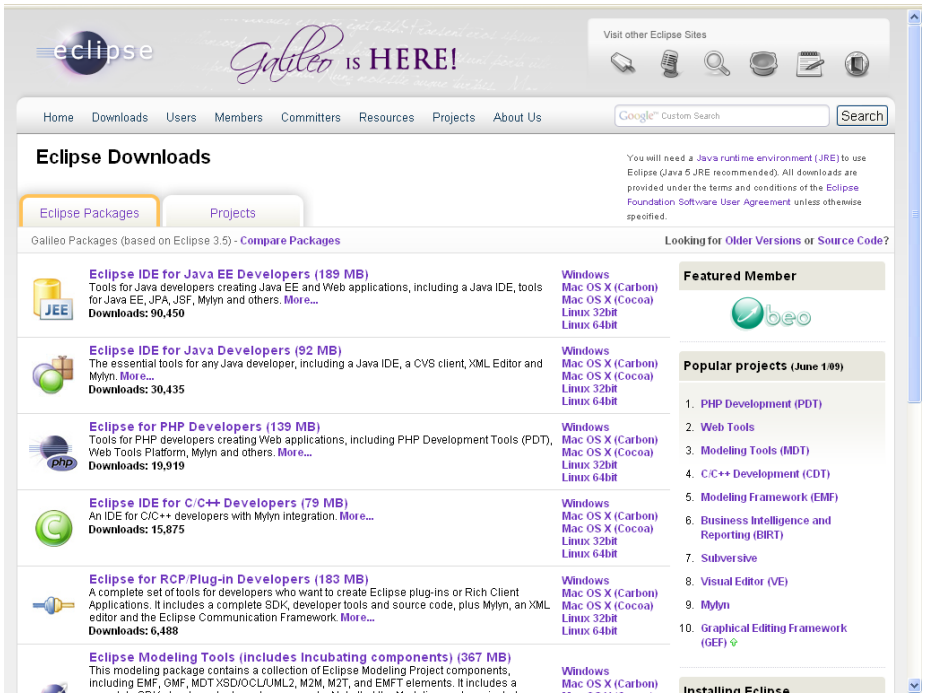
Los desarrolladores cuentan con herramientas de trabajo denominadas *Entorno de Desarrollo Integrado* (IDE), estas herramientas proveen un ambiente amigable para poder desarrollar, probar y depurar código, generalmente dan soporte a varios lenguajes de programación. Recientemente algunos de estos IDE pueden integrar de manera opcional plugins para el desarrollo de aplicaciones con JavaFX.

Enseguida mencionaremos los IDE utilizados para el desarrollo en JavaFX y más adelante se mencionarán las herramientas utilizadas para el diseño con JavaFX.

2.5.1 Eclipse

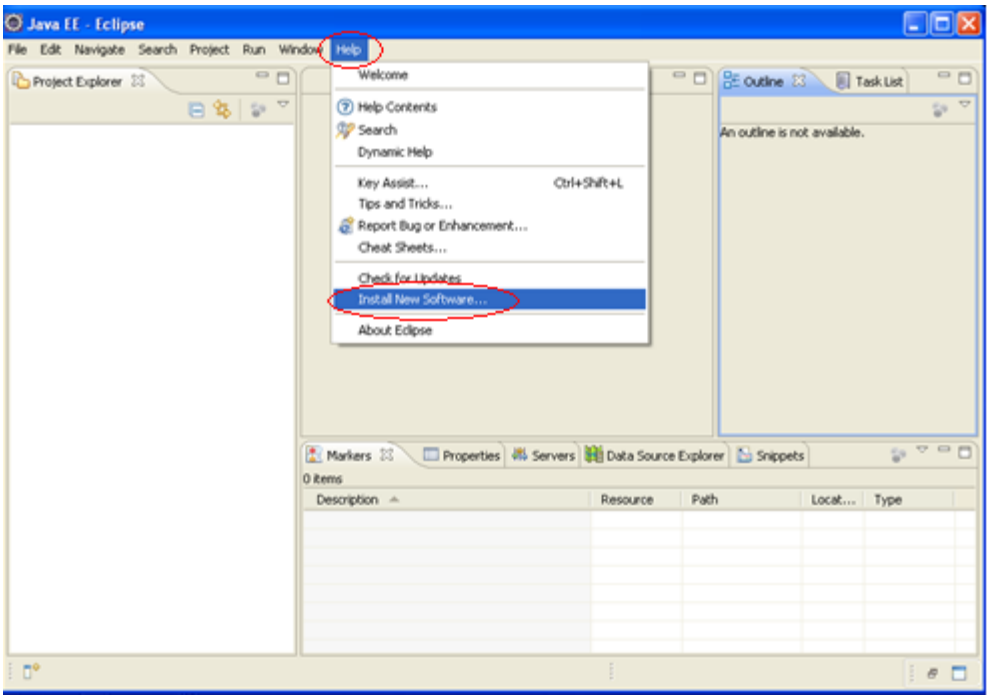
Este es el otro famoso IDE para el desarrollo en Java el cual también ya da soporte a JavaFX mediante un plugin. Al igual que Netbeans existen diferentes versiones del IDE de Eclipse que podemos descargar, ya que nos permite desarrollar en diferentes lenguajes y diversas tecnologías de Java. Podemos descargar una versión de de Java Developers o JavaEE.

http://www.eclipse.org/downloads/



Este IDE no requiere de una instalación, solo es necesario descomprimir el archivo que descargamos y asignar una ruta para almacenar los proyecto “Workspace” y empezar a trabajar.

Para agregar el plugin de JavaFX y una vez abierto el IDE, se va a la barra de menú y se escoge la opción *Help* y después *Install New Software*.

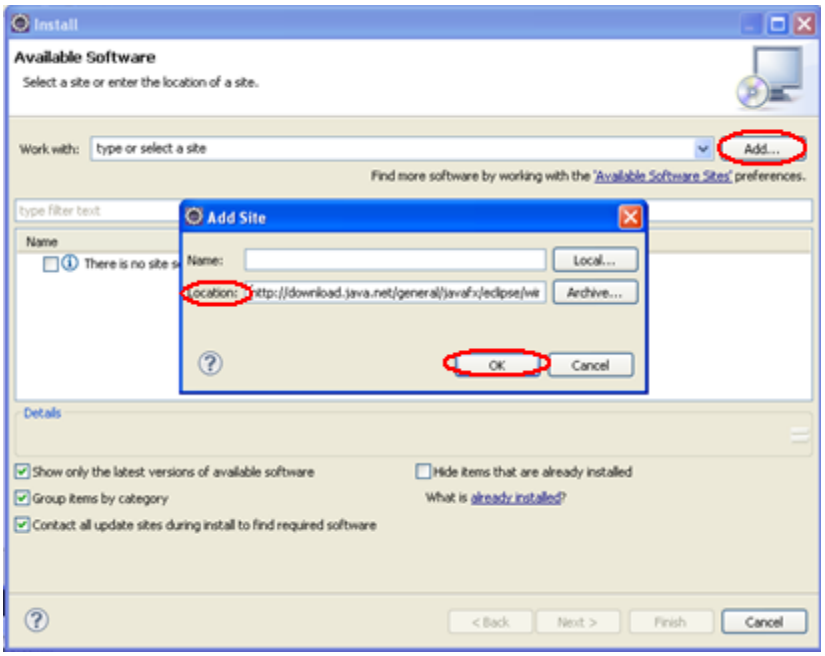


Nos aparecerá una ventana para el Software disponible y le damos en la opción *Add*, donde aparece un recuadro solicitando una ruta, una vez ahí introducimos la siguiente ruta en *Location* y pulsamos *OK*.

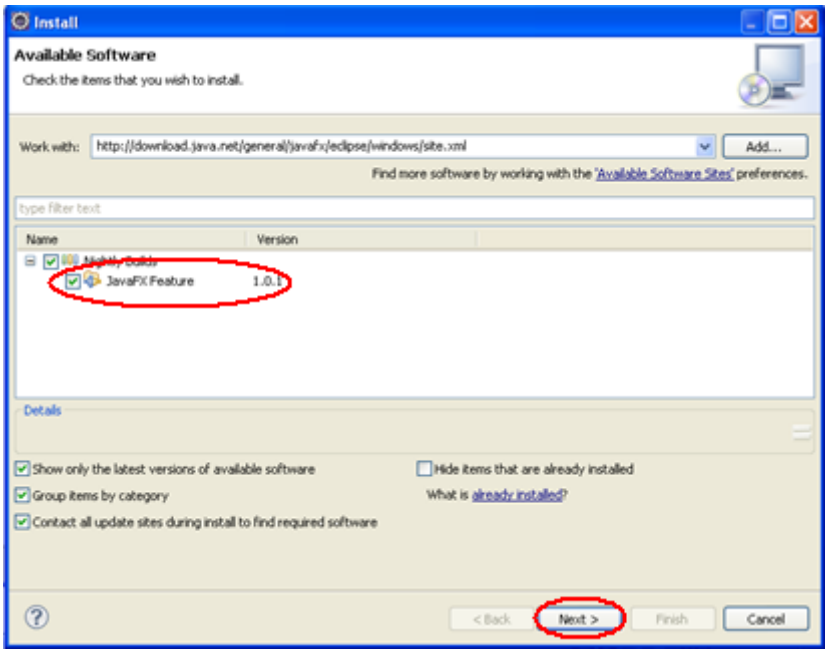
```
http://download.java.net/general/javafx/eclipse/{Sistema Operativo}/site.xml
```

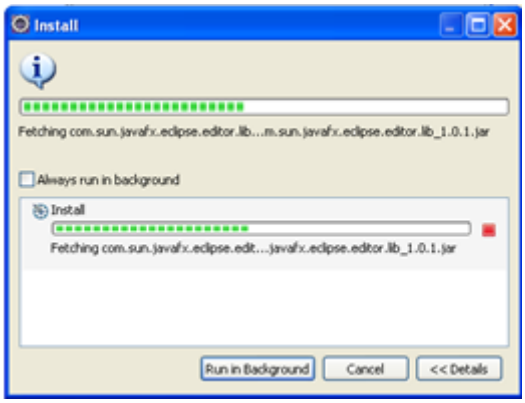
Sistema Operativo: windows | linux | macos

Dependiendo del Sistema Operativo en donde estemos trabajando, especificamos en la ruta Windows, Linux o Mac OS.



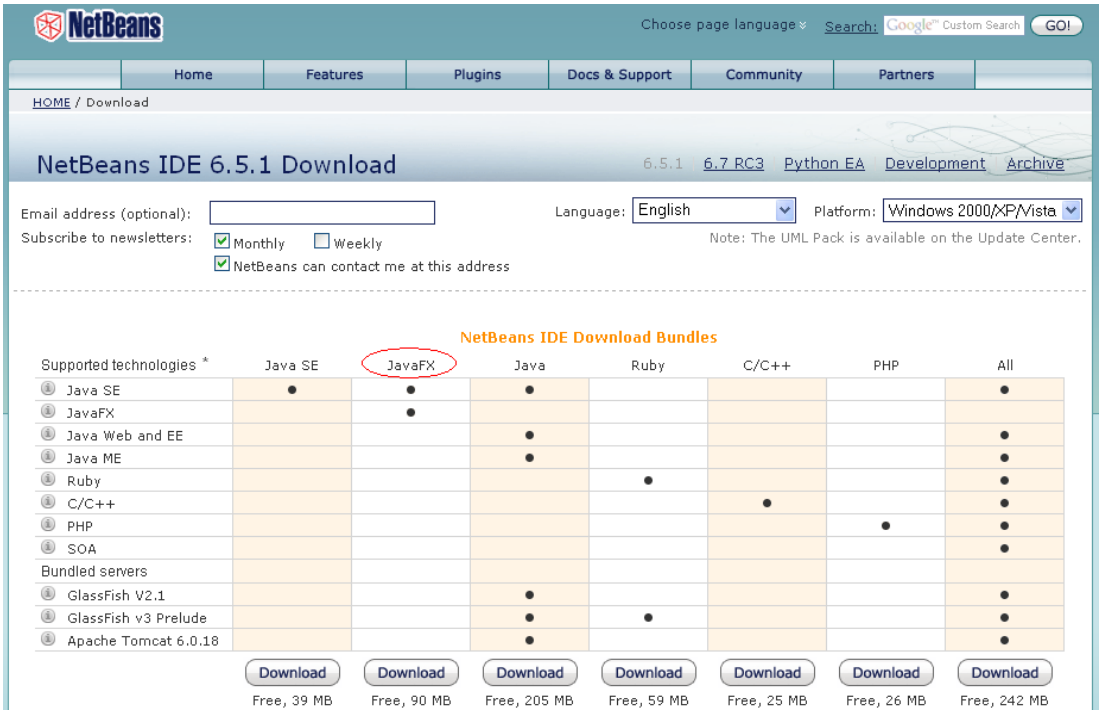
Ya nos aparecerá el plugin listo para descargar, y solo tenemos que seguir las demás ventanas con "siguiente", para completar la instalación de JavaFX.





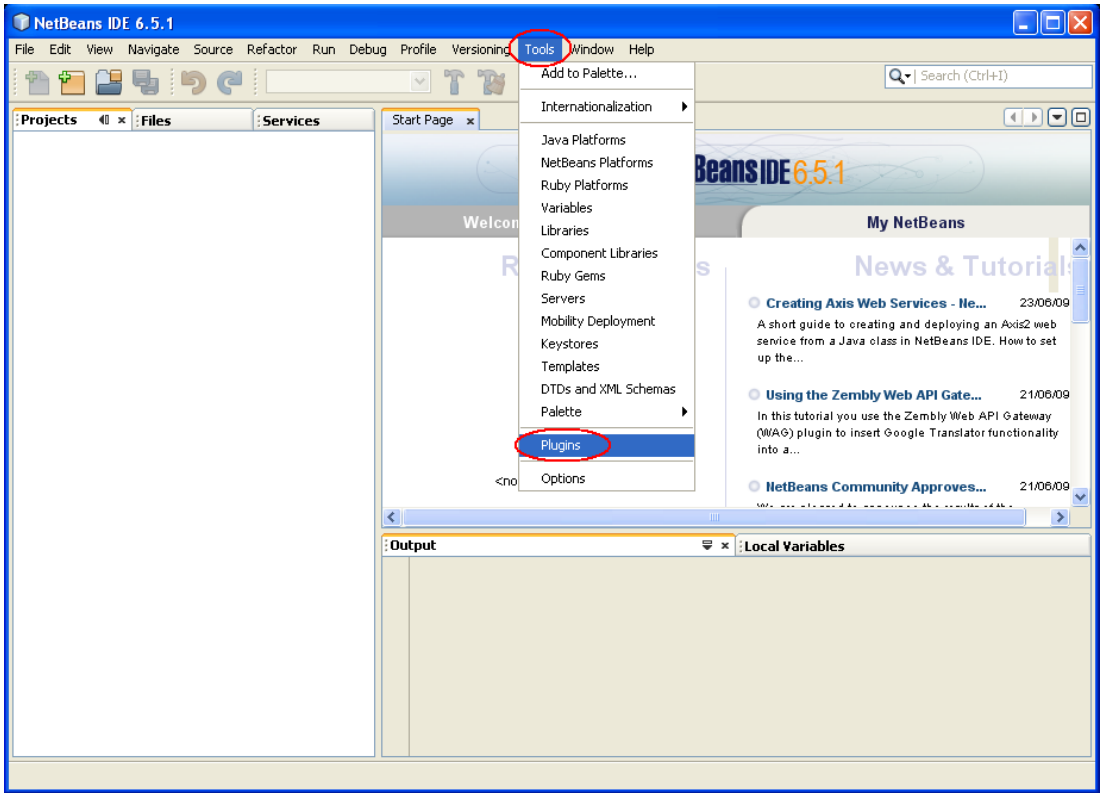
2.5.2 Netbeans

Este es el IDE de desarrollo en Java por parte de Sun Microsystems, y fue el primero en incorporar JavaFX, inclusive salió una versión especial para JavaFX y Java, que podemos descargar en esta ruta: http://www.netbeans.org/downloads/index.html.

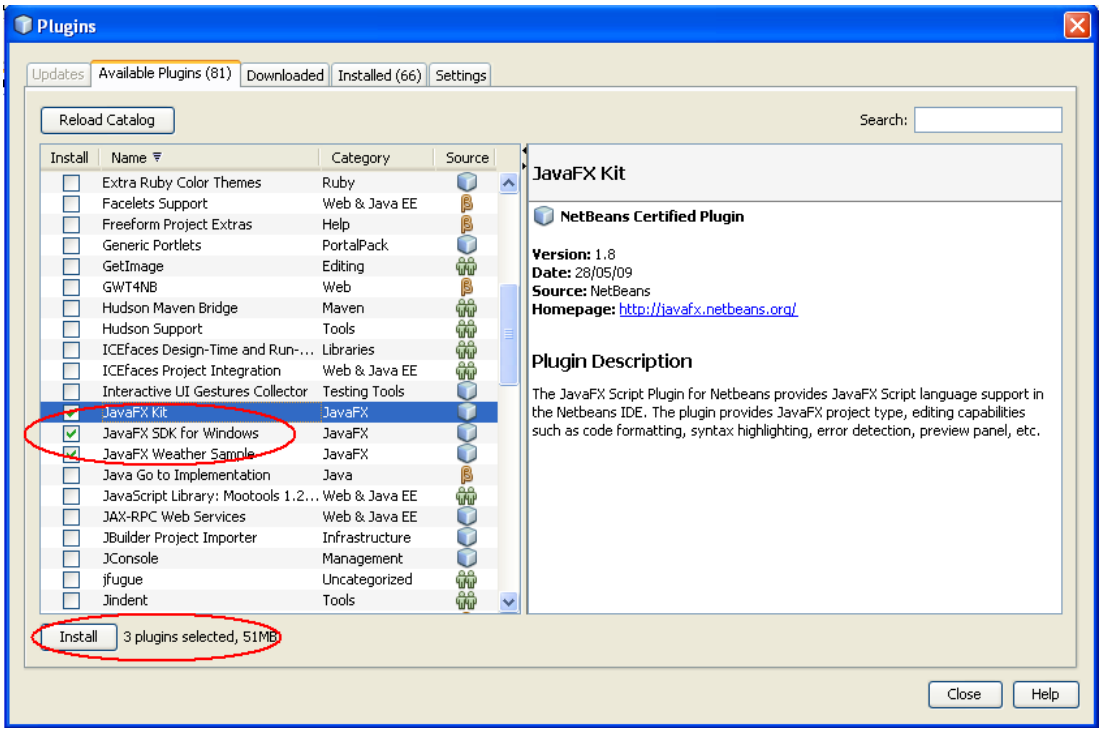


Las demás versiones de Netbeans a partir de la 6.5 pueden descargar los plugins para incorporar JavaFX. Estos plugins incluyen la API y su documentación así como algunos proyectos de ejemplo.

Para descargar los plugins de JavaFX tenemos que acceder al menú "Tools" y luego dar en la opción Plugins.



Posteriormente en la pestaña *Available Plugins* buscamos y seleccionamos los plugins de JavaFX y damos clic en instalar.



Una vez instalados los plugin podemos crear proyectos en JavaFX, el IDE nos sirve mucho a la hora de codificar ya que podemos ver la API y las funciones, así como también la correcta sintaxis del lenguaje. Netbeans también cuenta con un pre visualizador de lo que genera nuestro código.

Netbeans es un IDE muy amigable y ofrece diferentes maneras para desplegar aplicaciones en JavaFX como es el caso del emulador para aplicaciones móviles, además cuenta con un pre visualizador y otras opciones. En general es el IDE más completo para el desarrollo con JavaFX Script, por tales motivos se ha optado por Netbeans 6.5.1 para el desarrollo de ejemplos y aplicaciones de capítulos posteriores.

2.6 Herramientas para el diseño con JavaFX

JavaFX no sólo está enfocado al desarrollador, el desarrollo de RIA también implica un trabajo de diseño, para lo cual se han desarrollado plugins para herramientas de diseño. Para los diseñadores existe *JavaFX Production Suite* que es un conjunto de herramientas y

plugins para Adobe Photoshop CS3 y Adobe Illustrator CS3, y que nos permiten manipular imágenes y gráficos para después poder exportarlos a archivos .fx. Con tales ventajas es más fácil mejorar el aspecto gráfico de las aplicaciones que pueden hacer solo los desarrolladores mediante código.

2.6.1 JavaFX Production Suite

JavaFX Production Suite una la tecnología enfocada a diseñadores ya que consiste en una serie de herramientas y plugins para Adobe Photoshop CS3 e Illustrator CS3. Esto permite que los diseñadores trabajen de forma normal con sus herramientas para renderizar sus imágenes, la ventaja de esto es que una vez instalados los plugins se pueden exportar a código JavaFX Script las imágenes o gráficos que deseemos.

Esto permite un desarrollo e integración de código más rápido, explotando estas herramientas de diseño se puede lograr realizar tratamiento de imágenes que difícilmente se conseguirían solo escribiendo código.

2.6.2 JFXBuilder

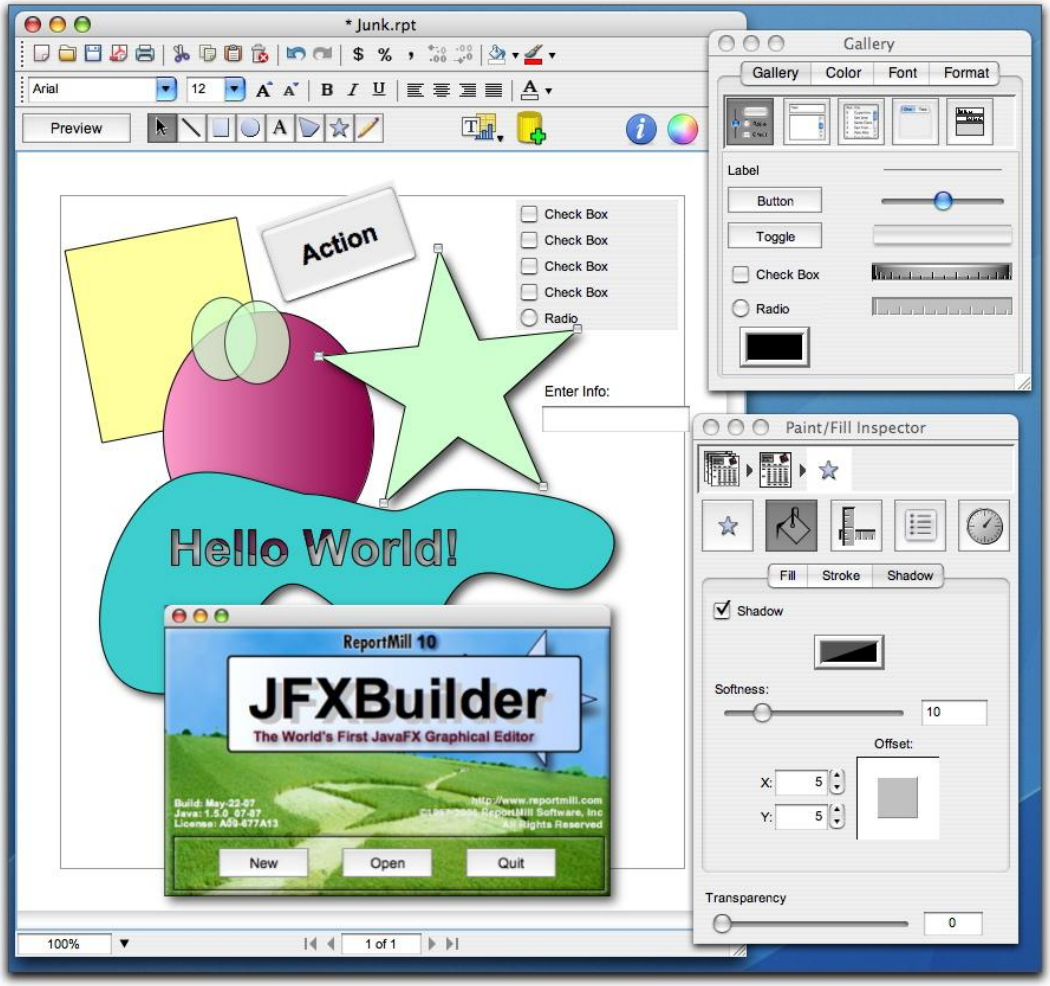
Es una herramienta grafica para el diseño de figuras con JavaFX y que nos genera el código de JavaFX Script en automático de las figuras y efecto que creamos. Esta herramienta ha sido desarrollada por ReportMill y podemos descargar el fichero jnlp desde http://www.reportmill.com/jfx/, de esta manera podemos tener disponible la ultima versión del programa.

Algunas de las cosas que podemos hacer con esta herramienta son:

- Añadir y editar textos, usando varios estilos, fuentes, colores, etc.
- Aplicar diferentes efectos de relleno, como texturas y colores degradados.
- Aplicar diferentes efectos, como sombras, reflexión, efectos de luz, relieve, etc.

- Aplicar transformaciones a las figuras como rotación y escalas.
- Arrastrar y manipular imagines y gráficos.
- Aplicar diferentes comportamientos como eventos provenientes del mouse.
- Enlazar datos directamente con una Base de Datos o un archivo XML.
- Diseñar plantillas para dispositivos que utilicen JavaFX Mobile.

Con las características que nos ofrece esta herramienta podemos generar código más rápido con solo arrastrar y manipular los diferentes objetos.



2.7 Modos de Despliegue de aplicaciones en JavaFX

Actualmente podemos ejecutar una aplicación escrita con JavaFX Script de cuatro modos distintos, solamente el IDE de Netbeans soporta estos modos de ejecución, por lo cual se convierte en el mejor entorno para desarrollar con Java y JavaFX, y el único que incorpora un emulador para aplicaciones móviles.

Actualmente en Netbeans tenemos cuatro modos de despliegue para aplicaciones JavaFX. Para configurar el tipo de ejecución para una aplicación tenemos que dar clic derecho sobre algún proyecto en JavaFX y después en la opción *properties*, nos aparecerá una ventana y una vez ahí escógenos la opción *Run*. Es ahí donde escogemos el tipo de ejecución de nuestra aplicación. Los modos de ejecución son los siguientes:

- Ejecución Estándar. Esta es la ejecución por defecto y se ejecuta como una aplicación de escritorio.

- Ejecución Web Star. También actúa como una aplicación de escritorio pero funciona con una tecnología llamada *Java Web Start Technology (JavaWS)*. JavaWS es una tecnología incluida en el ambiente de ejecución de Java (JRE) y cuya funcionalidad es poder descargar aplicaciones Java como archivos con la extensión *.jnlp* desde un servidor. La ventaja de esta herramienta es de que descarga en automático la última versión de la aplicación en cuestión y así poder ejecutarla de modo local la última versión.

- Ejecución dentro de un navegador web. La ejecución se lleva a cabo del mismo modo que un applet de Java incrustado dentro de un navegador web. Por ejemplo el siguiente código entre las etiquetas: <*script*>, invoca el JAR que contiene la aplicación.

```
<script>
    javafx(
        {
            archive: "AplicacionFX.jar",
            width: 320,
            height: 240,
            code: " AplicacionFX.Main",
            name: " AplicacionFX "
        }
    );
</script>
```

- Ejecución en un emulador de dispositivos móviles. Este es el emulador que provee Netbeans, y nos da una visión de las aplicaciones con JavaFX Script que podemos programar para dispositivos móviles. Cabe mencionar que por el momento solo esta disponible para el Sistema Operativo Windows.

Los archivos JNLP (Java Networking Launching Protocol) es la especificación que utiliza JavaWS y básicamente su estructura es como la de un archivo XML. Este archivo nos permite conectarnos directamente a un servidor web, de esta forma podemos descargar la última versión de la aplicación disponible. Una vez que se descarga la aplicación se necesita la autorización del usuario para poder ejecutarse.

CAPÍTULO 3
SINTAXIS DE JAVAFX SCRIPT

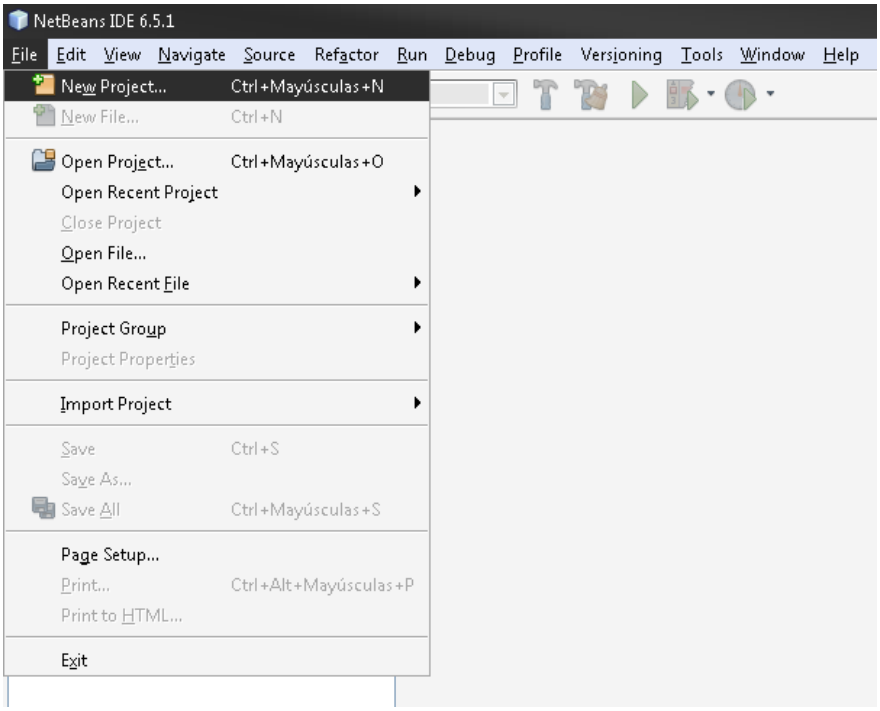
3.1 Introducción

JavaFX Script es un lenguaje orientado a objetos, los objetos pueden organizarse y agruparse, organizarse por jerarquías y nodos, pudiendo estar unos anidados dentro de otros, tal como si fueran una estructura de árbol.

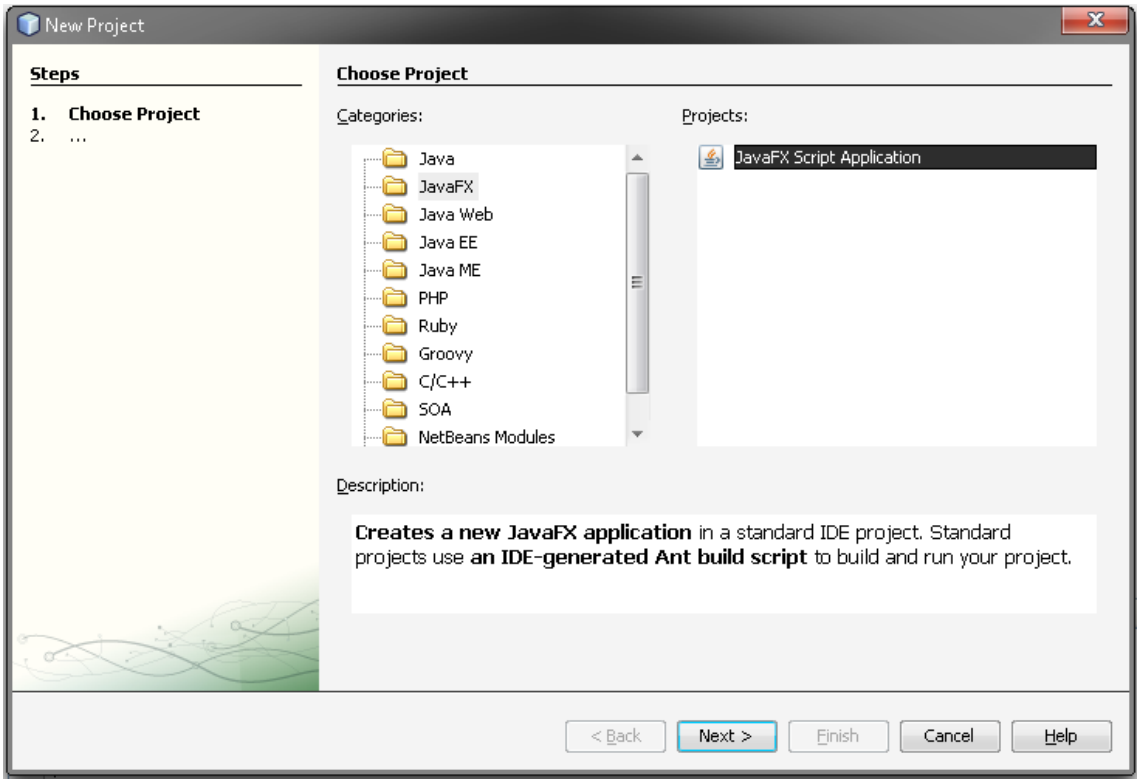
Es ente capitulo se revisaran las características fundamentales de la sintaxis JavaFX Script, también se verán algunos ejemplos donde se aplican estos fundamentos, empezando desde la declaración de tipos de datos y funciones hasta ver como enlazar variables, objetos, funciones y *triggers*.

Como primer ejemplo mostraremos un pequeño programa con un mensaje de saludo utilizando el IDE que elegimos anteriormente: Netbeans 6.5.1

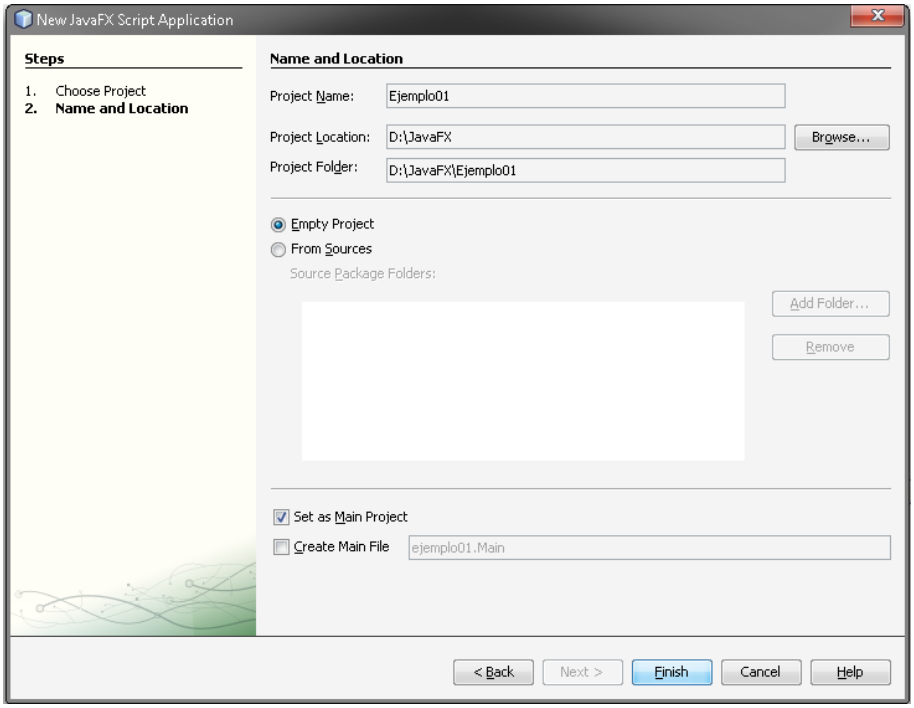
1. Primero creamos un nuevo proyecto con la opción *New Project.*



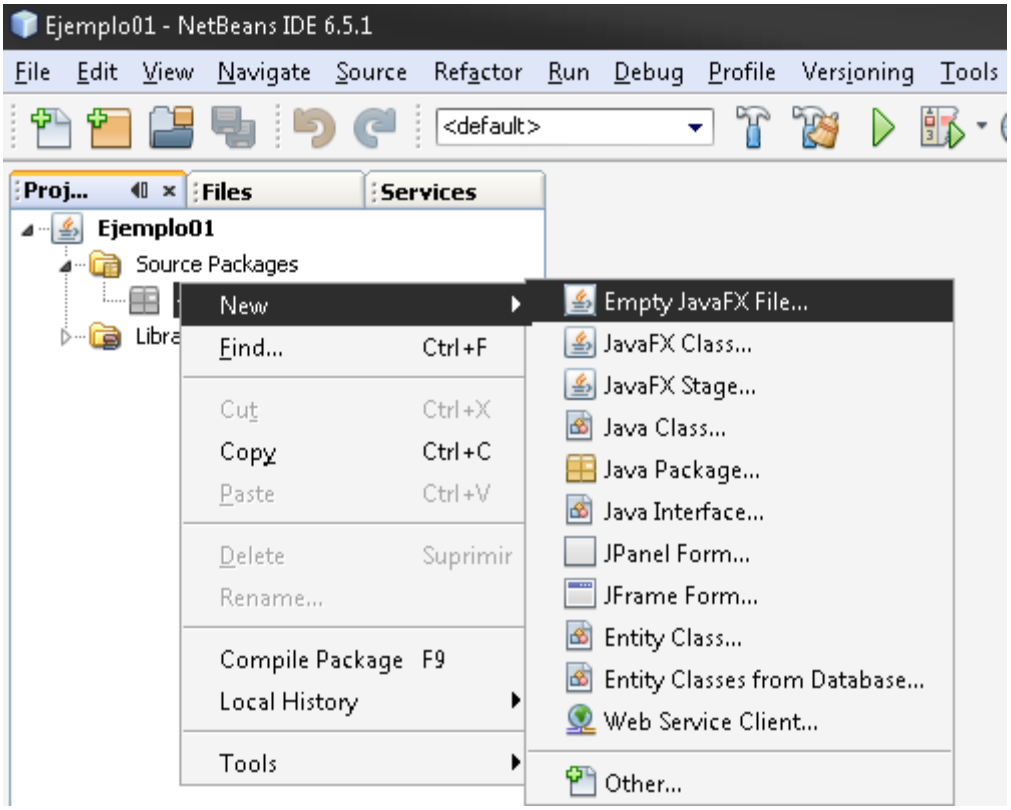
2. Dentro de la categoría JAVAFX escogemos una nueva aplicación del tipo JavaFX Script



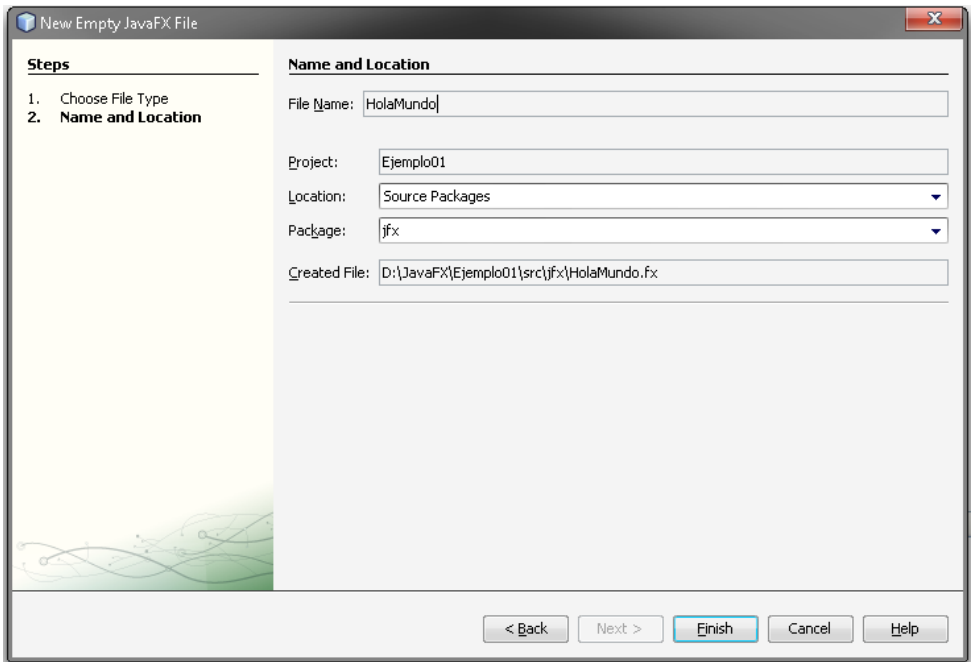
3. Nombramos el Proyecto y escogemos la ruta donde guardar el proyecto



4. Esto genera el nuevo proyecto, y a continuación añadimos un nuevo archivo vacío de tipo JavaFX



5. Indicamos el nombre del archivo y el paquete donde quedará almacenado el archivo.



6. Esto nos genera un archivo vacío, y lo sustituimos con el siguiente código:

```
package jfx;                          //paquete

import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Text;
import javafx.scene.text.Font;

Stage {                               //Declaración del objeto Stage
    title: "JavaFX Ejemplo 1 "        //titulo de la ventana
    width: 300                         //ancho
    height: 100                       //alto
    scene: Scene {                    //variable de instancia Scene
        content: [                    //contenido para Scene
            Text {                    //instancia de un objeto tipo
Text
                font : Font {         //variable de instancia Font
                    size : 16          //tamaño de la fuente
                }
                x: 10                  //posición de x para el
mensaje
                y: 30                  //posición de y para el
mensaje
                content:"Empezando a Programar con JavaFX"
            }
        ]
    }
}
```

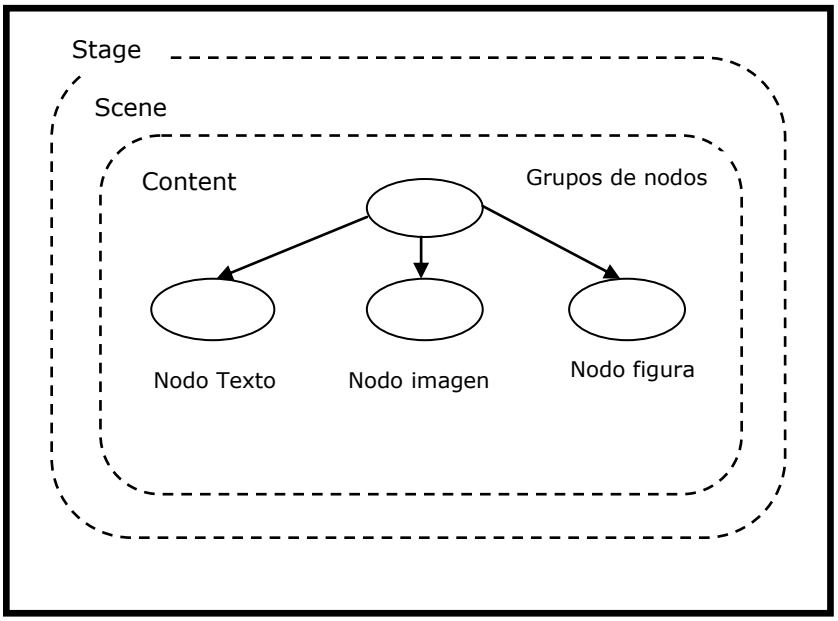
Lo primero que podemos observar son las cabeceras del programa. En la primer línea el paquete indica donde esta encuentra nuestro archivo; debajo se encuentran las declaraciones para importar las clases que se utilizarán, usamos la palabra *import*, importamos los objetos: *Stage, Scene, Text y Font.*

Stage es el objeto principal y funciona como un contenedor para poder desplegar objetos dentro de una ventana, aquí se definen propiedades de la ventana como el Título, y los tamaños de Ancho y Largo.

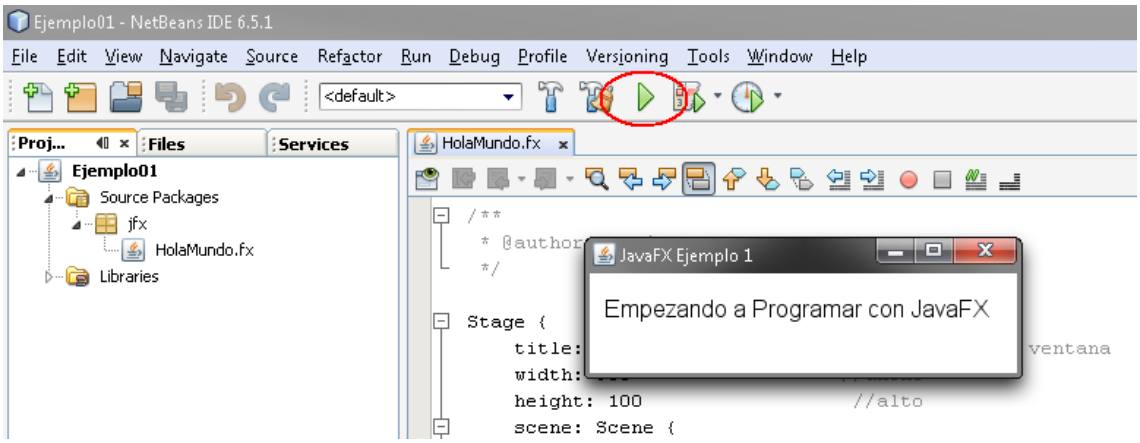
El objeto *Scene* es la instancia que actúa como una superficie sobre la cual podemos colocar demás objetos, o elementos gráficos. *Scene* tiene sus propios atributos, y en *content* es donde se sitúan los objetos gráficos que queremos que se muestren.

Para poder mostrar un mensaje recurrimos al objeto *Text*, y en este indicamos el contenido del texto dentro de *content*, declaramos una variable de instancia *Font* para indicar el tamaño de la fuente, y las propiedades de *x* así como *y* para posicionar el texto.

Como podemos observar la estructura del código en JavaFX es similar a la de un árbol donde la raíz es el elemento principal *Stage*, y este a su vez contiene a *Scene* donde podemos definir los demás elementos gráficos, que son denominados como nodos.



7. Para compilar y poder ejecutar el programa oprimimos el botón *Run* señalado en un rojo en la siguiente imagen, esto ejecuta el pequeño programa mostrando en una ventana el saludo que escribimos.



Este es el primer ejemplo un mensaje de saludo, en el próximo capítulo se continuará y se hablará de cómo construir interfaces gráficas más elaboradas, por lo mientras en este capítulo hablaremos de los fundamentos del lenguaje.

3.2 Uso de Variables y Funciones

En esta parte veremos un sencillo ejemplo del uso de variables, mediante un ejemplo de una sencilla calculadora. Así mismo se verá cómo empezar a implementar funciones y poder llamarlas.

Se pueden utilizar dos diferentes palabras reservadas para la asignación de valores a través de *var* que trabaja con valores que pueden cambiar durante la ejecución de un programa, y *def* que es usada para mantener valores constantes en el ciclo de vida de un programa.

Otra característica es que no es necesario declarar el tipo de dato para una variable o constante, el compilador identifica de manera implícita el tipo de dato que es asignado a una variable. En este caso ya tienen asignado un valor. A continuación veremos un ejemplo

de un programa que simula una calculadora que se explicará, y en donde se hace uso de variables y funciones.

```
//uso de variables
def numero1 = 20;
def numero2 = 5;
var resultado;

//se mandan a llamar las funciones
sumar(numero1, numero2);
restar(numero1, numero2);
multiplicar(numero1, numero2);
dividir(numero1, numero2);

//declaración de funciones
function sumar(operando1: Integer, operando2: Integer) {
    resultado = operando1 + operando2;
    println("Suma \n {operando1} + {operando2} = {resultado}");
}

function restar(operando1: Integer, operando2: Integer) {
    resultado = operando1 - operando2;
    println("Resta \n {operando1} - {operando2} = {resultado}");
}

function multiplicar(operando1: Integer, operando2: Integer) {
    resultado = operando1 * operando2;
    println("Multiplicación \n {operando1} * {operando2} =
{resultado}");
}

function dividir(operando1: Integer, operando2: Integer) {
    resultado = operando1 / operando2;
    println("División \n {operando1} / {operando2} = {resultado}");
}
```

JavaFX Script utiliza funciones, que son bloques de código que llevan a cabo una tarea en específico. Se les pueden pasar parámetros a una función, para realizar cálculos o tratar con valores. En el siguiente código vemos como llamar a estas funciones pasándoles como parámetro las variables que declaramos antes.

```
sumar(numero1, numero2);
restar(numero1, numero2);
multiplicar(numero1, numero2);
dividir(numero1, numero2);
```

Las funciones reciben estos parámetros para realizar los cálculos e imprimir el resultado de cada operación; almacenando el resultado en la variable *resultado*.

```
function sumar(operando1: Integer, operando2: Integer) {
    resultado = operando1 + operando2;
    println("Suma \n {operando1} + {operando2} = {resultado}");
}

function restar(operando1: Integer, operando2: Integer) {
    resultado = operando1 - operando2;
    println("Resta \n {operando1} - {operando2} = {resultado}");
}

function multiplicar(operando1: Integer, operando2: Integer) {
    resultado = operando1 * operando2;
    println("Multiplicación \n {operando1} * {operando2} =
{resultado}");
}

function dividir(operando1: Integer, operando2: Integer) {
    resultado = operando1 / operando2;
    println("División \n {operando1} / {operando2} = {resultado}");
}
```

Como resultado se muestran estos resultados en la consola

```
Suma
 20 + 5 = 25
Resta
 20 - 5 = 15
Multiplicación
 20 * 5 = 100
División
 20 / 5 = 4
```

Así mismo las funciones pueden retornar valores tras ejecutar el bloque de código, se especifica el tipo de dato que debe retornarse y dentro de la función se usa la palabra *return* con el valor esperado, si una función no retorna un valor por Default es un :*Void*

```
function sumar(operando1: Integer, operando2: Integer) : Integer {
     resultado = operando1 + operando2;
     println("{operando1} + { operando2} = {resultado}");
     return resultado;
}
```

El valor que es retornado podemos asignarlo a otra variable.

```
var total;
total = sumar(11,200) + sumar(32,25);
```

Si deseamos compilar el programa Calculadora fuera del entorno de Netbeans, debemos compilar primero el archivo con extensión .fx utilizando la siguiente instrucción.

```
javafxc Calculatora.fx
```

Esto nos genera un archivo *Calculadora.class*, Y para ejecutarlo desde la línea de comandos. Utilizamos la siguiente instrucción

```
javafx Calculadora
```

Es posible pasar parámetros desde aquí

```
javafx Calculadora 100 50
```

Para ello necesitamos poder atrapar estos parámetros mediante la función *run()*, que recibe los parámetros en forma de una secuencia de objetos de tipo *String*, Una secuencia es una lista de objetos ordenados, muy similar a un arreglo.

```
function run(args : String[]) {

     // Hacemos las conversiones de tipo cadena a enteros
     def numero1= java.lang.Integer.parseInt(args[0]);
     def numero2= java.lang.Integer.parseInt(args[1]);

     // Invocamos nuestras funciones
      sumar(numero1, numero2);
      restar(numero1, numero2);
      multiplicar(numero1, numero2);
      dividir(numero1, numero2);
}
```

La conversión de tipo Cadena a Entero la realizamos llamando directamente código Java utilizando el método *parseInt* del objeto *Integer*.

La función *run()* es el método principal para inicializar un programa en JavaFX, el compilador genera implícitamente un método *run()* sin argumentos si no se declara como tal y coloca el código que se ejecutará dentro de él.

Como hemos visto en las variables se nos permite utilizar datos de diferentes tipo, y el compilador las asigna e inicializa de manera automática. Las funciones permiten que podamos estructurar nuestro código separando diferentes funcionalidades, además podemos utilizar tanto parámetros de entrada como de salida.

3.3 Uso de Objetos

Un objeto es una pieza de software que cuenta con atributos y con un comportamiento propio. Al trabajar con objetos se puede trabajar y organizar nuestro código de mejor manera, especialmente porque JavaFX Script es un lenguaje enfocado a la construcción de interfaces gráficas y casi todo está compuesto por uno o varios grupos de objetos que trabajan en conjunto.

Por otro lado podemos explotar otras características de este paradigma de programación como son la herencia, encapsulación, polimorfismo.

Para poder crear un objeto es necesario crear una instancia a partir de una clase, como el siguiente ejemplo.

Creamos una clase llamada CPU, donde el procesador, memoria y el disco duro son sus atributos, y también tiene funciones que imprimen sus características.

```
package jfx;

public class CPU {
    public var procesador: String;
    public var memoria: String;
    public var discoDuro: String;

    public function muestraProcesador() {
        println("Procesador: {procesador}");
    }

    public function muestraMemoria() {
        println("Memoria: {memoria}");
    }

    public function muestraDiscoDuro() {
        println("Disco Duro: {discoDuro}");
    }
}
```

Literales tipo objeto

Para poder crear o instanciar un objeto, es necesario crear una literal de tipo objeto para este caso creamos una literal llamada *pc* para crear una instancia de Computadora. Dentro de las llaves asignamos valores a los atributos del objeto computadora, para inicializar al objeto

```
var pc = CPU {
     procesador:"Core2 Duo"
     discoDuro:"500 GB"
     memoria:"4 GB"
}
```

Una vez creada la literal podemos acceder a sus atributos y funciones mediante el nombre de la literal seguido de un punto y el nombre del atributo o función.

```
pc.muestraProcesador();
pc.muestraDiscoDuro();
pc.muestraMemoria();
```

Para este ejemplo el resultado de llamar las funciones es el siguiente:

```
Procesador: Core2 Duo
Disco Duro: 500 GB
Memoria: 4 GB
```

Es posible también poder tener anidados objetos dentro de otros y poder acceder a ellos

```
var pc = Computadora {
     marca:  "HP";
     modelo: "550";
     precio: "12000"
     cpu: CPU {
           procesador:"Core2 Duo"
           discoDuro:"500 GB"
           memoria:"4 GB"
     }
}
```

Para este segundo ejemplo tenemos otra clase que tiene sus características propias y de la clase *CPU* que ya teníamos definida

```
package jfx;

public class Computadora {
     public var marca: String;
     public var modelo: String;
     public var precio: String;
     public var cpu: CPU;

    public function muestraMarca() {
        println("Marca: {marca}");
    }
    public function muestraModelo(){
        println("Modelo: {modelo}");
    }
    public function muestraPrecio(){
        println("Precio: {precio}");
    }
    public function muestraCPU(){
        println("Procesador: {cpu.procesador}");
        println("Memoria: {cpu.memoria}");
        println("Disco Duro: {cpu.discoDuro}");
    }
}
```

Teniendo nuestra literal *pc* podemos acceder a los atributos y funciones de la clase Computadora; y a su vez a las de la clase CPU.

```
pc.muestraMarca();
pc.muestraModelo();
pc.muestraPrecio();
pc.muestraCPU();  //esta función accede a los atributos
                  //de la clase CPU
```

Como Resultado de estas funciones tenemos los siguientes resultados:

```
Marca: HP
Modelo: 550
Precio: $12,000
Procesador: Athlon 64 x2
Memoria: 4 GB
Disco Duro: 400 GB
```

El poder tener objetos anidados nos permite poder generar estructuras y elementos más complejos; y también tener bien definidas las propiedades y comportamientos de cada uno de los elementos que pueden formar un gráfico más complejo.

3.4 Tipos de Datos

En JavaFX Script cuenta con varios tipos de datos parecidos a los de otros lenguajes de programación, además de otros tipos de datos basados en la duración del tiempo y otros tipos especiales que representan vacíos o nulos.

String

Es el tipo usado para declarar cadenas pódenos usar comillas simples o dobles para asignar un valor por ejemplo:

```
var cadena = "Hola Mundo";
var cadena = 'Hola Mundo';
```

También es posible colocar dentro de una cadena una variable o una expresión que nos devuelvan una cadena, para hacer esto colocamos la variable o expresión entre llaves {}

```
def persona = 'Daniel';
var saludo = "Hola {persona}"; // esto imprime:  Hola Daniel
```

Number e Integer

Estos representan tipo de datos numéricos, la diferencia entre los dos radica en que *Integer* solo trabaja con valores enteros; en cambio *Number* trabaja con números de tipo flotante, es decir con decimales. El compilador es el que decide qué tipo de dato es asignado, aunque se puede declarar explícitamente el tipo de dato aunque no es necesario, como se muestra

```
def numero1 = 1.0;      // el compilador lo interpreta como Number
def numero2 = 1;        // el compilador lo interpreta como Integer

def numero1: Number = 1.0;    //Se declara como de tipo Number
def numero2: Integer = 1;     //Se declara como de tipo Integer
```

Boolean

Son los tipos de datos que solo pueden representar uno de dos valores: verdadero o falso

```
var bandera = true;
```

Duration

Representan unidades de tiempo, y son usadas principalmente para efectos de animación que se verán en el siguiente capítulo. Algunos ejemplos son:

```
var t1 = 5ms;         // 5 milisegundos
var t2 = 10s;         // 10 segundos
var t3 = 30m;         // 30 minutos
var t4 = 1h;          // 1 hora
```

Void

En JavaFX Script todo es manejado como una expresión, dado que una expresión retorna un valor se necesita que se indique tipo de valor de retorno o en su caso un *Void* si no se está retornando nada. Este es un ejemplo utilizando *Void*:

```
function imprimeMensaje() : Void {
     println("No hay valor de retorno");
}
```

No es forzosamente necesario indicar un *Void*, ya que implícitamente el compilador detecta si no está regresando algún valor. Así que puede quedar de la siguiente manera:

```
function imprimeMensaje (){
     println("No hay valor de retorno ");
}
```

3.5 Secuencias

Las secuencias son una estructura que representa una lista ordenadas de objetos. Como tal una lista no es un objeto, es una variable que puede contener todo tipo de objetos o mejor dicho un conjunto de uno o más elementos.

Ahora declaramos una secuencia donde los elementos van entre corchetes separados por comas

```
def trimestre = ["Enero","Febrero","Marzo"];
```

Para poder acceder y ver los elementos de la secuencia, accedemos a ellos indicando la posición del elemento dentro de la secuencia

```
println("Elemento 1: {trimestre[0]}");
println("Elemento 2: {trimestre[1]}");
println("Elemento 3: {trimestre[2]}");
```

Esto imprime cada elemento de nuestra secuencia:

```
Elemento 1: Enero
Elemento 2: Febrero
Elemento 3: Marzo
```

En el caso anterior no tuvimos que declarar el tipo de datos para la secuencia; sin embargo, también es válido indicar el tipo de dato para los elementos de la secuencia. Por ejemplo declaramos una secuencia de elemento *Integer*:

```
def numeros: Integer[] = [1,2,3,4,5,6,8];
```

Esta secuencia no permite otro tipo de dato que no sea entero, al contrario de lo que sucede con nuestra primera secuencia (trimestre) donde podemos tener cualquier tipo de objeto.

Las secuencias pueden contener a otras secuencias y demás elementos, por ejemplo:

```
//incluimos la primer secuencia
def semestre = [trimestre,["Abril","Mayo","Junio"]];
println(semestre[0]);
println(semestre[1]);
println(semestre[2]);
println(semestre[3]);
println(semestre[4]);
println(semestre[5]);
```

La nueva secuencia contiene los elementos de la secuencia trimestre más otros tres elementos, esto nos imprime la siguiente lista

```
Enero
Febrero
Marzo
Abril
Mayo
Junio
```

Es válido utilizar notaciones como la del siguiente ejemplo, si es que queremos construir secuencias más grandes. La secuencia *num* tiene elementos desde el 1 hasta el 50.

```
def num = [1..50];
```

Uso de expresiones booleanas para llenar secuencias

En JavaFX Script es posible utilizar expresiones booleanas a partir de las cuales se evalúan elementos para poder formar una secuencia a partir de otra. Por ejemplo formaremos una nueva secuencia a partir de la anterior *num.*

```
def numMayoresDiez = num[n | n > 10];
```

La expresión dentro de los corchetes de la secuencia *num*, evalúa que cada elemento sea mayor que diez, esto específica que elementos serán copiados a la nueva secuencia. *numMayoresDiez* contiene los elementos desde el 11 al 50.

Operaciones con secuencias

Una vez que tenemos definida una secuencia podemos efectuar varias operaciones sobre ella, siempre y cuando la secuencia haya sido declarada como de tipo *var*, se pueden efectuar operaciones para insertar o borrar nuevos elementos en la secuencia. Cabe mencionar que deseamos insertar o borrar un elemento, la secuencia es destruida y creada de nuevo con el mismo nombre. En realidad no estamos trabajando con la misma secuencia sino que estamos destruyendo la primera y creando una nueva.

Para insertar elementos en una cadena podemos hacerlo de diferentes formas, insertando solamente al final de la secuencia con las palabras reservadas *insert* e *into*. O también podemos indicar a partir de qué índice vamos a insertar el elemento, si lo queremos hacer antes (*before*) o después (*after*) del índice. Para la siguiente secuencia se muestran diferentes maneras para insertar elementos.

```
var dias = ["lunes", "miercoles"];
println(dias);

insert "viernes" into dias;  //inserta día vienes como el último
println(dias);               //elemento

insert "martes" after dias[0]; //inserta el día martes después del
println(dias);                 //elemento [0] lunes

insert "jueves" before dias[3];    //inserta el día jueves antes del
println(dias);                     //elemento [3] viernes
```

A la hora de mostrar los resultados, podemos ver cómo cambia la secuencia de días

```
[ lunes, miercoles ]
[ lunes, miercoles, viernes ]
[ lunes, martes, miercoles, viernes ]
[ lunes, martes, miercoles, jueves, viernes ]
```

La forma de eliminar elementos de una secuencia se indica con la palabra *delete*, el nombre del elemento a borrar, la palabra *from* y el nombre de la secuencia. O se puede indicar el índice donde se ubica el elemento en la secuencia

```
delete "viernes" from dias; //elimina el elemento "viernes" de la
secuencia
println(dias);

delete dias[0];             //elimina el elemento "lunes" de la
secuencia
println(dias);
```

Ahora bien para borrar todos los elemento de la secuencia se indica solo con la palabra *delete* y el nombre de la secuencia, como se muestra a continuación

```
delete dias;                //elimina todos los elementos de la
secuencia
println(dias);
```

El resultado de borrar estos elementos de la secuencia es el siguiente:

```
[ lunes, martes, miercoles, jueves ]
[ martes, miercoles, jueves ]
[ ]                          //secuencia vacía
```

Otra operación que podemos aplicar a una secuencia es invertir el orden de sus elementos usando el operador *reverse*, por ejemplo

```
var num  = [1..10];
reverse num; // devuelve [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Es posible comparar dos secuencias directamente, primero se compara el tamaño de la secuencia y después los elementos de cada secuencia.

```
def s1 = [1..5];                    //secuencia del 1 al 5
def s2 = [1..6];                    //secuencia del 1 al 6
def s3 = [5..10];                   //secuencia del 5 al 10
def s4 = [1..5];                    //secuencia del 1 al 5
println("s1 == s2 -> {s1 == s2}"); //secuencias de diferente tamaño
println("s1 == s3 -> {s1 == s3}"); //secuencias con diferentes
elementos
println("s1 == s4 -> {s1 == s4}"); //secuencias iguales
```

La comparación de estas cadenas imprime los siguientes resultados:

```
s1 == s2 -> false
s1 == s3 -> false
s1 == s4 -> true
```

Subsecuencias

A veces es útil poder generar una nueva secuencia a partir de otra, especificando solamente mediante una expresión los elementos que deseamos de tal secuencia.

Creamos la secuencia *finSemana* a partir de los de los índices indicados para la secuencia semana.

```
def semana =
["lunes","martes","miercoles","jueves","viernes","sabado","domingo"];
var finSemana = semana[5..6];
println(finSemana);                 //[ sabado, domingo ]
```

Se obtiene una secuencia de los elementos entre el primer índice indicado y todos los elementos menores al elemento del índice 5.

```
def entreSemana = semana[0..<5];
println(entreSemana);//[ lunes, martes, miercoles, jueves, viernes ]
```

Si se omite el segundo índice se toman todos los elementos a partir del primer índice hasta el último elemento de la secuencia.

```
finSemana = semana[5..];
println(finSemana);                // [ sabado, domingo ]
```

Para este caso se toman todos los elementos contenidos entre el primer índice y el penúltimo elemento, exceptuando al último.

```
def dias2 = semana[0..<];
println(dias2); //[lunes, martes, miercoles, jueves,viernes, sabado]
```

3.6 Operadores

JavaFX Script provee varios tipos de operadores para poder realizar diferentes tipos de operaciones. Se cuenta con operadores aritméticos, unitarios, relacionales, condicionales y de comparación.

Operadores aritméticos

Realizan las operaciones básicas de suma, resta, multiplicación, división y módulo entre dos operandos.

+	Suma
-	Resta
*	Multiplicación
/	División
mod	Residuo de una división

Operadores Unitarios

Se utilizan junto a un solo operando o valor, y sirven para incrementar, decrementar o negar su valor.

-	niega un número
++	incrementa un valor en 1
--	Decrementa un valor en 1
not	operador lógico, invierte un valor booleano

Algunos ejemplos de cómo usar estos operadores son los siguientes:

```
var resultado = 1;          // resultado es 1

resultado--;                // resultado es 0
println(resultado);
resultado++;                // resultado es 1
println(resultado);
resultado = -resultado;     // resultado es -1
println(resultado);
var success = false;
println(success);           // false
println(not success);       // true
```

Operadores de equivalencia y relacionales

Son útiles para comparar dos operandos; si son iguales, diferentes, mayores o menores respecto al otro.

==	Igual a
!=	Diferente a
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Operadores Condicionales

Son usadas para poder evaluar dos expresiones booleanas, y poder cumplir alguna condición.

and	se usa para establecer que si y solo si dos expresiones booleanas son verdaderas se cumpla la condición
or	Indica que con solo con una de las dos expresiones booleanas sea verdadera, se cumple la condición

El siguiente ejemplo demuestra el uso de *and* y *or*

```
def usuario = "user";
def password = "admin";

//se cumple la condición
if ((usuario == "user") and (password == "admin")) {
     println("Prueba 1: usuario y password son correctos");
}

//no cumple la condición
if ((usuario == "") and (password == "admin")) {
println("Prueba 2: usuario y password son correctos");
}

//se cumple la condición
if ((usuario == "user") or (password == "admin")) {
println("Prueba 3: usuario o password son correctos");
}

//se cumple la condición
if ((usuario == "") or (password == "admin")) {
println("Prueba 4: usuario o password son correctos");
}
```

Imprimen en pantalla lo siguiente:

```
Prueba 1: usuario y password son correctos
Prueba 3: usuario o password son correctos
Prueba 4: usuario o password son correctos
```

Operador de Comparación de Tipo

Compara el tipo de objeto o instancia de un valor o variable, y verifica que sea igual al tipo que se le indica.

Por ejemplo se tiene una cadena y con operador *instanceof* se verifica si la cadena es de tipo *String*, lo cual nos devuelve un *true*.

```
def str1="Saludos";
println(str1 instanceof String);     // imprime true
```

3.7 Expresiones

Las expresiones son bloques de código que retornan un valor. JavaFX Script es un lenguaje basado en expresiones, como se mencionó antes implícitamente cada expresión en JavaFx Script tiene a *Void* como valor de retorno por defecto. Esto quiere decir que cualquier bloque de código, incluyendo ciclos como *for* y *while* devuelven un valor vacío.

Bloques de expresiones

Son un conjunto de declaraciones o expresiones encerradas entre llaves {}, donde cada declaración o expresión está separada por punto y coma";". El valor para toda la expresión es el valor que toma en la última declaración o expresión interna, de lo contrario devuelve un *Void*.

El siguiente es un ejemplo de cómo una expresión hace la sumatoria de los elementos de la secuencia *nums*, y devuelve el valor de la sumatoria a través de la variable *sum*, por último el resultado del bloque es asignado a la variable *total* y es mostrado en pantalla.

```
var nums = [2, 4, 6, 8];
var total = {
     var sum = 0;
     for (a in nums) { sum += a };
     sum;
}
println("El total es {total}.");
```

El resultado que se muestra es el siguiente:

```
El total es 20.
```

Expresión if

La expresión *if* permite realizar una evaluación para poder decidir el flujo del programa, en caso de que la expresión sea verdadera o devuelva un *true* se continúa con la ejecución del código contenido dentro del *if*, de lo contrario salta este bloque de código.

En este ejemplo se evalúa la variable *edad* y dependiendo las condiciones se asigna un valor a la variable *entradaCosto* o se le da un valor por defecto.

```
def edad = 20;
var entradaCosto;

if (edad < 5 ) {                          //devuelve false
     entradaCosto = 0;
} else if (edad < 12 or edad > 65) {     //devuelve false
     entradaCosto = 25;
} else {             //entra por defecto en caso de no cumplirse las
     entradaCosto = 50;      //condiciones anteriores
}
println(entradaCosto);       //imprime 50
```

Rangos de Expresiones

Como se vio anteriormente podemos definir los elementos de una secuencia especificando un intervalo; a esto también se le conoce como un rango de expresiones, por defecto los intervalos son de 1, pero podemos especificar mediante la palabra reservada *step* cual será nuestro intervalo, incluso se puede hacer en orden descendente con valores negativos. Estos son algunos ejemplos:

```
var numeros = [1..10]; //secuencia de 1 a 10 en intervalos de 1
println(numeros);

numeros = [1..10 step 2];    //secuencia de 1 a 10 en intervalos de 2
println(numeros);

numeros = [10..1 step -1];   //secuencia de 10 a 1
println(numeros)
```

Lo que se muestra en pantalla es lo siguiente:

```
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
[ 1, 3, 5, 7, 9 ]
[ 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 ]
```

Expresión for

Este mecanismo es una forma muy útil para poder iterar sobre los elementos de una secuencia. Un sencillo ejemplo es:

```
var dias =
["lunes","martes","miercoles","jueves","viernes","sabado","domingo"];
for (dia in dias) {
     println(dia);
}
```

Se almacena cada elemento de la secuencia en la variable *dia* e imprime cada elemento en la iteración:

```
Lunes
martes
miercoles
jueves
viernes
sabado
domingo
```

Expresión while

Es otra forma en la que podemos iterar sobre un secuencia, a diferencia de un *for* el ciclo *while* continúa iterando hasta que la condición booleana que está evaluando llega a *false*.

```
var contador = 0;
while (contador < 10) {
    println("contador = {contador}");
    contador += 1;
}
```

El ciclo imprime el valor del contador en cada iteración:

```
contador = 0
contador = 1
contador = 2
contador = 3
contador = 4
contador = 5
contador = 6
contador = 7
contador = 8
contador = 9
```

Expresiones break y continue

Estas expresiones están relacionadas con los bucles o ciclos, estas afectan la forma en cómo itera un ciclo. Al utilizar *break* se interrumpe definitivamente el ciclo, mientras que *continue* interrumpe sólo la iteración actual y continua con la siguiente. En el siguiente ejemplo vemos como funcionan estas expresiones.

```
for (i in [0..10]) {   //secuencia de 1 a 10
     if (i > 5) {
          break;       //interrumpe ciclo cuando i es mayor a 5
     }
     if (i mod 2 == 0) {
          continue;    //interrumpe iteración cuando i vale 2 y 4
     }
     println(i);       //imprime valor de i
}
```

Se tiene una secuencia de 10 elementos que se va recorriendo con ayuda de un ciclo *for*, el ciclo se interrumpe completamente cuando el valor de la variable *i* es mayor a cinco, por otro lado si el residuo resultante de dividir el valor de *i* entre dos es igual a cero se interrumpe la iteración con *continue*. Así es como tenemos por resultado los siguientes elementos:

```
1
3
5
```

Expresiones throw, try, catch y finally

El uso de estas expresiones es una forma muy útil para controlar errores que se pueden presentar en tiempo de ejecución. Podemos hacer uso del control de excepciones al importar objetos del tipo *Exception* del lenguaje Java.

```
import java.lang.Exception;

function lanzaExcepcion() {
    throw new Exception("Se arroja nueva Excepción!");
}
try {
     lanzaExcepcion();
} catch (e: Exception) {
     println("{e.getMessage()} \nExcepción atrapada en CATCH");
} finally {
     println("Mensaje desde FINALLY");
}
```

Una excepción puede generarse inesperadamente durante la ejecución de un programa, o podemos provocarla intencionalmente mediante el uso de la palabra reservada *throw*, la cual genera un nuevo objeto del tipo *Exception*. Para este ejemplo creamos una función que lanza una excepción.

```
function lanzaExcepcion() {
    throw new Exception("Se arroja nueva Excepción!");
}
```

Para poder atrapar y tratar la excepción que generamos, encerramos la llamada a la función *lanzaExcepcion()* entre la clausula *try* e inmediatamente después declaramos el bloque *catch* donde se especifica el tipo de excepción que se puede provocar y entre las llaves podemos definir algún tratamiento para el error o simplemente mostrar un mensaje.

```
try {
     lanzaExcepcion();
} catch (e: Exception) {
     println("{e.getMessage()} \nExcepción atrapada en CATCH");
} finally {
     println("Mensaje desde FINALLY");
}
```

El uso de la clausula *finally* es de uso opcional, se declara después de la llave que cierra el bloque del *catch*, *finally* se utiliza para ejecutar algún bloque de código adicional sin importar si se generó o no una excepción dentro del bloque del *try*.

Al ejecutar el código anterior obtenemos los siguientes mensajes

```
Se arroja nueva Excepción!
Excepción atrapada en CATCH
Mensaje desde FINALLY
```

3.8 Enlace de Datos (Data Binding) y Disparadores (Triggers)

JavaFX Script tiene la capacidad de establecer directamente una relación directa entre dos variables, de tal forma que si el valor de una cambia afecta directamente a la otra. También podemos establecer la ejecución automática de ciertos fragmentos de código cuando el valor de una variable cambia, a esta relación la definimos como un *trigger*. Estas

características de JavaFX Script son fundamentales a la hora de crear relaciones entre elementos u objetos gráficos.

La palabra reservada *bind* asocia el valor de una variable con otra variable, una función o cualquier expresión. En el siguiente ejemplo se asocian dos variables, la variable *b* toma automáticamente el valor de *a* cada vez que este cambia, *b* solo puede tomar un valor de *a* ya que está declarada como *def* y no se le puede asignar un valor directamente.

```
var a = 0;
def b = bind a;
a = 10;
println(b);      // el valor de b es 10
a = 20;
println(b);      // el valor de b es 20
```

De la misma manera podemos enlazar un objeto con una variable de tipo *Object*, en el siguiente ejemplo creamos una instancia de la clase *CPU*, inicializando algunas variables para establecer los atributos del objeto. Si después decidimos modificar el valor de alguna variable, se estará modificando por igual a los atributos del Objeto que se asocio.

```
var procesador = "Core 2 Duo";
var memoria = "2 GB";
var discoDuro = "500 GB";

def cpu = bind CPU{
        procesador:procesador;
        memoria:memoria;
        discoDuro:discoDuro;
    }
println("CPU Procesador: {cpu.procesador}");
procesador = "Athon 64 x2";
println("CPU Procesador: {cpu.procesador}");
```

Al ejecutar el ejemplo anterior podemos observar cómo cambia el atributo *procesador* del objeto *cpu*.

```
CPU Procesador: Core 2 Duo
CPU Procesador: Athon 64 x2
```

Enlazar funciones

Para enlazar funciones hacemos uso las palabras *bind* y *bound* que trabajan en conjunto. Para enlazar una función debemos declararla con palabra *bound* al principio y al llamarla se debe de usar como *bind*.

En este ejemplo declaramos una función *escalaPunto* que tiene como tarea crear una Instancia del objeto *Puntos*, pasándole como parámetros los valores de *x*, *y* para inicializar los atributos de este objeto.

```
var escala = 1.0;

bound function escalaPunto(xPos : Number, yPos : Number) : Puntos {
     Puntos {
          x: xPos * escala
          y: yPos * escala
     }
}
class Puntos {
     var x : Number;
     var y : Number;
}
```

Se establecen los valores para *x*, *y*.

```
var iX = 2.0;
var iY = 2.0;
```

Se declara una variable de tipo objeto que contendrá la instancia que devuelve la función *escalaPunto*, y enlazamos la función con *bind*; después se imprimen los valores de los atributos del objeto.

```
def pt = bind escalaPunto(iX, iY);
println("Punto X: {pt.x}");        //imprime “Punto X: 2.0”
println("Punto Y: {pt.y}");        //imprime “Punto Y: 2.0”
```

Después se cambian los valores para *iX* y *escala*, esto cambia directamente los valores de los atributos del objeto *pt* que esta enlazado a la función *escalaPunto*.

```
iX = 5.0;
println("Punto X: {pt.x}");
println("Punto Y: {pt.y}");
escala = 2.0;
println("Punto X: {pt.x}");
println("Punto Y: {pt.y}");
```

Cada vez que se modifica un argumento o alguna variable que sea utilizada por la función, esta se vuelve a invocar. Al imprimir los resultados podemos observar cómo varía el valor de *pt.x* al cambiar los valores del argumento *iX* y la variable *escala*.

```
Punto X: 2.0
Punto Y: 2.0
Punto X: 5.0
Punto Y: 2.0
Punto X: 10.0
Punto Y: 4.0
```

Enlazar Secuencias

Es posible enlazar dos secuencias con el uso de *bind*. Para este ejemplo se tiene una primera secuencia de elementos del 1 al 5; con el uso de la palabra reservada *for* se define una segunda secuencia en donde para cada elemento de la primer secuencia le corresponde otro en la segunda secuencia, con *item*item* se define que la segunda secuencia contendrá los cuadrados de los elementos de la primer secuencia

```
var secuencia1 = [1..5];
def secuencia2 = bind for (item in secuencia1) item * item;
imprimeSecuencias();

function imprimeSecuencias() {
     println("Primer secuencia:");
     for (i in secuencia1){println(i);}
     println("Segunda secuencia:");
     for (i in secuencia2){println(i);}
}
```

Al llamar la función *imprimeSecuencias()* recorremos cada secuencia imprimiendo el valor de sus elemento, como se muestra a continuación:

```
Primer secuencia:
1
2
3
4
5
Segunda secuencia:
1
4
9
16
25
```

Si se realiza un cambio sobre la primer secuencia la segunda se ve afectada de la misma manera, para este caso insertamos un nuevo elemento en la primer secuencia por lo cual en la segunda secuencia también se redefinen sus elementos.

```
insert 6 into secuencia1;
imprimeSecuencias();
```

Como resultado se puede observar el nuevo elemento en las dos secuencias.

```
Primer secuencia:
1
2
3
4
5
6
Segunda secuencia:
1
4
9
16
25
36
```

Triggers (Disparadores)

Los *Trigger* son bloques de código que están asociados a una variable, cuando el valor de esta variable cambia por alguna razón el bloque de código del *trigger* es ejecutado. Para indicar el bloque de código como *trigger* se declaran las palabras *on replace* antes de

las llaves y un nombre cualquiera para una variable que contiene el valor original de la primer variable antes de que se ejecute el *trigger*.

Para el siguiente ejemplo el código entre llaves "{}" se ejecuta por primera vez cuando inicializamos la variable *mensaje*, y se vuelve a ejecutar o lanzar cuando cambiamos el valor de la variable *mensaje*.

```
var mensaje = "Probando trigger..." on replace valorOriginal {
     println("\n{valorOriginal}");
     println(mensaje);
};
mensaje = "Cambio mensaje, se lanza el tigger";
```

Este ejemplo imprime los siguientes mensajes:

```
Probando trigger...

Probando trigger...
Cambio mensaje, se lanza el tigger
```

3.9 Clases

Como se vio anteriormente al momento de crear objetos definimos algunas clases. Estas tienen atributos y funciones que son propios, estos atributos definen el comportamiento de la clase. Al igual que otros lenguajes de programación en JavaFX Script podemos aplicar la herencia entre clases y sobrescribir o agregar funcionalidades.

En el siguiente ejemplo se declara una clase de tipo *abstract*, este tipo de clases no pueden ser instanciadas directamente pero nos permiten generalizar algunas funciones básicas, podemos usar las clases abstractas para crear otras clases que hereden de ella funciones y atributos en común.

```
package jfx;

abstract  class CuentaBancaria {
     var cuentaNumero: Integer;
     var saldo: Number;

     function obtenerSaldo(): Number {
          return saldo;
     }
     function depositar(monto: Number): Void {
          saldo += monto;
     }
     function retirar(monto: Number): Void {
          saldo -= monto;
     }
}

class CuentaAhorro extends CuentaBancaria {

     var saldoMin = 1000.00;
     var comision = 150.00;

     function checarSaldoMinimo() : Void {
          if(saldo < saldoMin){
               saldo -= comision;
          }
     }

     override function retirar(monto: Number) : Void {
          if(saldo-monto<0){
                //código para impedir la operacion y notificar
          } else {
                saldo -= monto;
          }
     }
}//class
```

En este ejemplo se declara una clase abstracta llamada *CuentaBancaria*, esta tiene los atributos y funciones básicas para cualquier tipo cuenta. Se tiene otra clase llamada *CuentaAhorro* que mediante uso de palabra reservada *extends* se indica que esta clase hereda todas las propiedades y funciones de la clase *CuentaBancaria*.

Además de heredar todas las propiedades de *CuentaBancaria* se pueden definir nuevos métodos, o también sobrescribir los existentes en caso de ser necesario; la sobrescritura de funciones se puede hacer antecediendo el nombre de la función con la palabra *override*.

En el siguiente capítulo se analizarán las opciones que ofrece JavaFX Script para poder generar interfaces gráficas, creando figuras básicas y usando imágenes existentes, así como aplicar efectos de color, reflejo, animación, etc.

CAPÍTULO 4
CONSTRUCCIÓN DE INTERFACES GRÁFICAS DE USUARIO (GUI)

3.1 Introducción a las GUI

A través de este capítulo se verá cómo construir objetos gráficos y poder interactuar con estos, a estos objetos se pueden aplicar diversos efectos de iluminación, sombra, degradado de colores, entre otros efectos.

Se verá la manera de enlazar objetos entre sí y poder establecer una interacción entre ellos, por otro lado se utilizarán propiedades y elementos basados en el tiempo para poder establecer un efecto de animación para los objetos gráficos.

Por último se verá como poder captar e interpretar diferentes eventos que afectan de forma directa el comportamiento de los objetos gráficos que están enlazados.

3.2 Uso de la Sintaxis Declarativa

Como se mencionó anteriormente JavaFX Script es un lenguaje de sintaxis declarativa; codificar de esta manera permite diferenciar de una forma más clara la estructura del código para la construcción de interfaces gráficas.

En el siguiente ejemplo se dibujaran dos figuras geométricas, un círculo y un rectángulo, mediante el siguiente código el cual se explicará por partes.

```
package jfx;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;

Stage {
    title: "Usando la Sintaxis de JavaFX "
    scene: Scene {
        width: 300
        height: 250
        content: [
            Circle {
                centerX: 150  centerY: 120 radius: 80
                fill: Color.YELLOW
                stroke: Color.ORANGE
                strokeWidth: 7.0
            }, //Circulo
            Rectangle {
                x: 25, y: 80 width: 250, height: 80
                arcWidth: 70 arcHeight: 50
                fill: Color.web("#6699ff")
                stroke: Color.web("#003399")
                strokeWidth: 5.0
            } //Rectangulo
        ] //Content
    } //Scene
} //Stage
```

En primer lugar se declara el paquete donde está ubicado el archivo *.fx* y las librerías que se utilizarán para poder dibujar la ventana, la escena para desplegar los elementos gráficos, el objeto *Color* y las figuras de la librería *javafx.scene.shape*.

```
package jfx;
import javafx.stage.Stage;  //ventana
import javafx.scene.Scene;  //escena para desplegar las figuras
import javafx.scene.shape.Rectangle; //rectángulo
import javafx.scene.paint.Color;     //para colorear las figuras
import javafx.scene.shape.Circle;    //círculo
```

Se instancian los objetos *Stage* para crear la ventana y *Scene* que es el área principal para colocar los componentes gráficos. Al crear el objeto Scene se establecen las propiedades del ancho, alto y el *content* en donde se colocan las figuras geométricas.

```
Stage {
    title: "Usando la Sintaxis de JavaFX "
    scene: Scene {
        width: 300
        height: 250
        content: [...]
   }
}
```

Dentro de *content* se instancian las dos figuras geométricas que dibujaremos, de las clases *Circle* y *Rectangle*. Estos objetos provienen del paquete *javafx.scene.shape*, aparte de estas figuras podemos encontrar otras demás, cada una con características propias.

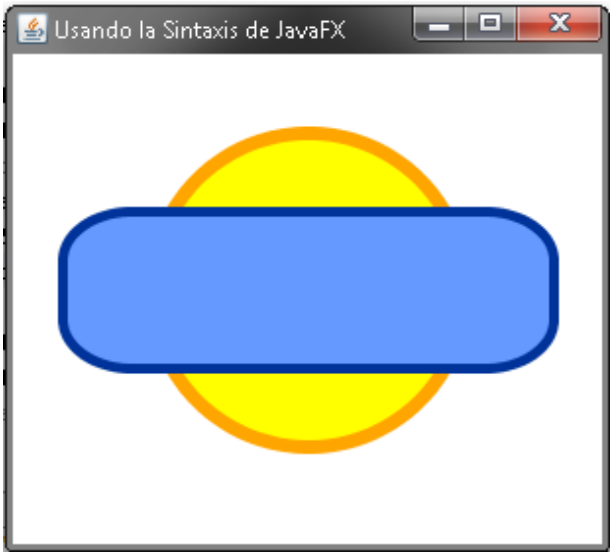
Para el caso de *Circle* establecemos la posición del centro del círculo y el radio esto está dado en pixeles; para *Rectangle* también se estable el centro de la figura, el ancho y el largo del rectángulo, además establecemos valores para redondear las esquinas del rectángulo con las propiedades *arcWidth* y *arcHeight*.

Para ambas figuras se establece el color de relleno (propiedad *fill*), el ancho y el color para la línea perimetral de la figura (*stroke* y *strokeWidht*); para colorear los objetos se hace uso de la clase *Color*, al instanciar un objeto de este tipo ya tenemos algunos colores preestablecidos, pero podemos definir otros colores más específicos indicando por

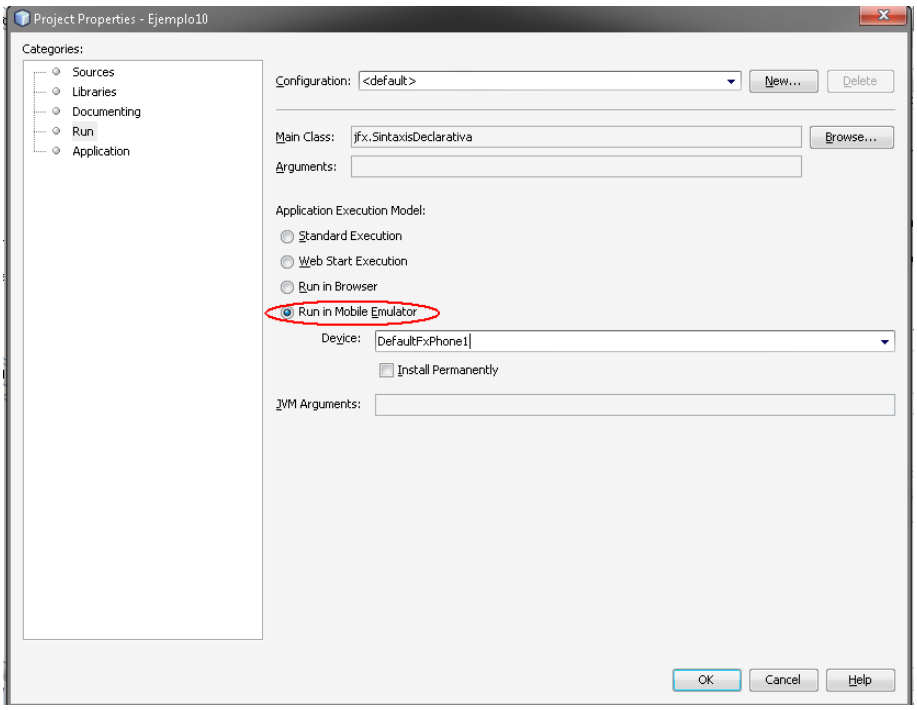
ejemplo un color web mediante su código en hexadecimal o también se pueden indicar otros colores mediante la combinación de tres valores *RGB* (valores rojo, verde y azul).

```
Content: [
    Circle {
        centerX: 150  centerY: 120 radius: 80 //posición y radio
        fill: Color.YELLOW
        stroke: Color.ORANGE
        strokeWidth: 7.0
    }, //Circulo
    Rectangle {
        x: 25, y: 80 width: 250, height: 80 //posición, alto y ancho
        arcWidth: 70 arcHeight: 50          //esquinas redondeadas
        fill: Color.web("#6699ff")
        stroke: Color.web("#003399")
       strokeWidth: 5.0
    } //Rectángulo
] //Content
```

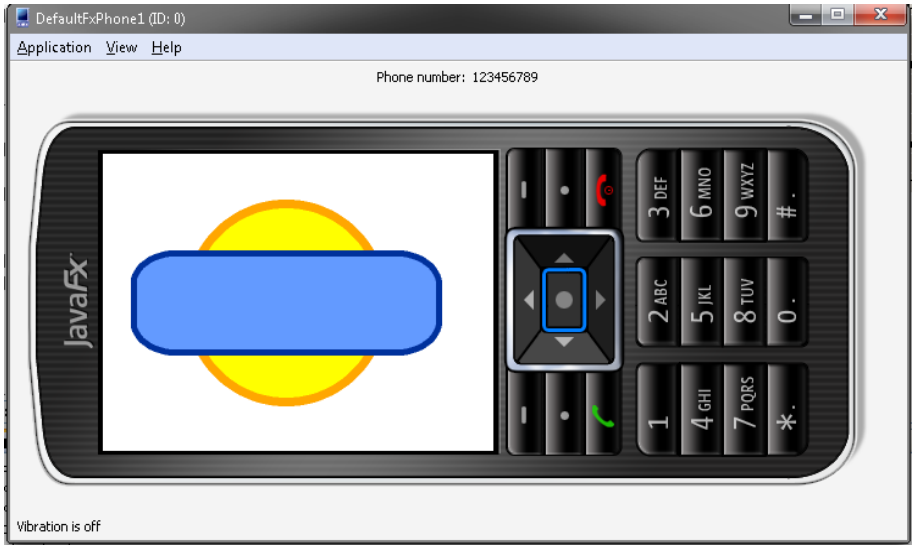
En la siguiente imagen podemos ver la imagen resultante, donde se ve el círculo y el rectángulo, estos aparecen en el orden que se declararon, por eso el rectángulo esta sobrepuesto al círculo.



Alternativamente podemos ejecutar la aplicación desde el modo de emulación para dispositivos móviles, para esto bebemos dar clic derecho al proyecto y seleccionar *Project Properties*, una vez ahí seleccionamos la categoría *Run* y por último seleccionamos el radio botón que dice: *Run in Mobile Emulator*.



Al ejecutar la aplicación en modo móvil nos mostrará la misma aplicación como la siguiente imagen dentro de un celular.



3.3 Escena Gráfica y Presentación de Objetos

La escena gráfica es el principal elemento para poder colocar figuras, imágenes, texto y elementos multimedia; actúa como un contenedor y en la cual podemos establecer una estructura de tipo árbol donde se define la jerarquía y orden de los objetos.

Los objetos gráficos individualmente son denominados como nodos, estos pueden tener nodos hijos o padres. La clase *javafx.scene.Node* es la clase base para los objetos gráficos, por ejemplo clases como *Circle* y *Rectangle* heredan de ella y toman propiedades como la posiciones de las coordenadas *X* y *Y*.

Los Nodos pueden ser agrupados mediante el uso de la clase *javafx.scene.Group*, esto permite conjuntar varios objetos y poder hacer que se comporten como una sola entidad, esto da la ventaja de poder aplicar un mismo efecto a todo el grupo.

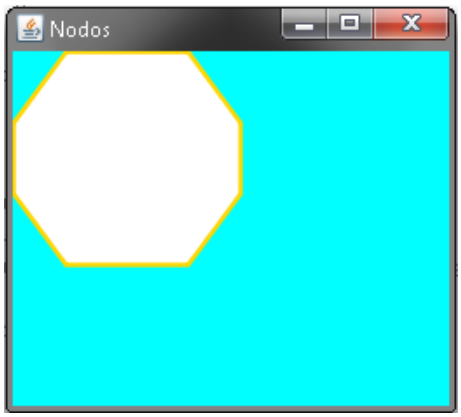
Java FX provee una amplia variedad de efectos visuales que podemos utilizar de las siguientes paqueterías *javafx.scene.effect* y *javafx.scene.effect.light.* A continuación se desarrollará un ejemplo usando tres nodos, para después agruparlos.

Se crearán tres elementos gráficos: un Polígono, un objeto de tipo Texto y se hará uso de una imagen; después se agruparán los tres elementos y los trasladaremos de posición.

Primero creamos un Polígono usando la clase *javafx.scene.shape.Polygon*, se establecen las propiedades para el contorno y el color de relleno de la figura, después se establecen las coordenadas para cada punto que conforman la figura mediante la propiedad *points*.

```
Polygon {
   stroke: Color.GOLD
   strokeWidth: 2.5
   fill: Color.WHITE
   points: [
             30.0, 0.0,
             0.0, 40.0,
             0.0, 80.0,
             30.0, 120,
             100.0, 120.0,
             130.0, 80.0,
             130.0, 40.0,
             100.0, 0.0
    ]
},
```

La figura resultante es un octágono como el siguiente:

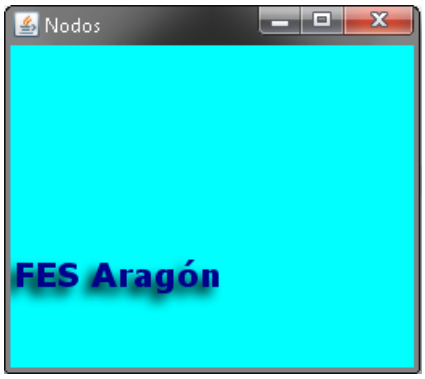


El siguiente elemento a crear es un objeto de tipo texto de la clase *javafx.scene.text.Text*. Se estable la posición del texto y hacemos uso de un efecto de tipo sombra *javafx.scene.effect.DropShadow*, se establecen los valores en pixeles de la posición de la sombra con respecto al texto original. Por último indicamos el contenido del texto, el color de relleno y el tipo de fuente.

Para poder establecer la fuente hacemos uso de los objetos *Font* y *FontWeight* del paquete *javafx.scene.text*. Al crear el objeto *Font* se le pasan como parámetros la familia de la fuente, la anchura determinada por *FontWeight*, y el tamaño de la fuente. Existen varios tipos de constructores al crear un objeto de tipo Font. El código es el siguiente:

```
Text {
    x: 2
    y: 150
    effect: DropShadow {
        offsetX: -5
        offsetY:  5
    }
    content: "FES Aragón"
    fill: Color.DARKBLUE
    font: Font.font("Verdana", FontWeight.BOLD, 20);
},
```

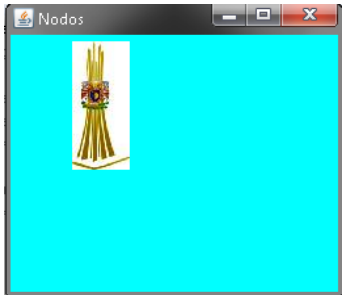
La siguiente figura muestra el texto resultante y los efectos aplicados



Por último se creará un nodo de tipo imagen usando las clases *ImageView* e *Image* del paquete *javafx.scene.text*, en *ImageView* se establece la posición de la imagen y en la propiedad *image* creamos el objeto *Image* especificando la ruta de la imagen, se pueden usar tanto imágenes de internet como imágenes en un directorio local, la variable *_DIR_* indica la ruta del directorio donde se encuentra el proyecto y el archivo, posteriormente se indica el nombre y extensión de la imagen, JavaFX acepta imágenes de los siguientes tipos PNG, GIF, y JPEG.

```
ImageView {
    x: 47
    y: 5
    image: Image {url: "{__DIR__}torres.jpg"}
}//ImageView
```

La imagen mostrada es la siguiente.



Después de generar los tres objetos anteriores, los integramos todos juntos dentro del *content* de la Escena en el orden que se crearon y al final se mostrarán de la siguiente manera:

Para poder agrupar los tres nodos en un grupo hacemos uso de la clase *javafx.scene.Group* envolviendo la declaración de los tres nodos dentro de las llaves del objeto *Group*, de esta manera podemos tratar los tres nodos como una sola entidad. Por último trasladamos al grupo de posición con las propiedades *translateX* y *translateY*. El código del uso de la clase *Group* es el siguiente:

```
  Stage {
      title: "Nodos"
      scene: Scene {
          width: 250
          height: 200
          fill: Color.CYAN
          content: Group {
              translateX: 55
              translateY: 10
              content: [
                  ....        //código de los nodos
              ]//Content
          }//Group
      }//Scene
}//Stage
```

Al mover de posición el grupo queda como la imagen que se muestra a continuación:

El código completo del ejemplo es el siguiente:

```
package jfx;
 import javafx.stage.Stage;
 import javafx.scene.Scene;
 import javafx.scene.Group;
 import javafx.scene.shape.Polygon;
 import javafx.scene.paint.Color;
 import javafx.scene.image.Image;
 import javafx.scene.image.ImageView;
 import javafx.scene.text.Text;
 import javafx.scene.text.Font;
 import javafx.scene.text.FontWeight;
 import javafx.scene.effect.DropShadow;
 Stage {
     title: "Nodos"
     scene: Scene {
         width: 250
         height: 200
         fill: Color.CYAN
         content: Group {
             translateX: 55
             translateY: 10
             content: [
                 Polygon {
                     stroke: Color.GOLD
                     strokeWidth: 2.5
                     fill: Color.WHITE
                     points: [
                          30.0, 0.0,
                          0.0, 40.0,
                          0.0, 80.0,
                          30.0, 120,
                          100.0, 120.0,
                          130.0, 80.0,
                          130.0, 40.0,
                          100.0, 0.0
                     ]
                 },
                 Text {
                     x: 2
                     y: 150
                     effect: DropShadow {
                         offsetX: -5
                         offsetY:  5
                     }
                     content: "FES Aragón"
                     fill: Color.DARKBLUE
                     font: Font.font("Verdana", FontWeight.BOLD, 20);
                 },
                 ImageView {
                     x: 47
                     y: 5
                     image: Image {url: "{__DIR__}torres.jpg"}
                 }//ImageView
            ]//Content
         }//Group
     }//Scene
 }//Stage
```

3.4 Construcción de Objetos Gráficos

En esta apartado se verá cómo construir elementos gráficos y aplicar algunos efectos que provee JavaFX, a través de estos efectos podemos enriquecer la apariencia visual de cualquier objeto. En el siguiente ejemplo dibujaremos un botón, y aplicaremos efectos de reflexión y degradados de color para pintar las figuras.

El botón se compone básicamente de un rectángulo y un círculo, podemos crear primero los objetos asignándolos a una variable de tipo objeto para después incluirlos dentro del *content* de la Escena. Primero creamos un Rectángulo aplicando un degradado de color mediante el uso de la clase *javafx.scene.paint.LinearGradient*, el código para generar el rectángulo es el siguiente:

```
var rect =  Rectangle {
    x: 40         y: 55
    width: 150    height: 50
    arcWidth: 20  arcHeight: 55
    stroke: Color.BLACK
    fill: LinearGradient {
        startX: 0.0,
        startY: 0.0,
        endX: 0.3,
        endY: 1.0,
        proportional: true
        stops: [
            Stop {
                offset: 0.0
                color: Color.WHITE},
            Stop {
                offset: 1.0
                color: Color.BLACK}
        ]
    } // fill
}
```

Declaramos una instancia de *Rectangle* y establecemos valores a las propiedades como las que ya habíamos visto: posición (x, y), esquinas arqueadas y color del contorno; ahora en la propiedad *fill* es donde declaramos el elemento *LinearGradient* que representa el efecto de color degradado, este objeto cuenta con las siguientes propiedades:

- *startX*, *startY*, *endX*, *endY*. Estas variables permiten controlar la dirección y el tamaño del efecto de degradado estableciendo los puntos de comienzo y fin, en

caso de que los puntos de fin (*endX, endY*) sean menores a los de inicio (*startX, startY*) el efecto de degradado se invertirá.

- *proportional*. Por lo general está con un valor en *true*, esto indica que los valores para las variables *startX, startY, endX* y *endY* están comprendidas entre 0.0 y 1.0, estos valores representan el porcentaje del degradado. Si por el contrario el valor de *porportional* está en *false*, se deben de indicar los valores del degradado en pixeles.

- *stops*. Representa una secuencia de objetos de tipo *Stop* que componen y definen la distribución de colores para el degradado, *Stop* representa cada uno de los colores aplicados, en cada uno de ellos establecemos el color y el punto donde el degradado tomará ese color en particular mediante la propiedad offset que comprende valores entre 0.0 y 1.0.

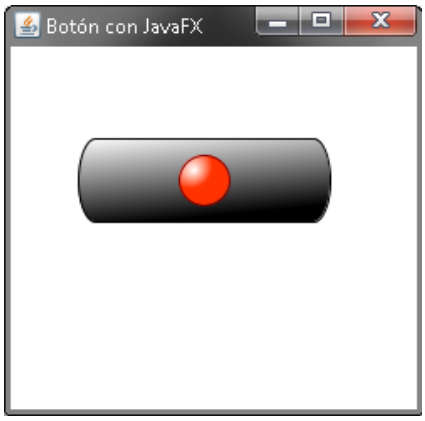
El rectángulo es como el que se muestra a continuación:

Ahora se dibujará un círculo rojo en la parte central del rectángulo utilizando un efecto de degradado circular usando la clase *javafx.scene.paint.RadialGradient*, el código del círculo es el siguiente:

```
var circle = Circle {
    centerX: 115
    centerY: 80
    radius: 15
    fill: RadialGradient {
        centerX: 0.0,
        centerY: 0.0,
        focusX: 0.3,
        focusY: 0.3,
        radius: 1.5,
        proportional: true
        stops: [
            Stop {
                offset: 0.0
                color: Color.WHITE},
            Stop {
                offset: 0.3
                color: Color.web("#ff3300")}
        ]
    } // fill
    stroke: Color.DARKRED
} // Circle
```

Declaramos una variable *circle* asignando un objeto de tipo *Circle* y establecemos sus propiedades de posición y radio. Dentro de la propiedad *fill* declaramos el objeto de tipo *RadialGradient.*

Para establecer la posición del círculo para el degradado radial indicamos valores a las propiedades *centerX* y *centerY*, mediante las propiedades *focusX* y *focusY* indicamos los valores donde el primer color del degradado será mapeado, con *radius* se define el tamaño del radio del círculo usado para el degradado; por último se establecen las demás propiedades *proporcional* y *stops* que se vieron en el ejemplo del rectángulo. La imagen que se genera al pintar el rectángulo y el círculo es la siguiente:



Una vez definidas las figuras que forman el botón, se procederá a agruparlas y aplicar un efecto de reflejo mediante el uso de la clase *javafx.scene.effect.Reflection*, el código para este ejemplo es el siguiente:

```
Stage {
    title: "Botón con JavaFX"
    width: 250
    height: 250
    visible: true
    scene: Scene {
        content: [
            Group {
                content: [
                    rect,
                    circle
                ]
                effect: Reflection {
                    fraction: 0.9
                    topOpacity: 0.5
                    topOffset: 2.5
                }
            }//Group
        ]//content
    }//Scene
}//Stage
```

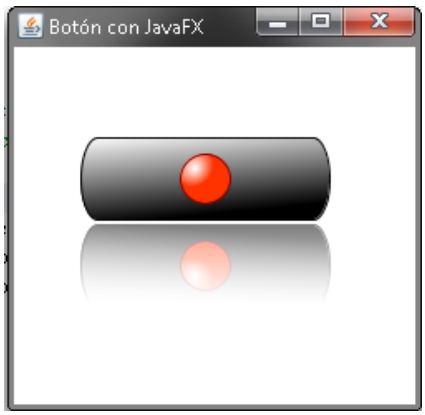
Se declara un objeto de tipo *Group* que contiene a las dos variables de tipo objeto (*rect* y *circle*) necesarias para dibujar el botón, y dentro de la propiedad *effect* declaramos un objeto de tipo *Reflection*. Dentro de la declaración de *Reflection* definimos valores para las siguientes propiedades:

- *fraction*. Indica el porcentaje de visibilidad que tendrá el efecto de reflexión, pudiendo estar en valores comprendidos entre 0.0 y 1.0.

- *topOpacy*. Nos indica el valor que tendrá la opacidad del efecto de reflejo, es decir la intensidad de luz con que se muestra la imagen reflejada.

- *topOffset*. Establece la distancia que debe de existir entre la imagen original y su reflejo.

La imagen final con el efecto de reflejo es como la que se muestra a continuación:



El código completo este ejemplo es el siguiente:

```
package jfx;
import javafx.scene.effect.Reflection;
import javafx.scene.Group;
import javafx.scene.paint.Color;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.RadialGradient;
import javafx.scene.paint.Stop;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

var rect =  Rectangle {
    x: 40            y: 55
    width: 150      height: 50
   arcWidth: 20    arcHeight: 55
   stroke: Color.BLACK
   fill: LinearGradient {
       startX: 0.0,
       startY: 0.0,
       endX: 0.3,
       endY: 1.0,
       proportional: true
       stops: [
           Stop {
               offset: 0.0
               color: Color.WHITE},
           Stop {
               offset: 1.0
               color: Color.BLACK}
       ]
   } // fill
}
```

```
var circle = Circle {
    centerX: 115
    centerY: 80
    radius: 15
    fill: RadialGradient {
        centerX: 0.0,
        centerY: 0.0,
        focusX: 0.3,
        focusY: 0.3,
        radius: 1.5,
        proportional: true
        stops: [
            Stop {
                offset: 0.0
                color: Color.WHITE},
            Stop {
                offset: 0.3
                color: Color.web("#ff3300")}
        ]
    } // fill
    stroke: Color.DARKRED
} // Circle

Stage {
    title: "Botón con JavaFX"
    width: 250
    height: 250
    visible: true
    scene: Scene {
        content: [
            Group {
                content: [
                    rect,
                    circle
                ]
                effect: Reflection {
                    fraction: 0.9
                    topOpacity: 0.5
                    topOffset: 2.5
                }
            }
        ]
    }
}
```

3.5 Enlace de Datos con Objetos Gráficos

JavaFX Script cuenta con un mecanismo de enlace de datos; este mecanismo permite establecer una relación entre dos variables, por lo tanto cuando el valor de una variable cambia afecta directamente el valor de la otra variable relacionada en el enlace.

En el siguiente ejemplo se dibujará un círculo y un objeto de tipo *Slider*, donde se establecerá una relación directa entre el centro del círculo y la posición del indicador del *Slider*.

En primer lugar se creará un objeto del tipo *javafx.scene.control.Slider*, asignando dicho objeto a la variable *slider*.

```
def slider = Slider{min: 0
                    max: 60
                    value: 0
                    translateX: 10
                    translateY: 110};
```

Con las propiedades *max* y *min* se establece el tamaño del desplazamiento para el objeto *Slider*, *value* indica el valor de la posición inicial para el indicador, y las propiedades *translateX* y *translateY* indican la posición del objeto *Slider* dentro de la Escena gráfica.

Ahora se creará un círculo con un efecto de degradado de color, y se asignará a la variable *circulo*.

```
var circulo = Circle {
                centerX: bind slider.value + 50  //enlace con slider
                centerY: 60
                radius: 50
                stroke: Color.BROWN
                fill: RadialGradient {
                    proportional: true
                    stops: [
                        Stop {
                            offset: 0
                            color: Color.CHOCOLATE},
                        Stop {
                            offset: 1
                            color: Color.BEIGE},
                    ]
                }
            }
```

Como se puede observar se establece un enlace entre el valor que este asignado en *slider.value* y el centro del círculo *CenterX*, mediante el uso de la palabra reservada *bind*. De tal manera que cuando se cambie la posición del indicador del slider también se verá afectado de manera automática la posición del eje X del círculo. A continuación se muestra

el código completo de la aplicación, en donde se insertan las variables *slider* y *circulo* dentro de la escena gráfica para poder pintarlas.

```
package jfx;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.scene.paint.Color;
import javafx.scene.paint.Stop;
import javafx.scene.paint.RadialGradient;
import javafx.scene.control.Slider;

def slider = Slider{min: 0
                    max: 60
                    value: 0
                    translateX: 10
                    translateY: 110};

var circulo = Circle {
                centerX: bind slider.value + 50
                centerY: 60
                radius: 50
                stroke: Color.BROWN
                fill: RadialGradient {
                    proportional: true
                    stops: [
                        Stop {
                            offset: 0
                            color: Color.CHOCOLATE},
                        Stop {
                            offset: 1
                            color: Color.BEIGE},
                    ]
                }
            }

Stage {
    title: "Enlace de Datos"
    width: 200
    height: 170
    scene: Scene {
        fill: Color.LIGHTGRAY;
        content: [
            slider,
            circulo
        ]
    }
    visible: true
}
```

Las imágenes resultantes de esta aplicación son las siguientes:

Se puede observar como al desplazar el indicador del *Slider* se mueve de manera automática la posición del eje X del Círculo.

3.6 Uso de Layouts

El uso de Layouts es una opción que provee JavaFX Script para poder distribuir los elementos gráficos; esto sin la necesidad de especificar coordenadas para cada elemento gráfico.

El siguiente ejemplo simula un semáforo, en este ejemplo se hace uso de radio botones para seleccionar el tipo de acción, círculos para simular las luces del semáforo y el uso de elementos de distribución para acomodar y alinear los elementos de forma vertical y horizontal como se muestra en la siguiente figura.

Para construir los radio botones se hace uso de la clase *javafx.scene.control.RadioButton*; para agrupar los radio botones y hacer seleccionable un

solo botón a la vez se hace uso de la clase *javafx.scene.control.ToggleGroup*. El código para generar y agrupar los radio botones es el siguiente.

```
def grupo = ToggleGroup{};
def opciones = ["Alto", "Listo", "Avance"];

def opcionesSemaforo = for (text in opciones){
    RadioButton{
            toggleGroup: grupo
            text: text
    }//RadioButton
}//for
```

Para construir los objetos de tipo *RadioButton* se hace uso de un ciclo *for* y el uso de la secuencia *opciones* para establecer el texto de cada radio botón en la propiedad *text*. En cada iteración los radio botones son asociados a la variable *grupo* del tipo *ToggleGroup*. La variable *opcionesSemaro* es la secuencia que contiene a los radio botones.

Ahora se crearán tres círculos que simulan las luces del semáforo, estos círculos están contenidos en la secuencia *luces*, mediante un ciclo *for* se comienza la construcción de los tres círculos. El código para las luces del semáforo es el siguiente:

```
def colores = ["RED", "GOLD", "GREEN"];
def luces = for (color in colores){
Circle {
    centerX: 12 centerY: 12
    radius: 12  stroke: Color.DARKGRAY
    fill: bind RadialGradient {
        centerX: 8,
        centerY: 8,
        radius: 12,
        proportional: false
        stops: [
            Stop {offset: 0.0 color: Color.WHITE},
            Stop {offset: 1.0 color:
                if (opcionesSemaforo[indexof color].selected)
                      then Color.web(color)
                      else Color.GRAY
            }//Stop
        ]
    }//RadialGradient
}//Circle
}//for
```

Se establecen las propiedades comunes para dibujar los círculos y un efecto de degradado radial para cada círculo. Mediante el uso de *bind* se enlaza el color de degradado con el radio botón que se seleccione, de esta manera el color de círculo cambiará cuando se seleccione su correspondiente radio botón.

Se cuenta con la condición de que cuando este seleccionado un radio botón de la secuencia *opcionesSemaforo*, se pinte el círculo con el color correspondiente de la secuencia *colores*, si los radio botones están deseleccionados la luz del círculo permanecerá en color gris.

Para finalizar se hará uso de las clases *Hbox* y *Vbox* del paquete *javafx.scene.layout* para agrupar y acomodar las luces y radio botones; *Hbox* acomoda las luces de manera horizontal y *Vbox* de manera vertical los radio botones, con el atributo *spacing* se estable la distancia entre cada objeto.

```
Stage {
    title: "Semáforo"
    scene: Scene{
          width: 200  height: 100
        content:
             HBox{ spacing: 20
                content:[
                   VBox{ spacing: 10
                   content: opcionesSemaforo}
                   HBox{ spacing: 15
                   content: luces translateY: 25}
                ]
             }//HBox
     } //Scene
}//Stage
```

En las siguientes imágenes se puede observar el funcionamiento del semáforo.

El código completo de la aplicación es el siguiente:

```
package jfx;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ToggleGroup;
import javafx.scene.shape.Circle;
import javafx.scene.paint.Color;
import javafx.scene.paint.RadialGradient;
import javafx.scene.paint.Stop;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.control.RadioButton;

def grupo = ToggleGroup{};
def opciones = ["Alto", "Listo", "Avance"];
def colores = ["RED", "GOLD", "GREEN"];

def opcionesSemaforo = for (text in opciones){
    RadioButton{
            toggleGroup: grupo
            text: text
    }//RadioButton
}//for
def luces = for (color in colores){
Circle {
    centerX: 12 centerY: 12
    radius: 12  stroke: Color.DARKGRAY
    fill: bind RadialGradient {
        centerX: 8,
        centerY: 8,
        radius: 12,
        proportional: false
        stops: [
            Stop {offset: 0.0 color: Color.WHITE},
            Stop {offset: 1.0 color:
                if (opcionesSemaforo[indexof color].selected)
                      then Color.web(color)
                      else Color.GRAY
            }//Stop
        ]
    }//RadialGradient
}//Circle
}//for
Stage {
    title: "Semáforo"
    scene: Scene{
          width: 200  height: 100
        content:
             HBox{ spacing: 20
                content:[
                   VBox{ spacing: 10
                   content: opcionesSemaforo}
                   HBox{ spacing: 15
                   content: luces translateY: 25}
                ]
```

```
            }//HBox
     } //Scene
}//Stage
```

3.7 Animación de Objetos

Una de las ventajas de JavaFX Script es que provee varias librerías para animación proveniente del paquete *javafx.animation*, estas librerías permiten animar cualquier elemento gráfico.

Se tomará el ejemplo visto en la sección *4.3 Escena Gráfica y Presentación de Objetos*, y se procederá a implementar la animación para el grupo de objetos gráficos; de tal manera que se mostrará un efecto de desplazamiento y rotación como se muestra en la siguiente secuencia de imágenes.

Imagen 1

Imagen 2

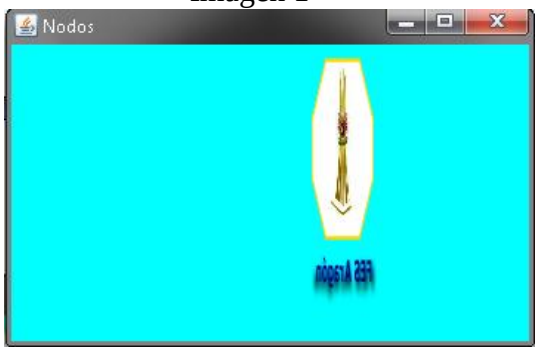


Imagen 3

Imagen 4

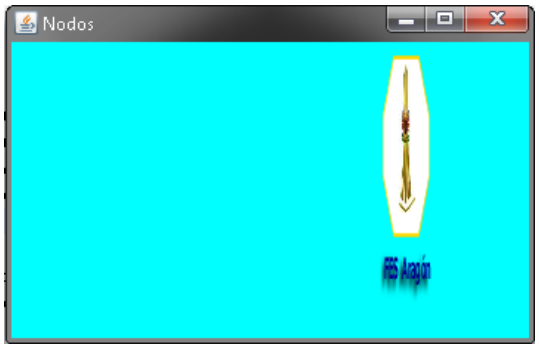


Imagen 5

Imagen 6

El código utilizado para dar tal efecto se muestra a continuación:

```
var scala = 1.0;
var pivoteX = 100;

var tScale: Timeline = Timeline {
    repeatCount: 10
    autoReverse: true
    keyFrames: [
        KeyFrame {
            time: 0s
            values: [ scala => -1.0,  pivoteX => 0]
        },
        KeyFrame {
            time: 5s
            values: [
                scala => 1.0 tween Interpolator.LINEAR,
                pivoteX => 300 tween Interpolator.LINEAR
            ]
        },
    ]
};

Stage {
    title: "Nodos"
    scene: Scene {
        width: 350
        height: 200
        fill: Color.CYAN
        content: Group {
            translateX: 110
            translateY: 10
            content: [
                 ... //Declaración de objetos
           ]//Content
        transforms: bind Transform.scale(scala, 1.0, pivoteX, 10)
        onMouseClicked: function( e: MouseEvent ):Void {
                        tScale.playFromStart();}
        }//Group
    }//Scene
}//Stage
```

En esta animación se hace un enlace de datos entre la función de escala *Transform.scale* dentro de la propiedad *transforms*, y el valor que tomará la variable *scala*; esto aunado al valor de *pivoteX*, definirá el tamaño y posición de la imagen a lo largo de la animación.

El inicio de la animación comienza con el inicio de un evento, en este caso un clic de mouse sobre el grupo de imágenes, el inicio de este evento está definido en el método *onMouseClicked*; este método inicia con la llamada al método *playFromStart()*, que inicia las acciones definidas en la variable de instancia *tScale* que pertenece a la clase *TimeLine*.

La instancia *tScale* representa una línea de tiempo para la ejecución de la animación, cada línea de tiempo se compone de dos o más fotogramas clave denominados como *keyFrame*; estos fotogramas definen el lapso de tiempo y los valores que tomará la imagen para definir su comportamiento. Además se hace uso de otros atributos como: *repeatCount* que indica cuantos ciclos durará la animación y *autoReverse* indica que la animación se ejecutará en sentido inverso cada que termine una iteración.

Las posiciones de inicio y fin de los fotogramas se calculan por medio de una interpolación lineal, así automáticamente el sistema construye la animación a partir de los *keyFrames* definidos. Cada *keyFrame* posee una variable time que toma un valor en segundos *s* o milisegundos *ms*. El operador => indica que habrá una literal para la lista de valores. El operador *tween* es un constructor para el valor de interpolación.

El código completo para la animación es el siguiente:

```
package jfx;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Polygon;
import javafx.scene.paint.Color;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.effect.DropShadow;
import javafx.scene.input.MouseEvent;
import javafx.scene.transform.Transform;
import javafx.animation.Interpolator;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;

var scala = 1.0;
var pivoteX = 100;
var tScale: Timeline = Timeline {
    repeatCount: 10
    autoReverse: true
```

```
    keyFrames: [
        KeyFrame {
            time: 0s    values: [ scala => -1.0,  pivoteX => 0]
        },
        KeyFrame {
            time: 3s
            values: [
                scala => 1.0 tween Interpolator.LINEAR,
                pivoteX => 300 tween Interpolator.LINEAR
           ]
        },
    ]
};
Stage {
    title: "Nodos"
    scene: Scene {
        width:350   height:200
        fill: Color.CYAN
        content: Group {
            translateX:110   translateY:10
            content: [
                Polygon {
                    stroke: Color.GOLD
                    strokeWidth: 2.5
                    fill: Color.WHITE
                    points: [
                         30.0, 0.0,
                         0.0, 40.0,
                         0.0, 80.0,
                         30.0, 120,
                         100.0, 120.0,
                         130.0, 80.0,
                         130.0, 40.0,
                         100.0, 0.0
                    ]
                },
                Text {
                    x:2  y:150
                    effect: DropShadow {
                        offsetX: -5
                        offsetY:  5
                    }
                    content: "FES Aragón"
                    fill: Color.DARKBLUE
                    font: Font.font("Verdana", FontWeight.BOLD, 20);
                },
                ImageView {
                    x:47  y:5
                    image: Image {url: "{__DIR__}torres.jpg"}
                }//ImageView
           ]//Content
            transforms: bind Transform.scale(scala, 1.0, pivoteX, 10)
            onMouseClicked: function( e: MouseEvent ):Void {
            tScale.playFromStart();}
        }//Group
    }//Scene
}//Stage
```

3.8 Gráficas

Una de las utilidades que provee JavaFX Script es de que ofrece la librería *javafx.scene.chart*, esta librería ofrece una variedad de gráficas como son: de barras, en forma de pay, lineales, de burbujas y algunas en 3D. Estas gráficas son fáciles de implementar y personalizar. Se mostrará algunos ejemplos a continuación.

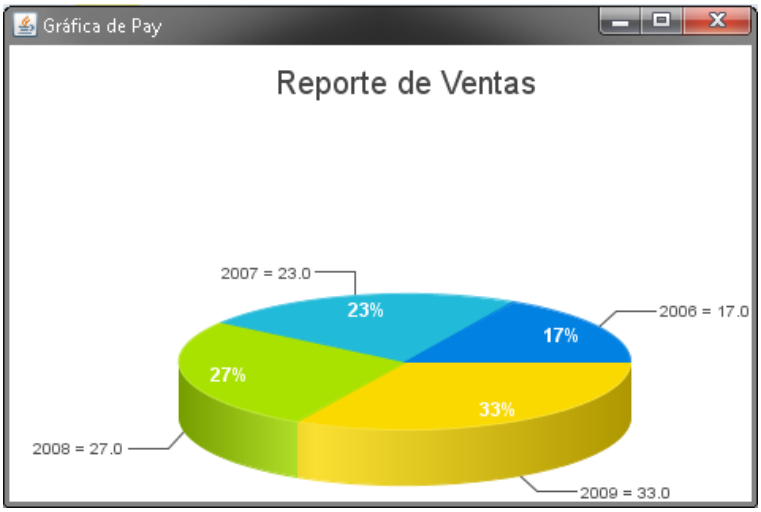
El siguiente código genera una grafica en forma de pay 3D, en primer lugar se establecen propiedades comunes como el titulo y la fuente; la propiedad data representa la secuencia de elementos que alimentarán de datos a la gráfica, estos elementos son *PieChart3D.Data* teniendo cada uno de ellos un valor propio y una etiqueta.

```
package jfx;

import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.chart.PieChart3D;

Stage {
  title: "Gráfica de Pay"
  scene: Scene {
    content: [
      PieChart3D {
        title: "Reporte de Ventas"
        titleFont: Font { size: 20 }
        data: [
          PieChart3D.Data {
            value: 33
            label: "2009"
          }
          PieChart3D.Data {
            value: 27
            label: "2008"
          }
          PieChart3D.Data {
            value: 23
            label: "2007"
          }
          PieChart3D.Data {
            value: 17
            label: "2006"
          }
        ]
      }
    ]//content
  }//scene
}//stage
```

La gráfica generada tras ejecutar el código anterior es como la que se muestra a continuación.



En el siguiente capítulo se desarrollará una aplicación enriquecida gráficamente, a través de dicha aplicación se tratarán de aplicar varios de los temas vistos en este capítulo y el anterior básicamente.

CAPÍTULO 5
DESARROLLO DE UNA APLICACIÓN TIPO RIA CON JAVAFX

5.1 Introducción

En este capítulo se llevará el cabo el desarrollo de una aplicación tipo RIA, durante el desarrollo de esta aplicación se tratarán de aplicar algunos de los temas vistos en los capítulos anteriores como son el manejo de imágenes, efectos visuales, enlace de datos, eventos de mouse sobre nodos, uso de clases, tipos de datos y funciones; por lo tanto se está hablando de una aplicación orientada a objetos.

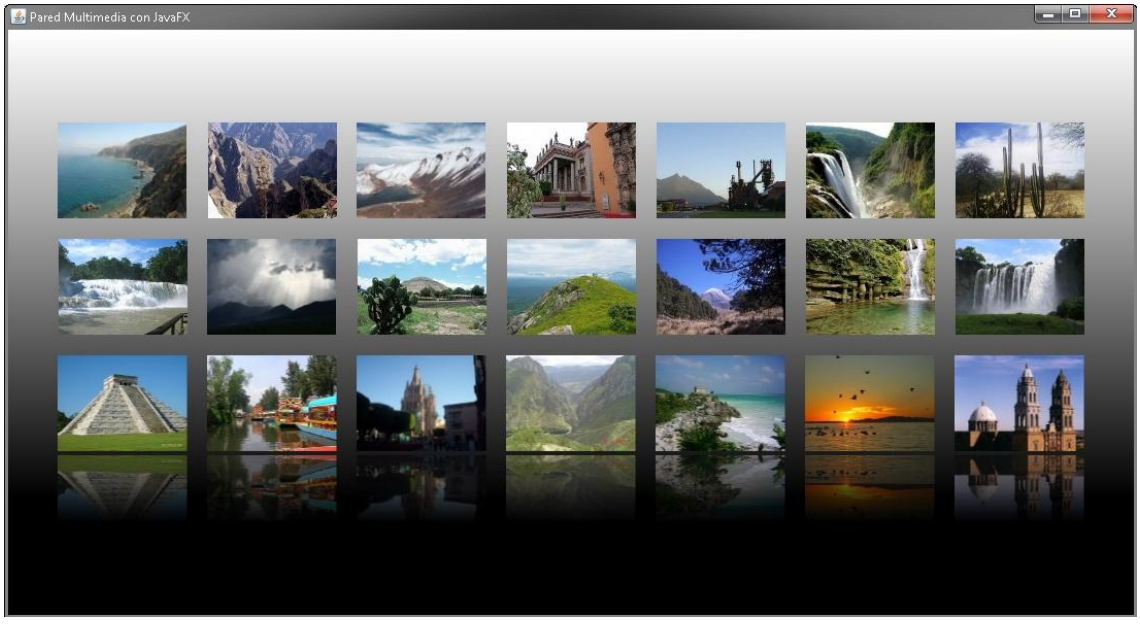
5.2 Funcionamiento de la aplicación y componentes

La aplicación que se desarrollará representa una pared gráfica de varios elementos, a través de esta pared podemos visualizar diferentes imágenes, cada imagen representa un nodo. Los nodos son agrupados y se acomodan de forma dinámica y automática para su presentación.

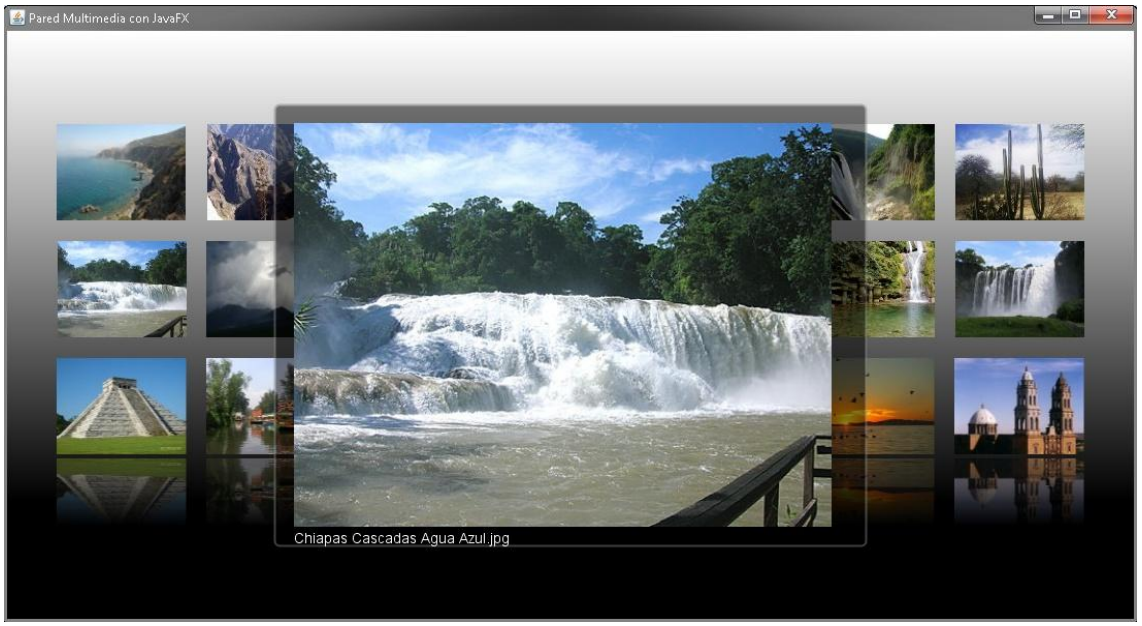
Cada una de las imágenes presentadas representa un objeto que es susceptible a los eventos que puede generar el usuario mediante el uso del mouse; cuando se hace clic en alguno de estos elementos, la imagen es presentada en el centro de la ventana y es escalada en un porcentaje relativo y acorde al tamaño de la ventana.

La pared tiene como fondo un arreglo de colores con efecto de degradado, y las imágenes que forman la tercer fila de nodos tienen aplicado un efecto de reflejo.

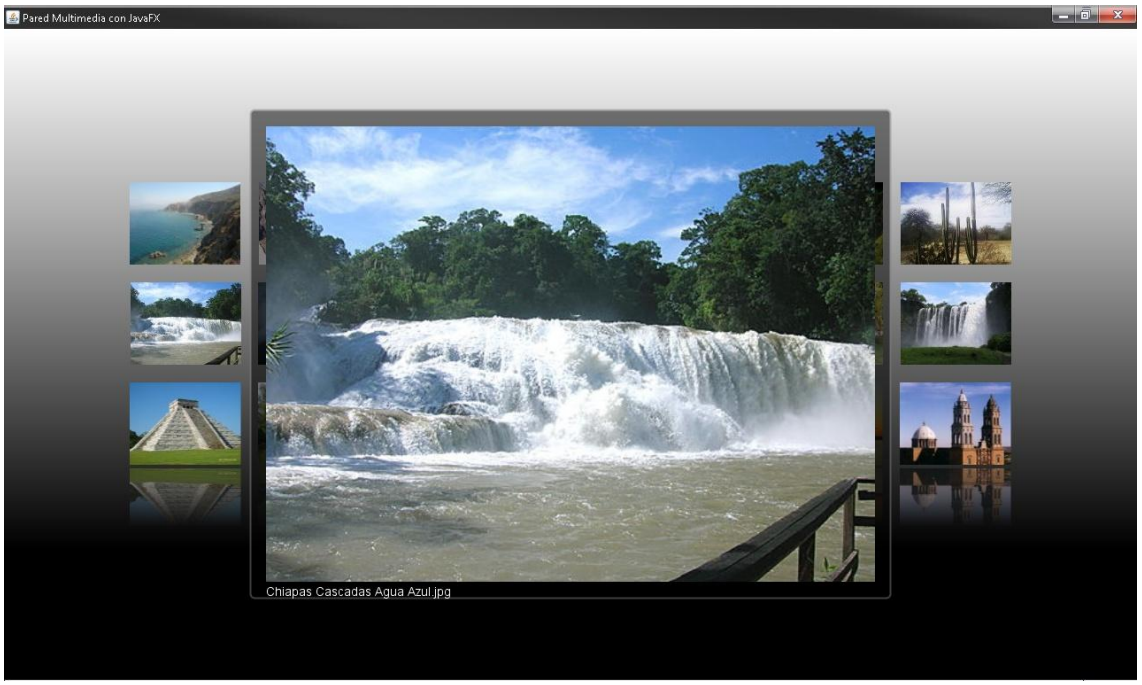
A continuación se muestra una captura de pantalla de la aplicación en ejecución.



Una vez haciendo clic sobre alguna de las imágenes, se muestra la imagen escalada a un porcentaje equivalente al 75% del tamaño de la total de la ventana. Por default la ventana cuenta con un tamaño de 1100x600 pixeles.



Si se maximiza el tamaño de la ventana los elementos que forman la pared son reacomodados al centro de la ventana, y la imagen escalada cambia de tamaño de manera automática, adaptándose al nuevo tamaño de la ventana.



La aplicación está organizada en cuatro paquetes:

- *jfx*. Contiene el código fuente para el funcionamiento de la aplicación
- *jfx.imagenes*. Contiene las imágenes usadas para el efecto de escalar las imágenes que forman la pared.
- *jfx.miniaturas*. Contiene las imágenes pequeñas que forman la pared gráfica.
- *jfx.recursos*. Contiene imágenes que se mostrarán como marco alrededor de las imágenes una vez escaladas de tamaño.

Los archivos de código fuente JavaFX, los archivos son los siguientes:

- *Main.fx*. Este es el archivo principal de la aplicación donde se encuentra la definición de los elementos *Stage* y *Scene* de la ventana. El contenido de la escena gráfica está formado por una secuencia de imágenes miniaturas representadas por la clase *Pared*.
- *Constantes.fx*. Este archivo contiene la definición de diferentes constantes usadas por la aplicación, estas contantes son usadas para parametrizar valores como tamaños, espacios y colores de la ventana e imágenes. Al tener estos valores parámetrizados se hace más sencilla la labor de poder modificar o adaptar el código de la aplicación.
- *ModeloDatos.fx*. Esta clase define el origen y ruta de cada una de las imágenes, una secuencia de todos estos objetos son usados para llenar la pared gráfica.
- *ImagenMiniatura.fx*. Es la Clase que representa cada una de las imágenes pequeñas que forman parte de la pared, y contiene los eventos de mouse que pueden ser aplicados sobre cada objeto.
- *Pared.fx*. Esta clase tiene como función desplegar las imágenes miniatura en forma ordenada a través de la ventana
- *Foto.fx*. Esta clase representa una instancia de la imagen que es escalada a partir de la imagen pequeña seleccionada en la pared gráfica, incluye código para el cálculo del espacio entre el marco y la imagen mostrada, así como el texto correspondiente de cada imagen.

Las Clases *ImagenMiniatura* y *Foto* heredan las propiedades de la clase *javafx.scene.Node* sobrescribiendo la función *create()*; estas subclases son conocidas como nodos personalizados por sobrescribir la definición de esta función.

5.3 Descripción de Main.fx

En este archivo es donde el elemento *Stage* es creado, el contenido de *Scene* está formado por una instancia de la clase *Pared* estableciendo sus propiedades y funciones. Las propiedades como el tamaño de la ventana, color de fondo están definidas en el archivo *Constantes.fx*.

Se hace uso de la clase *ModeloDatos* para obtener las direcciones de cada una de las imágenes que se mostrarán en la pared. Cuando el usuario hace clic sobre alguna de las imágenes presentadas se crea una instancia de la clase Foto para presentar la imagen escalada en el centro de la escena gráfica. A continuación se muestra el código de *Main.fx*.

```
package jfx;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.image.Image;

var stage : Stage;
var pared : Pared;
var foto : Foto;

function muestraFoto(modeloDatos : ModeloDatos, marco : Image) : Void
{
    ocultaFoto();
    foto = Foto {
        modeloDatos: modeloDatos
        marcoImagen: marco
        limiteAlto: bind stage.scene.height * .75
        limiteAncho: bind stage.scene.width * .75
        ocultar: ocultaFoto
        translateX: bind (stage.scene.width - foto.width)/2
        translateY: bind (stage.scene.height - foto.height)/2
    }
    insert foto into stage.scene.content;
}

function ocultaFoto() : Void {
    delete foto from stage.scene.content;
    foto = null;
}

bound function xoffset() : Number {
    if ( pared.boundsInLocal.width < (stage.scene.width -
Constantes.MINIATURA_ESPACIO_HORIZONTAL) ) {
        (stage.scene.width - pared.boundsInLocal.width) / 2
    } else {
        Constantes.MINIATURA_ESPACIO_HORIZONTAL
    }
}

stage = Stage {
    title: "Pared Multimedia con JavaFX"
    height: Constantes.ESTAGE_ALTO
    width : Constantes.ESTAGE_ANCHO
    scene : Scene {
        fill: Constantes.ESTAGE_COLOR_FONDO
        content: [
            pared = Pared {
                modeloDatos: ModeloDatos.coleccionImagenes
                vistaCompleta: muestraFoto
                translateX: bind xoffset()
                translateY: bind (stage.scene.height -
                                  pared.boundsInLocal.height) / 2
            }
        ]//Content
    }//Scene
 }//Stage
```

Se declaran las variables de instancia *stage*, *pared* y *foto* para poder manipularlas e inicializar la aplicación.

Al momento de inicializar la variable *pared* en la escena gráfica se establece el modelo de datos para obtener las referencias de cada una de las imágenes, y se establece el atributo *vistaCompleta* con la función *muestraFoto()*; por último se establecen las propiedades *translateX* y *translateY* que están enlazadas con la función *xoffset()*para la primer propiedad y para la segunda se enlaza con un cálculo sobre el alto de la ventana.

La función *muestraFoto* puede ser llamada desde un objeto del tipo *ImagenMiniatura,* que contiene definidas las acciones que realizarán en caso de dar clic en el mismo objeto. Cuando esto sucede se crea una instancia de la clase *Foto* y es insertada dentro de la escena gráfica de la ventana.

En las propiedades para el objeto *Foto* se establece el tamaño para la imagen con las propiedades *limiteAlto* y *limiteAncho* , estos valores esta enlazados con el tamaño de la ventana en una relación del 75%; del mismo modo la posición de la imagen está definida por las propiedades *translateX* y *translateY* , la posición es calculada de forma dinámica con la diferencia de tamaños entre la ventana y la imagen a mostrar.

La imagen a escalar se pasa como parámetro a la función *muestraFoto()* y se establece en la propiedad *marco* del objeto *Foto*, también se pasa como parámetro el modelo de datos para obtener la ruta de la imagen.

La función *ocultaFoto()* se ejecuta cada vez que se crea un objeto de tipo Foto o cuando se hace clic sobre la misma imagen presentada en la escena gráfica para quitarla, en caso de que ya exista una imagen se elimina el objeto de la escena gráfica, para posteriormente poder insertar otra nueva imagen en la escena.

La función *xoffset(),* es una función enlazada con la propiedad *translateX* del objeto *pared*; la razón de ser de esta función es la de calcular de forma automática la posición del

eje X para el objeto pared, ya que esta posición depende del tamaño de la ventana y de el número de imágenes contenidas en la pared.

5.4 Descripción de Constantes.fx

En este archivo se establecen los valores que tomara la aplicación para establecer el color de fondo, tamaño de la ventana y las imágenes pequeñas que forman la pared, así como el espacio entre ellas. Las constantes están declaradas con el modificador de acceso *package*, para que estén disponibles para todos los archivos y clases dentro del paquete *jfx*.

A continuación se muestra el código para la declaración de constantes de la aplicación

```
package jfx;
import javafx.scene.paint.Color;
import javafx.scene.image.Image;
import javafx.scene.paint.LinearGradient;
import javafx.scene.paint.Stop;

/** Alto de imagen miniatura */
package def MINIATURA_ALTO = 93;
/** Ancho de la imagen miniatura */
package def MINIATURA_ANCHO  = 125;
/** Espacio vertical entre cada imagen miniatura */
package def MINIATURA_ESPACIO_VERTICAL= 20;
/** Espacio horizontal entre cada imagen miniatura */
package def MINIATURA_ESPACIO_HORIZONTAL= 20;
/** Número de filas que serán desplegadas en la pared gráfica */
package def MINIATURA_FILAS = 3;
/** Alto de la Ventana */
package def ESTAGE_ALTO = 600;
/** Ancho de la ventana */
package def ESTAGE_ANCHO  = 1100;
```

```
/** Color de fondo para la ventana, efecto de degradado */
package def ESTAGE_COLOR_FONDO = LinearGradient {
                                startX: 0.0,
                                startY: 0.0,
                                endX: 0.0,
                                endY: 1.0,
                                proportional: true
                                stops: [
                                    Stop {
                                        offset: 0.0
                                        color: Color.WHITE},
                                    Stop {
                                        offset: 0.3
                                        color: Color.DARKGREY},
                                    Stop {
                                        offset: 0.8
                                        color: Color.BLACK}
                                ]
                                }

/** Imagen que servirá como marco al mostrar una imagen escalada */
package def FOTO_MARCO : Image = Image {
    url: "{__DIR__}recursos/photoIcon.png"
}

/** Determina la ruta donde se encuentran los recursos */
package def CODEBASE : String = {
    var codebase = FX.getProperty("javafx.application.codebase");

    if ( codebase == "" or codebase.startsWith("file:") ) {
        codebase =   DIR
    }
    codebase
}
```

La constante *CODEBASE* representa la ruta para poder ubicar el origen de las imágenes tanto para la pared gráfica como para las imágenes expandidas. Para poder obtener la ruta en caso de desplegar la aplicación en un navegador web la ruta se obtiene mediante la línea de código: *FX.getProperty("javafx.application.codebase").* Esta línea obtiene la ruta base de la aplicación mediante el método *getProperty* de la clase FX. Si se devuelve un nulo se toma la ruta física mediante el uso de *_DIR_*.

5.5 Descripción de ModeloDatos.fx

En este archivo esta defina una clase llamada ModeloDatos y tiene por atributos el nombre del archivo, la ruta de la imagen a mostrar y la ruta de la imagen miniatura. El nombre de las imágenes miniatura y el de las imágenes de tamaño real es el mismo, solo que se encuentran en diferentes paquetes.

La secuencia *archivosImagen* contiene los nombres para las imágenes que se utilizarán para la aplicación. Por último se construye una secuencia mediante el uso de un ciclo *for*, la secuencia llamada *coleccionImagenes* formada por objetos del tipo *ModeloDatos*; cada uno de estos contiene el nombre y las rutas para las imágenes que utilizará la aplicación.

```
package jfx;

package class ModeloDatos {
    package var imagenMiniatura_url : String;
    package var media_url : String;
    package var nombreArchivo  : String;
}

def base_url : String = Constantes.CODEBASE;

def archivosImagen = [
    "Baja California Ensenada.jpg",
    "Chiapas Cascadas Agua Azul.jpg",
    "Chiapas Chichen Itza.jpg",
    "Chihuahua La Sinforosa.jpg",
    "Coahuila Arteaga.jpg",
    "DF Xochimilco.jpg",
    "Edo. Mexico Nevado Toluca.jpg",
    "Edo. Mexico Teotihuacan.jpg",
    "Guanajuato San Miguel de Allende.jpg",
    "Guanajuato Tetro Juarez.jpg",
    "Guerrero Ometepec.jpg",
    "Hidalgo Tolantongo.jpg",
    "Monterrey Fundidora.jpg",
    "Puebla Pico de Orizaba.jpg",
    "Quintana Roo.jpg",
    "San Luis Potosi Huasteca.jpg",
```

```
    "San Luis Potosi Xilitla.jpg",
    "Sonora Bahia Kino.jpg",
    "Tamaulipas CD. Tula.jpg",
    "Veracruz San Andres Tuxtla.jpg",
    "Zacatecas Chalchihuites.jpg"
];

package var coleccionImagenes = for (nombre in archivosImagen) {
    ModeloDatos {
        imagenMiniatura_url : "{base_url}miniaturas/{nombre}"
        media_url : "{base_url}imagenes/{nombre}"
        nombreArchivo  : nombre
    }
}
```

5.6 Descripción de ImagenMiniatura.fx

Esta clase encapsula la funcionalidad de cada una de las pequeñas imágenes que estarán agrupadas y presentadas por la pared gráfica. El código para esta clase es el siguiente, después se explicará el detalle de la funcionalidad de este código.

```
package jfx;
import javafx.scene.CustomNode;
import javafx.scene.Node;
import javafx.scene.input.MouseEvent;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.effect.Effect;
import javafx.scene.effect.Reflection;

def efectoReflejo : Effect = Reflection {
    fraction: 0.7
    topOpacity: 0.6
    bottomOpacity: 0
    topOffset: 4
}

package class ImagenMiniatura extends CustomNode {
   package var modeloDatos : ModeloDatos;
   package var reflejo : Boolean = false;
   package var vistaCompleta : function(metaData : ModeloDatos,
                                placeholder : Image) : Void;

    def imagen : Image = Image {
        url: modeloDatos.imagenMiniatura_url
        placeholder: Constantes.FOTO_MARCO
           backgroundLoading: true
    }
```

```
    override function create() : Node {
        ImageView {
            effect: if ( reflejo ) then efectoReflejo else null
            fitWidth: Constantes.MINIATURA_ANCHO
            fitHeight: Constantes.MINIATURA_ALTO
            image: imagen
            onMouseClicked: function(evt : MouseEvent) : Void {
                vistaCompleta(modeloDatos, imagen);
            }
        }//ImageView
    }//Node
}//Class
```

Primero se declara la variable *efectoReflejo* que contiene el efecto de reflejo que será aplicado a ciertos objetos del tipo *ImagenMiniatura*.

Dentro de la definición de la clase *ModeloDatos* se establecen las siguientes propiedades:

- *modeloDatos*. Es la ruta del archivo que será modelado, la imagen pequeña es la vista de este dato.
- *reflejo*. Bandera que indica si el efecto de reflejo será aplicado o no.
- *vistaCompleta.* Indica la función que se aplicará para expandir el tamaño de la imagen. La implementación de la función se lleva a cabo por la función *muestraFoto* en el archivo *Main.fx.*
- *Imagen*. Es la referencia al objeto de tipo *Image*, esta imagen es pasada a la función *vistaCompleta* para poder mostrarla en su tamaño normal.

Se ha sobrescrito el método *create()* para la definición de esta clase, dentro de la función se declaran las propiedades para crear un objeto del tipo *ImageView* que es el encargado de desplegar la imagen.

Dentro de *ImageView* se establece el tamaño del objeto que contendrá a la imagen, el efecto de reflejo es aplicado en caso de que la bandera reflejo tenga un valor true, se establece la imagen que será desplegada; por último se activa el evento de mouse *onMouseClicked,* cuando se de clic en cualquier objeto de tipo *ImagenMiniatura* se activará

la función *vistaCompleta()* para mostrar la imagen expandida, esta función es implementada será implementada en *Main.fx*.

5.7 Descripción de Pared.fx

Esta clase tiene como función desplegar las imágenes miniatura en forma ordenada a través de la ventana. Las referencia para cada imagen pequeña está dada por la secuencia *ModeloDatos[]*que se definió en *ModeloDatos.fx*, A continuación se muestra el código de la clase *Pared.fx*.

```
 package jfx;
import javafx.scene.CustomNode;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.image.Image;

public class Pared extends CustomNode {

    package var vistaCompleta : function(metaData : ModeloDatos,
placeholder : Image) : Void;

    package var modeloDatos : ModeloDatos[];

     def imagenMiniaturas : ImagenMiniatura[] = bind for (dato in
      modeloDatos) {

        var col = (indexof dato) / Constantes.MINIATURA_FILAS;
        var row = (indexof dato) mod Constantes.MINIATURA_FILAS;

        ImagenMiniatura {
            modeloDatos: dato
            vistaCompleta: vistaCompleta
            translateX: (col * Constantes.MINIATURA_ANCHO) +
                   (col * Constantes.MINIATURA_ESPACIO_HORIZONTAL)
            translateY: (row * Constantes.MINIATURA_ALTO) +
                   (row * Constantes.MINIATURA_ESPACIO_VERTICAL)
            reflejo: (row+1) == Constantes.MINIATURA_FILAS
        }
    }

    override function create() : Node {
        Group {
            content: imagenMiniaturas
        }
    }
}
```

Para los atributos de esta clase se declara la función *vistaCompleta()* que está asociada a la función del mismo nombre para cada una de las imágenes miniatura, se establece el modelo de datos para obtener la referencia para cada imagen, y se declara una secuencia del tipo *ImagenMiniatura[]*.

Mediante el uso de un ciclo *for* se va estableciendo el número de columna (*col)* y fila (*row)* para cada imagen; esto se calcula usando el índice que ocupa la referencia de cada imagen en el modelo de datos, y el número de filas establecidas en las contantes.

Al crear la instancia de cada imagen se establecen las siguientes propiedades:

- *modeloDatos*. Establece la ubicación de la imagen.
- *vistaCompleta*. Declara la función que se activará con el evento clic del mouse.
- *translateX*. Posición de la imagen en el eje X, se calcula por medio del número de columna, el ancho de la imagen miniatura y el espacio horizontal entre cada imagen.
- *translateY*. Posición de la imagen en el eje Y, se calcula por medio del numero de fila, el alto de la imagen miniatura y el espacio vertical entre cada imagen.
- *reflejo*. Es una expresión booleana que determina si el efecto de reflejo es aplicado, comparando el valor de la fila con el número de filas definido en las constantes.

Por último se sobrescribe el método *create(),* lo que se hace aquí es solo agrupar la secuencia *imagenMiniaturas*, esta secuencia contiene todas instancias de la clase *ImagenMiniatura*.

5.8 Descripción de Foto.fx

Esta clase define las propiedades y comportamientos que debe de tener una imagen cuando es expandida, los objetos de este tipo de clase cuentan con funciones para calcular de forma automática el tamaño de la imagen, esto en caso que la ventana principal cambia de tamaño; también esta clase cuenta con el evento *onMouseClicked* para ocultar la imagen si se hace clic sobre ella. A continuación se muestra el código de la clase *Foto.fx*.

```
package jfx;
import javafx.scene.CustomNode;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.input.MouseEvent;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.shape.Rectangle;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.layout.Resizable;

package class Foto extends CustomNode, Resizable {

    package var modeloDatos : ModeloDatos;
    package var limiteAncho  : Number on replace {
        cambiarTamanio();
    }
    package var limiteAlto : Number on replace {
        cambiarTamanio();
    }
    package var marcoImagen : Image;
    def imagen : Image = Image {
            url: modeloDatos.media_url
            backgroundLoading: true
            placeholder: marcoImagen
        } on replace {
            cambiarTamanio();
        }
    def espacio : Number = 18;
    package var ocultar : function() : Void;

    bound function imagenLimiteAlto() : Number {
        limiteAlto - texto.boundsInLocal.height - 3 * espacio
    }

    bound function imagenLimiteAncho() : Number {
        limiteAncho - 2 * espacio
    }

    def imageView : ImageView = ImageView {
        blocksMouse: true
        fitHeight: bind height - (2 * espacio)
        fitWidth : bind width - (2 * espacio)

        preserveRatio: true
        translateX: espacio
        translateY: espacio

        image: imagen
        onMouseClicked: function(evt : MouseEvent) : Void {
            imageView.image.cancel();
            ocultar();
        }
    }
```

```
    def texto : Text = Text {
        font : Font {
            size : 14
        }

        fill: Color.WHITE
        translateX: espacio
        translateY: bind imageView.boundsInLocal.maxY +
                            texto.boundsInLocal.height + espacio

        content: nombreBase(modeloDatos.nombreArchivo)
        clip: Rectangle {
            translateY: bind -texto.font.size
            width: bind width
            height: bind texto.font.size + texto.font.size / 4
            smooth: false
        }
    }

    function nombreBase(path : String) : String {
        var basename : String = null;

        if ( path != null ) {
            var offset : Integer = path.lastIndexOf('/') + 1;
            basename = path.substring(offset);
        }

        return basename;
    }

    def frame : Rectangle = Rectangle {
        width: bind width
        height: bind height
        arcWidth: 10
        arcHeight: 10
        fill: Color.BLACK
        stroke: Color.GRAY
        strokeWidth: 2
        opacity: 0.5
    }

    override function create(): Node {
        return Group {
            content: [
                frame,
                imageView,
                texto
            ]
        };
    }
```

```
override function getPrefWidth(h) {
    var imageWidth: Number = 0.0;
    var imageHeight: Number = 0.0;
    var fitH  = h;

    if (imagen == null or imagen.width==0 or imagen.height==0) {
        if (marcoImagen != null) {
            imageWidth = marcoImagen.width;
            imageHeight = marcoImagen.height;
        }
    } else {
        imageWidth = imagen.width;
        imageHeight = imagen.height;
    }
    if (imagen!= null and imageHeight > 0 and imageWidth > 0) {
        if (fitH > 2*espacio) {
            return (fitH -( 2*espacio)) *(imageWidth/imageHeight)
                        + 2*espacio
        } else {
            return imageWidth + (2*espacio);
        }
    } else {
        if (fitH > (2*espacio)) {
            return (fitH-(2*espacio))*3/2 + (2*espacio);
         } else {
             return 360 + (2*espacio);
         }
    }
 }//function
override function getPrefHeight(w) {
    var imageWidth: Number = 0.0;
    var imageHeight: Number = 0.0;

    if (imagen == null or imagen.width==0 or imagen.height==0) {
        if (marcoImagen != null) {
            imageWidth = marcoImagen.width;
            imageHeight = marcoImagen.height;
        }
    } else {
        imageWidth = imagen.width;
        imageHeight = imagen.height;
    }
    if (imagen != null and imageHeight > 0 and imageHeight > 0) {
        if (w >(2*espacio)) {
            return (w-(2*espacio)) *(imageHeight/imageWidth) +
                        (2*espacio)
        } else {
            return imageHeight  + (2*espacio);
        }
    } else {
        if (w > (2*espacio) ) {
            return (w-(2*espacio))*2/3;
        } else {
            return 240 + (2*espacio);
        }
    }
}//function
```

```
    package function cambiarTamanio():Void {
        if (getPrefHeight(limiteAncho) > limiteAlto){
            if (getPrefWidth(limiteAlto) > limiteAncho) {
                width = limiteAncho;
                height = limiteAlto;
            } else {
               height = limiteAlto;
               width = getPrefWidth(limiteAlto);
            }
        } else {
           if (getPrefHeight(limiteAncho) > limiteAncho) {
               width = limiteAncho;
              height = limiteAlto;
            } else {
               width = limiteAncho;
               height = getPrefHeight(limiteAncho);
            }
       }
    }//function
}//class
```

Esta clase sobrescribe la función *create()*, es aquí donde se agrupan los tres nodos que se definen por medio de esta clase, estos elementos son:

- *frame*. Representa la figura que servirá de fondo y marco para la imagen que se muestra en el centro de la ventana. Esta figura es un objeto del tipo *Rectangle*, el tamaño de esta figura esta enlazado directamente con el tamaño de la ventana que está definido en *Main.fx*.
- *imageView*. Es el elemento que funciona como contenedor para la imagen que se va mostrar, esta imagen está representada por la variable *imagen*, otras de las propiedades para este objeto son:
    - *blocksMouse*. Esta propiedad permite captar los eventos de mouse solo sobre la imagen expandida y que no se liguen con los de las imágenes pequeñas que están detrás de la imagen presentada en el centro.
    - *fitHeight* y *fitWidth*. Estas propiedades establecen el tamaño de la imagen y son las responsables de escalar el tamaño de la imagen de acuerdo al tamaño de la ventana.
    - *translateX* y *translateY*. Establecen el valor de la posición relativa de la imagen en relación del valor de la variable *espacio*, este espacio es el

que se encuentra entre el elemento *frame* y la *imagen* que se mostrará, esto da el efecto de marco a la imagen y deja espacio para colocar el texto con el nombre de la imagen.

  - *onMouseClicked().* Este evento indica que se ejecutará la función ocultar cada que se haga clic sobre la imagen presentada. Este método se implementa en *Main.fx*.

- *texto*. Contiene un objeto del tipo *Text*, en este objeto establecemos la fuente, el color de la fuente en blanco; en la propiedad *translateY* se establece la posición que tendrá el texto en relación del límite del objeto *imageView*, así el texto aparecerá justo debajo de la imagen presentada. El nombre de la imagen se obtiene del modelo de datos; por último se estable la propiedad *clip* en donde se establece un rectángulo que cambia de posición y tamaño de acuerdo a las propiedades de imagen presentada.

Algunas de los atributos de la clase *Foto* tienen asociados la función *cambiarTamanio()* como es el caso de los atributos *limiteAlto*, *limiteAncho* e *imagen*; esta función es ejecutada cada vez que los valores del tamaño de la ventana cambian o que la referencia de la imagen cambia.

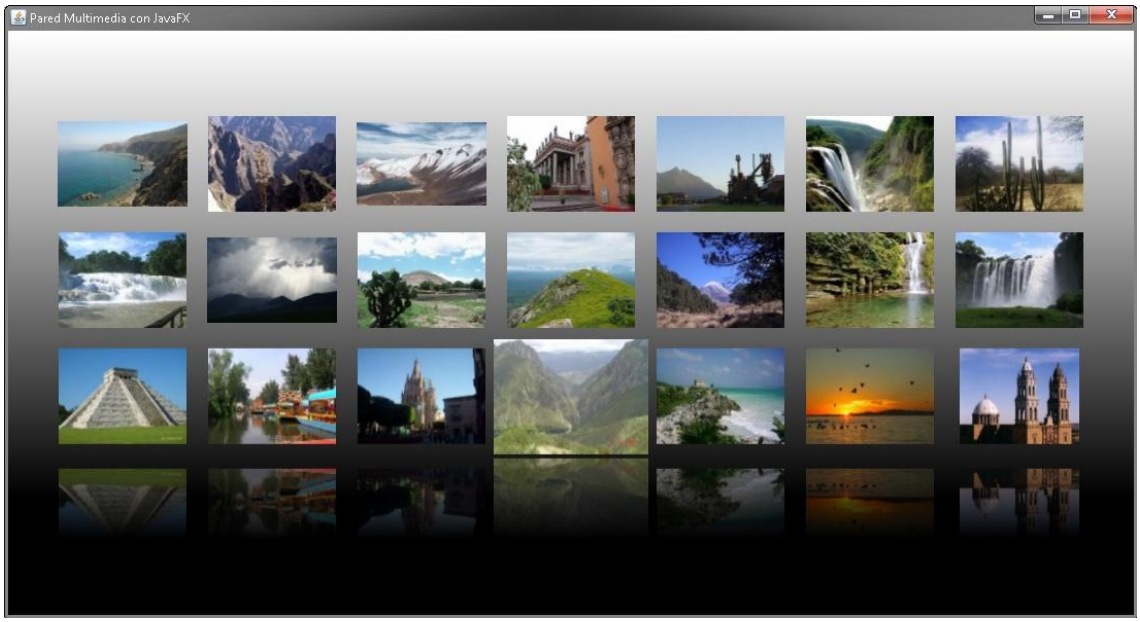
La función *cambiarTamanio* calcula y actualiza los límites de ancho y alto para la imagen, hace uso de otras dos funciones para poder hacer los cálculos: *getPrefWidth* y *getPrefHeight*,

5.9 Efecto de Expansión para ImagenMiniatura

Como último tema se modificará y añadirá a la aplicación un efecto de expansión para cada una de las instancias de las pequeñas imágenes que conforman la pared gráfica.

Para llevar a cabo el efecto de expansión de las imágenes pequeñas se modificará la definición de clase *ImagenMiniatura* y se agregaran dos eventos de mouse: *onMouseEntered* y *onMouseExited*; el primer evento para aumentar en un 20% el tamaño

de la imagen cuando el mouse pase sobre ella, y el segundo evento regresar la imagen a su tamaño original cuando el mouse esta fuera del área que abarca la imagen. En la siguiente captura de pantalla se puede apreciar como la cuarta imagen de la tercera fila, es aumentada de tamaño cuando el puntero del mouse esta posicionada sobre ella y regresa a su tamaño original cuando ya no se está apuntando a ella, lo mismo sucede con cada una de las imágenes que forman la pared gráfica.



A continuación se muestra el nuevo código para la clase *ImagenMiniatura*

```
package jfx;
import javafx.scene.CustomNode;
import javafx.scene.Node;
import javafx.scene.input.MouseEvent;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.effect.Effect;
import javafx.scene.effect.Reflection;
import javafx.scene.shape.Rectangle;
import javafx.scene.Group;


def efectoReflejo : Effect = Reflection {
    fraction: 0.7
    topOpacity: 0.6
    bottomOpacity: 0
    topOffset: 4
}

package class ImagenMiniatura extends CustomNode {
   package var modeloDatos : ModeloDatos;
   package var reflejo : Boolean = false;
```

```
    package var vistaCompleta : function(metaData : ModeloDatos,
                          placeholder : Image) : Void;
    def imagen : Image = Image {
        url: modeloDatos.imagenMiniatura_url
        placeholder: Constantes.FOTO_MARCO
           backgroundLoading: true
        width: Constantes.MINIATURA_ANCHO *
                          Constantes.MINIATURA_EXPANDIDA
        height: Constantes.MINIATURA_ALTO *
                          Constantes.MINIATURA_EXPANDIDA
        preserveRatio: true
    }

    def rectanguloLimite = Rectangle {
        width: Constantes.MINIATURA_ANCHO *
                          Constantes.MINIATURA_EXPANDIDA
        height: Constantes.MINIATURA_ALTO *
                          Constantes.MINIATURA_EXPANDIDA
        visible: true
        opacity:0
    }

    def imagenVista: ImageView = ImageView {

        image: imagen
        translateX: bind (rectanguloLimite.boundsInLocal.width -
                          imagenVista.boundsInLocal.width) / 2.0
        translateY: bind (rectanguloLimite.boundsInLocal.height -
                          imagenVista.boundsInLocal.height) / 2.0
        fitWidth: Constantes.MINIATURA ANCHO
        fitHeight: Constantes.MINIATURA_ALTO
        preserveRatio: true
        blocksMouse: true

        onMouseClicked: function(evt: MouseEvent) : Void {
            vistaCompleta(modeloDatos, imagen);
        }
        onMouseEntered: function(evt: MouseEvent) : Void {
            imagenVista.scaleX = Constantes.MINIATURA_EXPANDIDA;
            imagenVista.scaleY = Constantes.MINIATURA_EXPANDIDA;
        }
        onMouseExited: function(evt: MouseEvent) : Void {
            imagenVista.scaleX = 1;
            imagenVista.scaleY = 1;
        }
    }
    override function create() : Node {
        Group {
            effect: if (reflejo) then efectoReflejo else null;
            content: [
                rectanguloLimite,
                imagenVista
            ]
        }
    }
}
```

Se ha modificado la función *create()*, ahora se agrupan dos nodos: *rectanguloLimite* e *imagenVista*.

El objeto *rectanguloLimite* representa el tamaño que tomará cada una de las imágenes pequeñas, este rectángulo es un objeto invisible que sirve para establecer bien la posición de la pared gráfica, ya que la posición de la pared está determinada por las dimensiones de las imágenes miniatura. Si no se hiciera uso de este objeto la pared cambiaria constantemente de posición cada que se pase el mouse sobre las imágenes, se vería un efecto de brinco en la pared gráfica. Para determinar el tamaño del rectángulo se hace uso de de las constates que definen el tamaño de la imagen miniatura y se hace uso de una nueva constante:

```
Constantes.MINIATURA_EXPANDIDA
```

El otro nuevo elemento que se ha definido es *imagenVista*, este es el objeto que contiene a la imagen que se mostrará y aquí es donde se definen los nuevos eventos de mouse para la imagen:

- *onMouseEntered*. Este evento establece la nueva escala para el objeto *imagenVista* estableciendo el valor de la constante *MINIATURA_EXPANDIDA* para las propiedades *scaleX* y *scaleY*. De esta manera cada vez que se pasa el mouse sobre las imágenes están son escaladas al valor establecido por la constante.
- *onMouseExited*. Este otro evento establece el tamaño normal para las imágenes cuando el puntero no está posicionado sobre las imágenes.

CONCLUSIONES

A través de este trabajo se ha estudiado la tecnología JavaFX y más en específico el lenguaje de programación JavaFX Script, se ha visto todo el entorno que ha dado como resultado la gesta de esta nueva tecnología, así como la comparación con otras tecnologías similares.

El lenguaje JavaFX Script como tal ofrece grandes capacidades para dibujar figuras y manipular imágenes, ya que cuenta con una amplia API para el manejo de efectos visuales. Al ser un lenguaje orientado a objetos y de estructura declarativa, facilita el manejo de código mediante el uso de objetos y sus funciones, lo que lo convierte en un lenguaje fácil de usar y aprender.

La sintaxis del lenguaje está dada en forma estructurada y de forma jerárquica, permitiendo la organización y agrupamiento de varios objetos o nodos, esto supone un mejor control y manejo de los objetos; permitiendo a la aplicación dar ciertos efectos de manera más sencilla.

La API para efectos es amplia y se puede aplicar una gran variedad de combinaciones sobre cualquier imagen, figura, o grupo de objetos. Este es uno de los puntos fuertes de JavaFX Script ya que esta es la parte que hace posible el enriquecimiento gráfico de imágenes y figuras.

En enlace de datos es una característica muy importante del lenguaje, esto permite controlar de manera dinámica el comportamiento de los objetos y por ende de la interfaz gráfica, el comportamiento de dichos objetos está definido por funciones que son ejecutadas o disparadas ante cualquier cambio en la interfaz, y/o por el cambio de valor en ciertas variables; estos cambios se llevan a cabo de forma directa ante los eventos que genera el usuario sobre la interfaz gráfica.

Otra característica muy importante es la posibilidad de integrar código de Java dentro de las aplicaciones hechas con JavaFX Script, esto es posible gracias a que ambas tecnologías se basan y respaldan en la plataforma Java. A fin de cuentas cualquiera de los

dos lenguajes es compilado en el mismo tipo de código intermedio, que después es interpretado por la Máquina Virtual de Java. Las aplicaciones hechas en JavaFX Script cuentan con la misma portabilidad que existe actualmente con las aplicaciones Java. La portabilidad de aplicaciones aventaja a la Plataforma JavaFX sobre otras tecnologías similares, ya que muchas de estas sólo pueden ser ejecutadas sobre algún dispositivo o plataforma en específico.

Una ventaja interesante entre la interacción de Java y JavaFX Script, es la posibilidad de construir interfaces gráficas para otras aplicaciones que ya existen en Java, y de esta manera poder ofrecer una aplicación más amigable y vistosa, conservando la funcionalidad original.

JavaFX Script cuenta con varias librerías que proveen diferentes capacidades a las aplicaciones, por ejemplo existen librerías para el manejo de objetos 3D, el uso fuentes web y hojas de estilo. Estas librerías y otras más añaden mejoras gráficas a las aplicaciones, y permiten hacer a la aplicación más interactiva e independiente; el uso de fuentes web permite contar con información actualizada en todo momento, ya que esta información está disponible en internet, como ejemplo pueden ser estadísticas del clima o simplemente noticias.

JavaFX se posiciona como la nueva propuesta por parte Sun Microsystems para el desarrollo de aplicaciones enriquecidas (RIA), esto surge como necesidad de cubrir el hueco que existía en el sector por parte de Sun Microsystems. Antes de que existiera la plataforma JavaFX ya estaban disponibles otras propuestas en el mercado, estas otras tecnologías se encuentran mejor posicionadas en el mercado, incluso cuentan con versiones superiores a la primera, lo cual indica que ya existe un mejor trabajo en cuanto a pruebas y desarrollo.

Como es natural en toda primera versión de un lenguaje de programación o plataforma, suele haber un tiempo de prueba y aceptación del nuevo lenguaje por parte de la comunidad de desarrolladores. Será cuestión de tiempo ver como se desenvuelve la tecnología JavaFX dentro del sector de las RIA.

Otro punto importante a considerar es que esta es la primera versión de la plataforma JavaFX, lo cual indica que en futuras versiones se contará con un lenguaje más amplio, maduro y robusto. Además de la integración de las aplicaciones entre diferente tipo de dispositivos y/o plataformas; lo cual supone una característica muy importante en cuanto a portabilidad de aplicaciones. JavaFX al formar parte de la familia de tecnologías Java, tiene como respaldo toda la plataforma y tecnologías Java, el camino a recorrer parece un poco más sencillo.

BIBLIOGRAFÍA

- Deitel, Paul J. y Harvey M. Deitel. CÓMO PROGRAMAR EN JAVA. Séptima edición. Prentice Hall, México 2008.
- Ceballos Sierra, Francisco Javier. JAVA 2: INTERFACES GRÁFICAS Y APLICACIONES PARA INTERNET. Alfaomega Grupo Editor, México 2008.
- Clarke, Jim; Connors, Jim; J. Bruno, Eric. JAVAFX: DEVELOPING RICH INTERNET APPLICATIONS. Prentice Hall. Estados Unidos 2009.
- Anderson, Gail; Anderson, Paul. ESSENTIAL JAVAFX. Prentice Hall, Estados Unidos 2009.
- Weaver, James L. JAVAFX SCRIPT: DYNAMIC JAVA SCRIPTING FOR RICH INTERNET/CLIENT-SIDE APPLICATIONS. Apress, Estados Unidos 2009.

FUENTES ELECTRÓNICAS

- *http://javafx.com/*. Sitio Oficial de JavaFX.
- *http://download.oracle.com/javafx/tutorials.html*. Documentación de JavaFX Tutoriales y Artículos.
- *http://javapassion.com/portal/javafx-programming-with-passion/javafx-programming-with-passion*. Curso en línea JavaFX 2009.
- *http://www.revista.unam.mx/vol.1/num2/art4/*. Artículo: Arquitectura de la Máquina Virtual Java.
- *http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/compila.pdf*. Del Funcionamiento y comportamiento de los Compiladores.
- *http://www.bindows.net/features.html*. Características de Bindows.
- *http://www.openlaszlo.org/architecture*. Arquitectura de OpenLaszlo.
- *http://www.adobe.com/products/flex/*. Sitio Oficial de la Plataforma Flex.
- *http://www.silverlight.net/*. Sitio Microsoft Silverlight.
- *http://www.netbeans.org/. Sitio Oficial del IDE Netbeans para Java.*
- *http://www.eclipse.org/. Sitio Oficial de IDE Eclipse para Java.*