GOOGLE WEB TOLKIT (GWT)

HIVY MARGARITA MORA REDONDO
ALFONSO MENESES MERCADO

FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
CARTAGENA DE INDIAS
2010

GOOGLE WEB TOLKIT

HIVY MARGARITA MORA REDONDO
ALFONSO MENESES MERCADO

Monografía presentada para optar el título de Ingeniero de Sistemas

Director
GIOVANNY VASQUEZ
Ingeniero de Sistemas

FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
CARTAGENA DE INDIAS
2010

TABLA DE CONTENIDO

1 INTRODUCCIÓN

Google Web Toolkit es una herramienta de fácil manejo y de gran ayuda para la creación de sitios web, que no necesita de conocimientos profundos sobre Ajax y HTML.

En la actualidad muchos desarrolladores desconocen de este framwork y de su utilización, por esto queremos dar a conocer todo lo referente a GWT, sus características, arquitectura y demás herramientas.

En la web existen muchos tutoriales sobre que es GWT pero en este trabajo no nos dedicaremos solo a la parte teórica sino también a la práctica mostrando paso a paso la realización de ejemplos y de instalación.

En el capítulo 1 hablaremos sobre Google Web Toolkit (GWT) el cual proporciona un conjunto de herramientas para desarrollar funcionalidades JavaScript de alto rendimiento, soporta la mayoría de las características del lenguaje de programación y emula una gran parte de las librerías Java.

En el capítulo 2 mostraremos paso a paso como es la configuración e instalación de Google Web Toolkit (GWT) para el sistema operativo Windows, así como la configuración del entorno de desarrollo Eclipse el cual es con el que trabajaremos en el presente documento.

En el capítulo 3 hablaremos sobre que es Widgets y Paneles los cuales pueden ser creados, modificados o utilizar los que proporciona GWT. Los paneles están diseñados para contener Widgets dentro de estos. Un Widgets son componentes visibles de una aplicación como son botones, menús, caja de textos, entre otros. Los Widgets y paneles trabajan de igual manera sobre los diferentes navegadores y eliminan la necesidad de escribir un código especial para cada uno de ellos.

En el capítulo 4 hablaremos sobre que son Widgets compuestos que son la combinación de los Widgets y Paneles que proporciona GWT con los Widgets creados por el usuario, con el fin de realizar controles de usuario personalizados y reutilizables.

En el capítulo 5 se hablará sobre RPC el cual es un protocolo que permite a un programa ejecutar código en otra máquina remotamente sin preocuparse por la comunicación entre ambas. En GWT se trabaja con RPC debido a que no es necesario de otras páginas web mientras es ejecutada la aplicación que corre sobre el navegador, también facilitan el envío y admisión de objetos java sobre HTTP, mejora el ancho de banda y la carga del servidor

GWT necesita para su desarrollo un entorno en el cual puede implementar las aplicaciones Java correspondientes por eso en el Capitulo 6 mostraremos los diferentes entornos en los que puede trabajar GWT. En el presente trabajo se trabajará con el entorno de desarrollo Eclipse la cual es una plataforma de software de código abierto. El motivo de escoger este entorno es debido a que es el que venimos utilizando en la universidad y tenemos más conocimiento de este, no por esto los otros entornos son menos importantes.

Por último en el Capitulo 7 se construirá una aplicación GWT con el entorno de desarrollo Eclipse para logra mostrar al lector todo lo explicado y así poner en práctica lo aprendido.

2 OBJETIVOS

2.1 Objetivo general:

Explicar las principales características y utilidades de Google Web Toolkit como herramienta eficaz para el desarrollo de entornos web, utilizando un lenguaje de propósito general como JAVA.

2.2 Objetivos específicos:

- ✓ Conocer las características de GWT y sus principales componentes.
- ✓ Realizar un manual para la ayuda del desarrollador web que desee utilizar GWT.
- ✓ Desarrollar ejemplos sobre la aplicación de GWT y la creación de paneles y Widgets.
- ✓ Mostrar de manera práctica la instalación de GWT en el entorno de desarrollo Eclipse.

3 JUSTIFICACIÓN

Actualmente nos encontramos en una época donde la mayoría de las cosas se realizan por medio de la web.

En este trabajo queremos realizar un manual para la ayuda de aquellos programadores Web que han oído hablar sobre Google Web Toolkit (GWT) y quieren empezar la realización de sus proyectos con esta herramienta, que les permitirá la creación de páginas web dinámicas y bien diseñadas sin la necesidad de conocer mucho sobre JavaScript y HTML, bastará con saber programar en JAVA.

Este manual les ayudará con los conceptos básicos sobre Google Web Toolkit, mostrando ejemplos que le serán de gran utilidad al lector al momento de empezar a utilizar Google Web Toolkit.

4 INTRODUCCIÓN A GOOGLE WEB TOOLKIT (GWT)

Google web toolkit es un framework desarrollado en Java de código abierto creado por google que permite ocultar la complejidad de la tecnología AJAX programando desde Java, traduciendo y compilando el programa a JavaScript y HTML con cualquier navegador web.

GWT no es solo una interface de programación; está proporciona un conjunto de herramientas las cuales permiten desarrollar funcionalidades Javascript de alto rendimiento en el navegador del cliente.

El compilador de GWT soporta la mayoría de las características del lenguaje de programación, Mientras que las librerías GWT runtime emulan una gran parte de las librerías de Java.

4.1 CARACTERISTICAS

- ✓ Compatible con la mayoría de navegadores de una forma sencilla.
- ✓ Es un proyecto de código abierto.
- ✓ La interfaz de usuario contiene componentes dinámicos y reutilizables.
- ✓ RPC (*Remote Procedure Call*, *Llamada a Procedimiento Remoto*) fácil. Su mecanismo permite incluso manejar jerarquías de polimorfismo en clases y las posibles excepciones.
- ✓ Administración del historial del navegador. No es necesario llamar a otras páginas para realizar las diferentes acciones, ni recargar el navegador ya que él para no cargar muchas páginas lo que hace es que se actualiza en la misma página.

- ✓ Depuración en tiempo real. Mientras desarrollas tu código este se va traduciendo en JavaScript por medio de una Java Virtual Machine (JVM) permitiendo depurar la aplicación con los sistemas avanzados de debuggin y manipulación de excepciones incluidos en IDEs.
- ✓ Integración con Junit. Mediante esta integración se puede probar las aplicaciones y depurarlas en un navegador mientras se construye, también se puede probar las RPC esto es gracias a que GWT dispone de TestCase especiales que permiten dichas pruebas unitarias.
- ✓ Internacionalización.
- ✓ Se puede mezclar JavaScript en el código de la aplicación cuando las clases GWT no son suficientes.
- ✓ Posee componentes de interface de usuario dinámicos y re-utilizables, además que puedes enviar tus Widgets a otros usuarios en archivos JAR.

4.2 PORQUE USAR GWT

- ✓ Pude convertir tus aplicaciones AJAX en una plataforma sólida para su desarrollo.
- ✓ Se puede usar cualquier IDEs (eclipse, Junit,…).
- ✓ Los errores son detectados mientras se desarrolla la aplicación y no a final de esta.
- ✓ El código es más comprensible y tiene menos documentación debido a que los diseños en Java basados en programación orientada a objetos es mucho más fácil de comunicar y de entender.
- ✓ GWT utiliza el proceso de refactorización o Refactoring el cual consiste en modificar el código fuente de un sistema que ya ha sido diseñado con anterioridad y codificado de manera inexperta para lograr un sistema más estructurado y fiable.

4.3 COMO SE USA GWT

Las aplicaciones GWT pueden ser ejecutadas en:

- ✓ **Modo hosted:** en este modo la aplicación corre como bytecodes de Java sobre una maquina virtual, ya que en este modo se cuenta con todas las ventajas que posee Java para depurar, tiende a gastarse más tiempo para su desarrollo.
  Para soportar este modo, GWT cuenta con un navegador especial que está acoplado a la maquina virtual de java.

- ✓ **Modo Web:** en este modo la aplicación corre como HTML + JavaScript sobre un navegador traduciéndolo desde el código original en Java por medio del compilador de GWT. Cuando ya esté la aplicación terminada solo hay que montarla en un servidor web para que los usuarios finales accedan a ella por medio de un navegador en modo web.

4.4 ARQUITECTURA DEL GWT

GWT posee cuatro componentes principales:

- ✓ **Compilador GWT Java a JavaScript:** Traduce el lenguaje de programación a JavaScript de Java. Es utilizada cuando la aplicación se necesitada que corra en modo Web.
- ✓ **Navegador Web Hosted de GWT:** Permite correr y ejecutar GWT en modo hosted, el cual corre bytecodes de Java en una maquina virtual sin ser compilados en JavaScript. Para lograrse esto el navegador de GWT aloja un controlador de browser especial (un control del Internet Explorer sobre

Windows o un control de Gecko/Mozilla sobre Linux) con hooks dentro de la JVM.

- ✓ **Emulación de librerías JRE:** GTW contiene las librerías de clases más usadas como implementación en JavaScript. GWT incluye la mayoría de las clases del paquete java.lang y un subconjunto de clases del paquete java.util. el resto de las librerías no son soportadas por GWT como por ejemplo el paquete de java.io ya que esta no se utiliza en aplicaciones web debido a que acceden a recursos en la red y al sistema de archivos locales.
- ✓ **Librería de clases de interfaz de usuario de GWT:** es el núcleo de las librerías de interfaz de usuario para crear aplicaciones GWT permitiendo crear Widgets.

4.5 MODULOS

Son archivos XML que reúnen todos los datos de configuración que un proyecto de GWT necesita como los módulos heredados, nombre de clase, entrada a los source paths y a los public phats. Pueden aparecer en cualquier paquete en el

classpath, es recomendable que estos aparezcan en el paquete raíz de un proyecto estándar.

- **Clases entry-point:** es cualquier clase asignable a EntryPoint y que puede ser construida sin parámetros.
- **Source Path:** los archivos localizados en el source path son candidatos para ser traducidos a JavaScript, haciendo que se mezclen los códigos fuentes del lado del cliente con los del lado del servidor en el mismo classpath sin ningún conflicto. Al momento de que un modulo hereda a otro sus source path son combinados teniendo acceso al código fuente traducible que requiera cada modulo.
- **Plublic path:** al compilar la aplicación a JavaScript todos los archivos que puedan encontrase sobre el public path de la aplicación, son copiados al directorio de salida de los módulos. En la red las URLs que son visibles al usuario no necesitan incluir un nombre de paquete completo. En los public path también sucede como en los source path al momento de que un modulo hereda a otro.

4.5.1 ESPECIFICACIONES

- ✓ **Formato de modulo XML:** los módulos son definidos y son situados dentro de la jerarquía de paquetes del proyecto.
- ✓ **Inclusión automática de paquetes:** cuando el modulo es cargado se carga también las referencias a archivos JavaScript y CSS externos.
- ✓ **Filtrado de paquetes públicos:** para poder evitar la publicación accidental de archivos, se filtran los archivos dentro y fuera del public path.

4.5.2 FORMATO DE MÓDULOS XML

Los módulos son definidos en ficheros XML cuya extensión de archivos es .gwt.xml. En el paquete raíz del proyecto deben residir los archivos de módulos XML.

Ejemplo:
<module>
<inherits name=”com.google.gwt.user.User”/>
<Entry-point class=”com.example.cal.client.CalendarApp”/>
</module>

4.5.3 CARGANDO MODULOS

Los módulos XML son encontrados en el classpath de Java y son referenciados con su propio nombre de modulo desde las páginas alojadas en el proyecto.
Los módulos son referenciados por su nombre lógico el cual es de tipo pkg1.pkg2.NombreModulo y excluyendo la extensión del archivo, un ejemplo puede ser:

~/src/com/ejemplo/cal/Calendario.gwt.xml
Es
Com.ejemplo.com.ejemplo.cal.calendario

4.6 GWT WIDGETS Y PANELES

Los paneles están diseñados para contener Widgets dentro y son usados para determinar como la interface de usuario será distribuida sobre el navegador utilizando códigos en java del lado del cliente.

Se le denomina Widgets a objetos como botones, caja de texto y arboles. En GWT no se está limitado a un conjunto de Widgets ya que se pueden crear Widgets personalizados de diferentes formas.
Los Widgets y paneles trabajan de igual manera sobre los diferentes navegadores eliminando la necesidad de escribir código especial para cada navegador.

4.7 ALGO DE RPC

El RPC es un protocolo que permite a un programa de un ordenador ejecutar código en otra máquina remota sin preocuparse por las comunicaciones entre ambos. En GWT no es necesario de otras páginas web mientras se ejecutan dichas aplicaciones ya que corren como aplicaciones sobre el navegador.

Las RPC son muy utilizadas dentro del paradigma cliente-servidor facilitando al cliente enviar y recibir objetos de java sobre HTTP. También con las RPC se tiene la oportunidad de mover toda la lógica de la interfaz de usuario al cliente mejorando así el funcionamiento de la aplicación, reduce el ancho de banda usado y la carga del servidor, y al usuario final le presenta una forma más agradable de navegar por la pagina.

5 CREANDO UNA APLICACIÓN GWT

5.1 COMO INSTALAR Y CONFIGURAR GWT

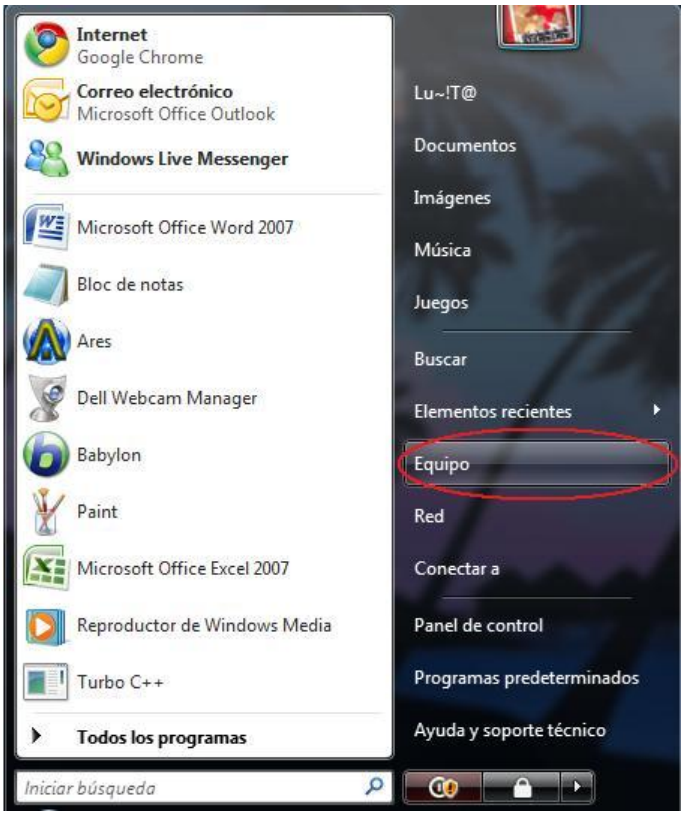
A continuación se mostrara detalladamente como instalar y configurar GWT para Windows.

1. Lo primero que se debe hacer es descargar el archivo que se usara para el sistema operativo Windows. En la siguiente dirección se podrá descargar el archivo: http://code.google.com/webtoolkit/download.html

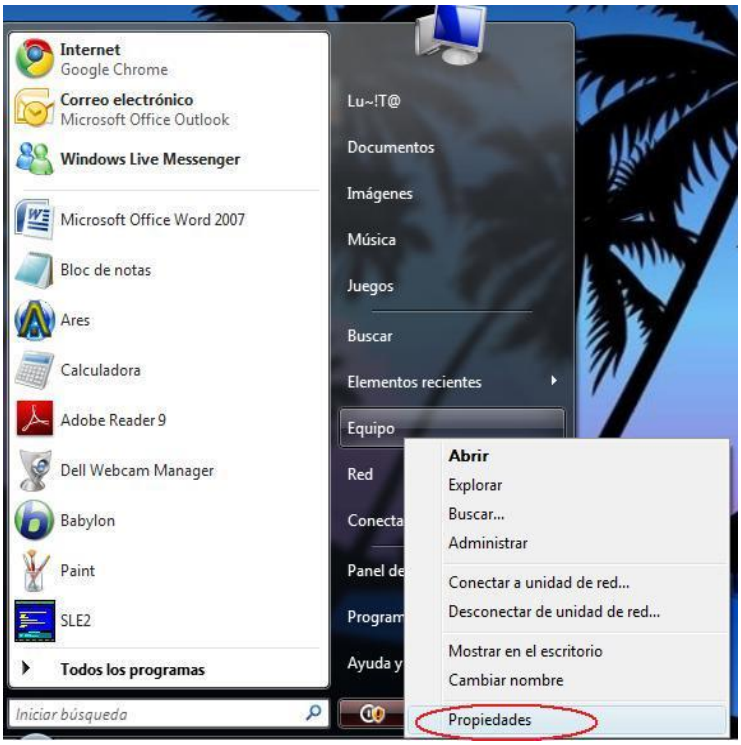
2. Después de descargar el archivo de GWT para Windows se descomprime en una carpeta.

3. Ahora se va a añadir su ruta en el path del sistema operativo.

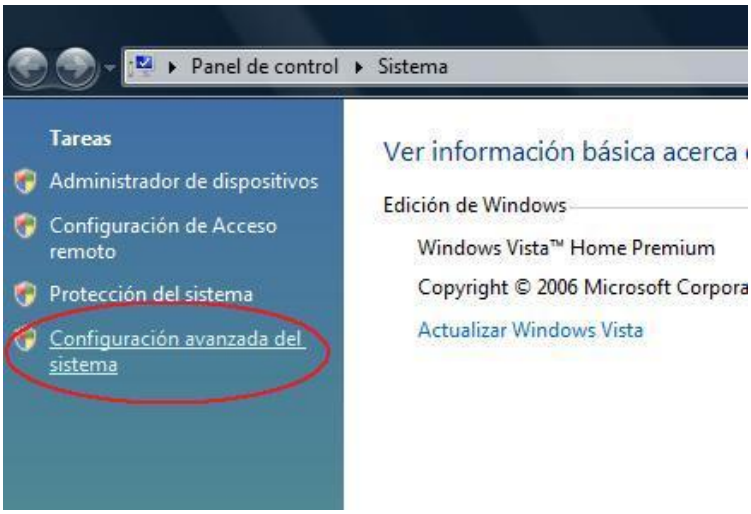
4. Se hace clic derecho en Equipo.



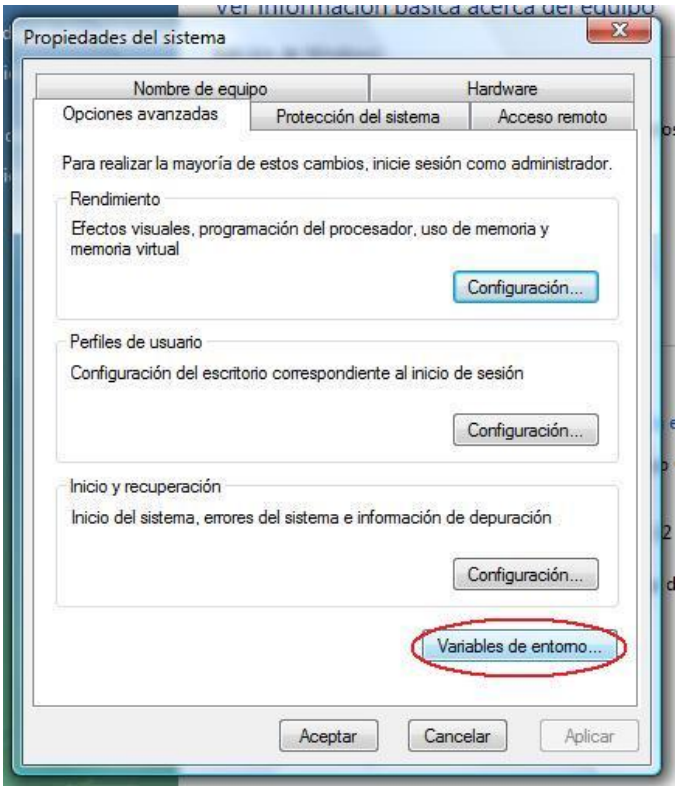
5. Luego hacer clic en Propiedades.



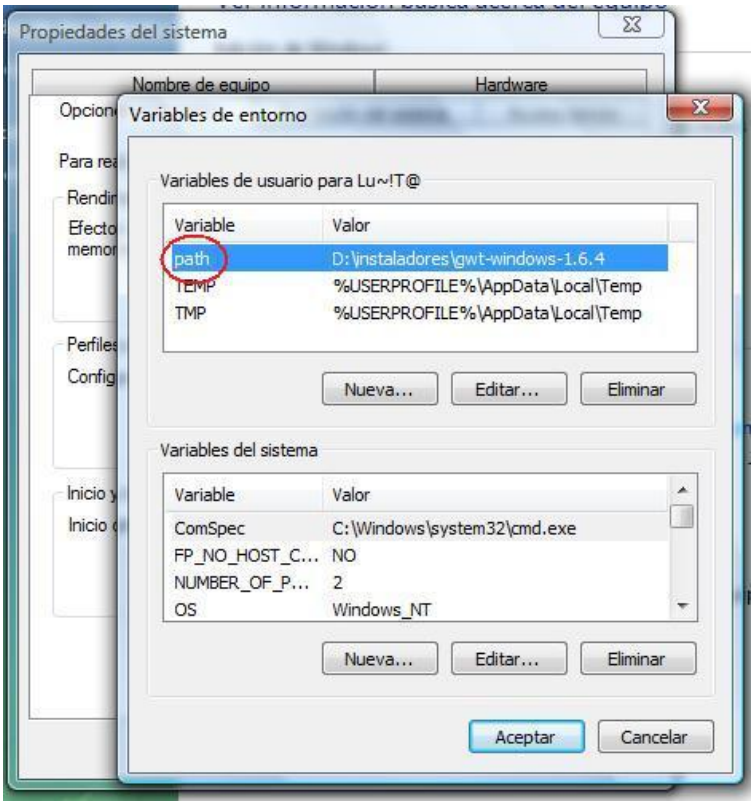
6. Después hacer clic en Configuración Avanzada del Sistema que se encuentra en el menú de la parte izquierda de la ventana



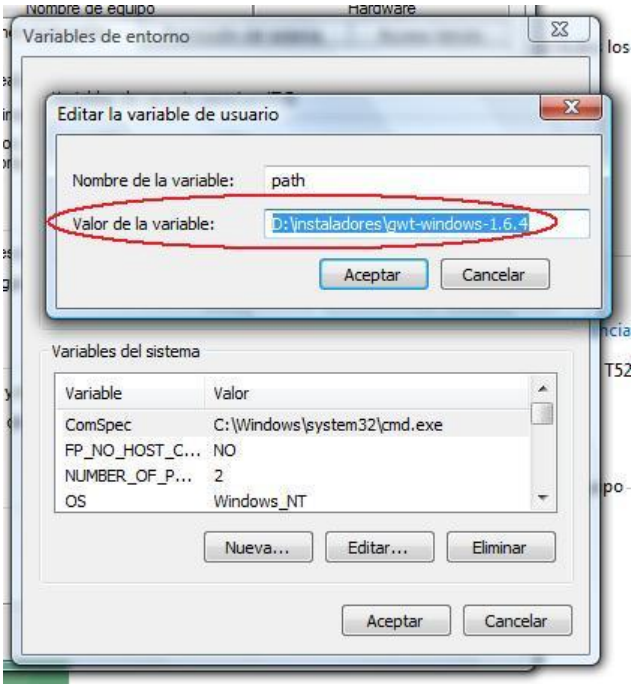
7. Posteriormente hacemos clic en Variables de Entorno…



8. Por último en Variables de Usuario se da doble clic a Path.



9. Después de darle doble clic a Path se añade la ruta donde descomprimimos el GWT.

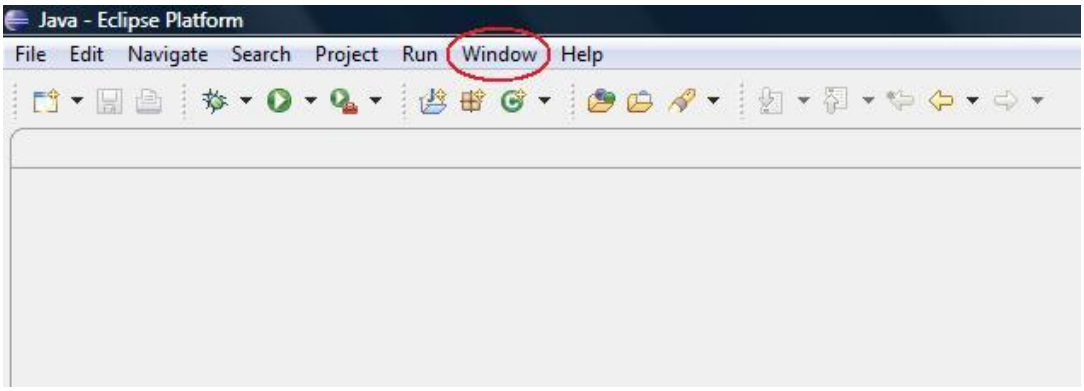


Esto se realiza con el fin de buscar la comodidad a la hora de desarrollar, debido a que de esta manera no es necesario situarse en el path de instalación, desde la línea de comandos, para ejecutar los comandos más comunes.

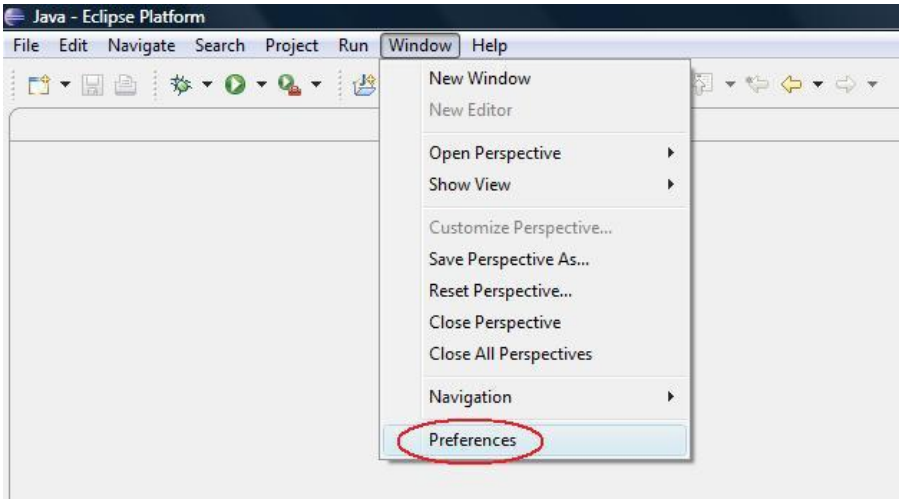
En este documento se trabajará con el entorno de desarrollo Eclipse. Para configurar GWT en este entorno se hará lo siguiente:

1. Se ingresa al entorno de desarrollo Eclipse.

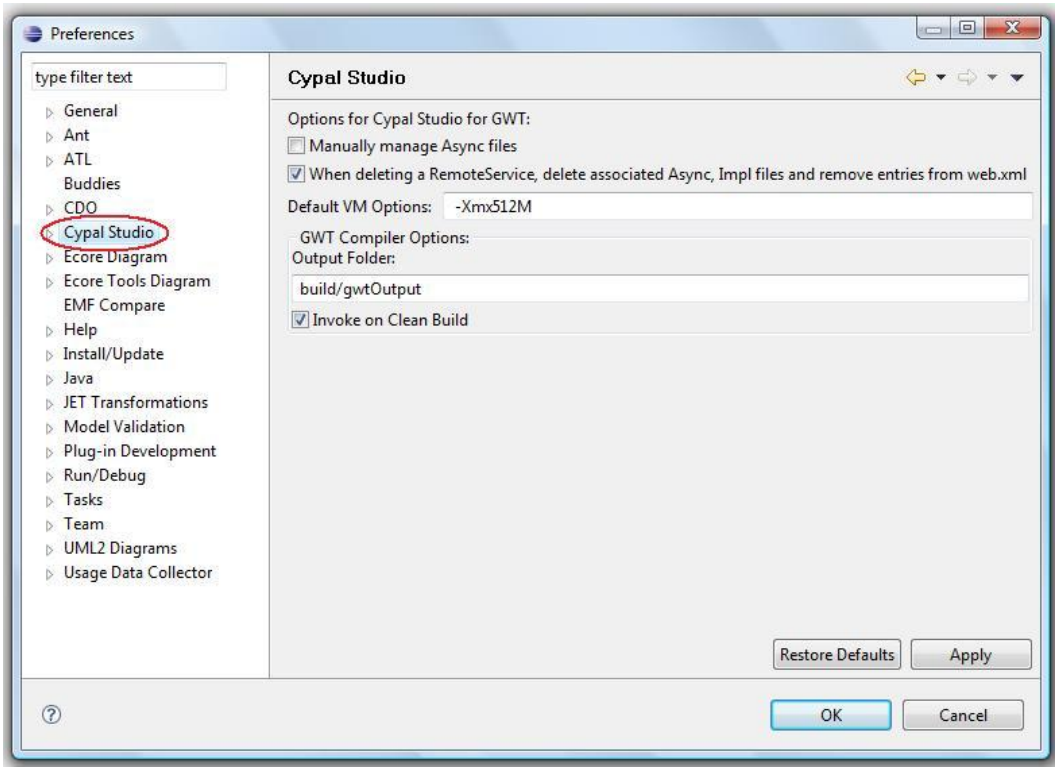
2. Luego se van al menú de Windows.



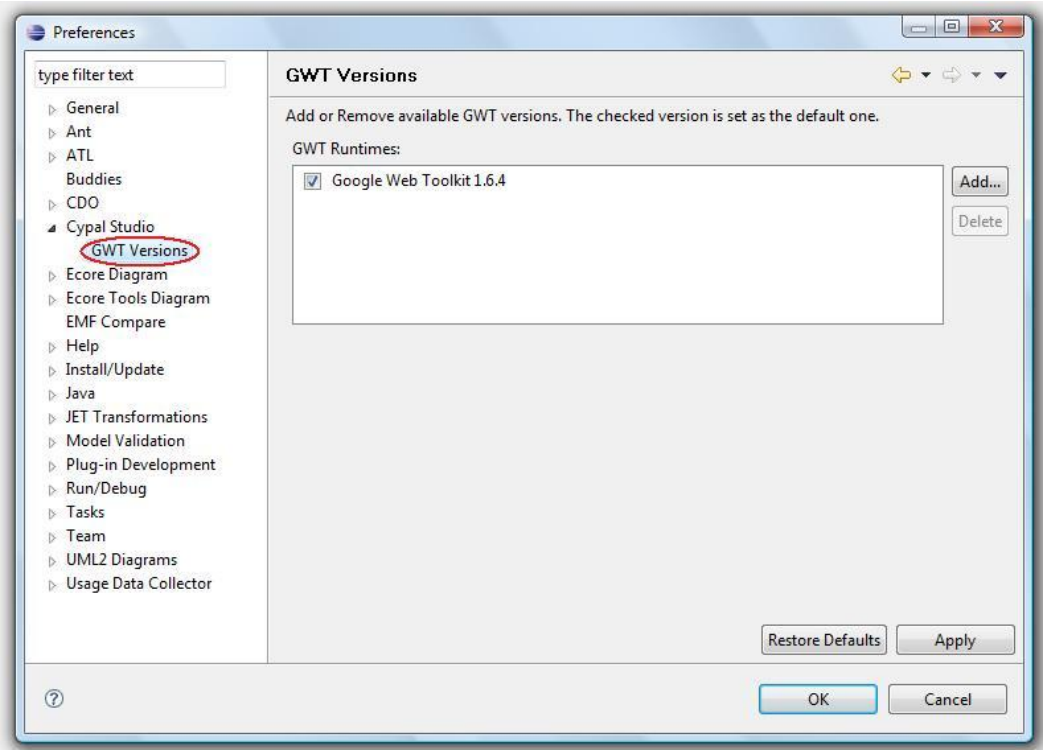
3. Se sitúan donde dice Preferences



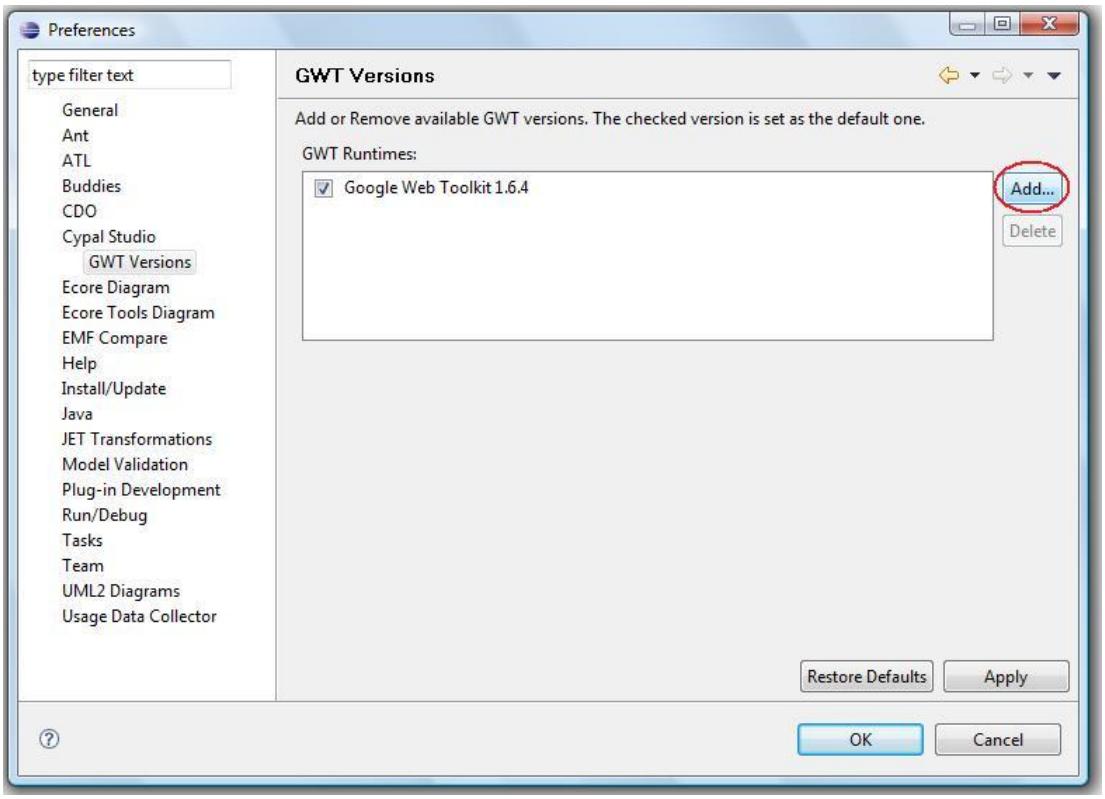
4. Dan clic donde dice Cypal Studio



5. Se sitúan en GWT Versions



6. Le dan en Add…



7. Buscar la carpeta donde se encuentra descomprimido GWT y se señala, dar OK y luego Aceptar.

6 WIDGETS Y PANELS

6.1 ¿Qué es un Widget?

Los Widget son los componentes y partes visuales de una aplicación GWT los cuales son vistos por los usuarios que usan la página web, como por ejemplo labels, botones, imágenes, menús, grids etc. La GUI de una aplicación GWT está basada en Widgets.

Una definición formal de Widgets puede ser la siguiente:
Los Widgets son los componentes visibles de una aplicación GWT que un usuario puede ver en la página por la que esta navegando.

El paquete com.google.gwt.user.client.ui contiene las clases que permiten crear interfaces de usuario dinámicas usando técnicas que ya se han probado exitosamente con otros frameworks que proveen esta funcionalidad como por ejemplo el paquete AWT de Java.

Las clases en este paquete aprovechan las características de las interfaces de usuario de un navegador web para proveer componentes dinámicos reutilizables, los cuales tienen el mismo comportamiento en las versiones e implementaciones de los navegadores soportados. La librería de interface de usuario de GWT llama a estos componentes Widgets, y estos van desde simples botones o labels hasta controles más complejos como tabs o arboles. Algunos de estos Widgets se traducen directamente en elementos HTML que normalmente se usaría en la elaboración de una página web, mientras que otros son la composición de muchos elementos HTML combinados con scripts y manejo de eventos.

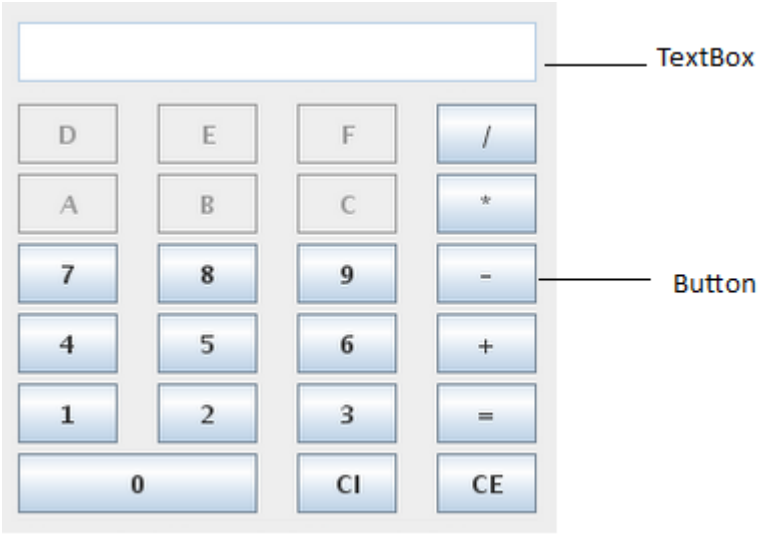
Los Widgets son uno de los 4 cuatro bloques fundamentales de construcción de aplicaciones GWT(los otros son los panels, eventos y la comunicación con el servidor).

Cuando un usuario está utilizando una aplicación GWT, lo que se encuentra mirando es un grupo de Widgets distribuidos de cierta manera en paneles y que responderán a eventos.

Los Widgets son los componentes con los que el usuario interactúa. GWT proporciona muchos Widgets diferentes, entre estos los controles más usados en las interfaces gráficas como lo son los botones, labels, menús cuadros de textos, etc.

Muchas de las aplicaciones serán construidas usando varios Widgets los cuales se distribuirán de cierta manera dentro de paneles para darle alguna estructura a la interfaz de la aplicación.

Un buen ejemplo de mostrar lo anterior es la interfaz gráfica de una calculadora



Ejemplo de la distribución de Widgets en una interfaz gráfica

6.2 Usando Widgets como objetos Java

El propósito general de GWT es desarrollar rich internet applications en Java y después el compilador de GWT (como ya se vio en capítulos anteriores) genera el código HTML y java script necesario para que la aplicación funcione bien en los diferentes navegadores existentes. Para eso se hace necesaria una manera de representar varios objetos del navegador, es decir los llamados Widgets de GWT, en nuestro código Java.

Se tomará ventaja de la habilidad de la programación orientada a objetos para modelar objetos y conceptos como objetos de programación. Por ejemplo, en una aplicación GWT cualquiera se hará uso de un objeto Java llamado Button este objeto modelará varias propiedades que se espera que un botón tenga, como la habilidad de asignarle un texto además de permitir oprimir dicho botón. Así como modelamos un botón se puede modelar los demás componentes que se desean ver en la aplicación, es decir los Widgets, como objetos Java con métodos y propiedades.

Entonces, en la programación diaria en GWT se irá considerando todos los Widgets como objetos naturales de Java. El botón del que se hablaba anteriormente se crea llamando al constructor del Button de GWT como sigue:

Button boton = new Button (“Aceptar”);

Este código crea un nuevo objeto Button de GWT del cual después se puede usar varios métodos.

Esta vista de los Widgets es bastante sencilla para cualquiera familiarizado con la programación en Java u otro lenguaje orientado a objetos como C#, se crean objetos de clases y se llaman a los métodos de esas clases. Lo que da esta

manera de ver las cosas es el entendimiento de cómo los Widgets son mostrados en la página web, esta cuestión es dada por una representación alternativa de los Widgets la cual es la DOM.

6.3 Considerando Widgets como elementos DOM

La representación Java de los Widgets que se acaba de ver funciona muy bien en el código java y permite construir una aplicación GWT como se pretende, usando cualquiera cantidad de Widgets y usando sus métodos asociados para construir la funcionalidad de la aplicación. Sin embargo no se puede mostrar esos objetos Java en el navegador, así que no se tiene todavía una aplicación Ajax. Aquí es donde la representación DOM de los Widgets entra en juego.

Todos los Widgets de GWT tienen una alternativa representación DOM que es construida paralelamente con el objeto Java. El constructor por lo general es el responsable de crear esta representación DOM, así que si se mira más en detalle el constructor de la clase Button, se verá algo como esto:

```
public Button() {
        super(DOM.createButton());
        adjustType(getElement());
        setStyleName("gwt-Button");
}
```

La llamada a DOM.createButton() crea el elemento DOM <button> a través de las clases DOM de GWT. Además en el constructor del padre, el método setElement es llamado para asignar la representación DOM a este nuevo elemento <button>. Es usando este valor asignado por el método setElement que se tiene acceso a la

representación DOM desde la perspectiva java. Si se fuera a ver la representación DOM de un Button de GWT se vería algo como esto:

```
<button class=”gwt-Button”
        eventbits="7041"
        onchange="null"
        onload="null"
        onerror="null"
        Click me
</button>
```

Esta es la representación DOM estándar de un objeto <button> con un par de atributos adicionales específicos de GWT. Estos atributos adicionales son asignados por el constructor del objeto cuando esta representación DOM es creada. No hay que confundir el primer atributo “class”, algunos pueden cometer el error de pensar que se trata de la clase Java de dicho Widgets pero no es así, este atributo se refiere al estilo CSS que puede ser aplicado al Widgets.

El siguiente atribute “eventbits”, es uno de los que se verá en muchos Widgets y le indica a GWT que tipo de eventos son escuchados por el Widgets.

No se debe confiar en el hecho de que un Widgets en particular es implementado como un elemento DOM en particular, ya que dicho Widgets puede ser implementado usando una representación DOM diferente en versiones futuras. Si se concentra en usar el código java usando los métodos que no da cada Widgets de GWT se protege contra cambios futuros a nivel DOM.

Ahora que se sabe que es un Widgets, se mirara algunos Widgets que provee GWT.

6.4 Widgets Estándar

La versión estándar de GWT viene con amplio número de Widgets para usar en las aplicaciones. Estos Widgets cubren las áreas que uno esperaría con botones, cuadros de texto y otros Widgets. Sin embargo habrá ocasiones en las que querrá usar algún Widgets, como una barra de progreso, pero que no viene dentro de los Widgets que GWT proporciona pero que usted podrá crear (ya veremos más adelante como hacer esto).

Dentro del conjunto de Widgets, los diseñadores de GWT han implementado una fuerte jerarquía de clases de Java con el fin de proporcionar un elemento de coherencia en los Widgets donde dicha coherencia naturalmente exista. Por ejemplo tomemos los Widgets TextBox, TextArea y PasswordTextBox, es natural pensar que ellos compartan ciertas propiedades. GWT reconoce esto y captura las propiedades comunes en una clase llamada TextBoxBase de la cual heredan estos 3 Widgets. Para ilustrar esto de la jerarquía se muestra la siguiente figura

Jerarquía de clases de los Widgets GWT. Tipos de eventos que aceptan.
Gráfico tomado de Manning GWT in Action Easy Ajax with the Google Web Toolkit Jun 2007

Se puede ver en la jerarquía de clases que todos los Widgets heredan de la clase UIObject, la cual proporciona un conjunto de métodos y atributos para todos los Widgets, incluyendo el tamaño, visibilidad y nombres de estilos. Es dentro de esta clase que se encuentra el método setElement() del que se hablaba anteriormente. Las subclases de UIObject deben llamar a este método como la primera cosa que hacen antes llamar a cualquier otro método para asegurarse que el vínculo a algún elemento del navegador es establecido.

La clase UIObject permite el acceso a una variedad de funcionalidades DOM sin que se tengamos acceso directo al DOM. Por ejemplo es posible asignar el height

(alto) de un UIObject usando el setHeight() método, el cual usa el método setStyleAttribute() de la clase DOM

```
public void setHeight(String height) {
DOM.setStyleAttribute(element, "height", height);
}
```

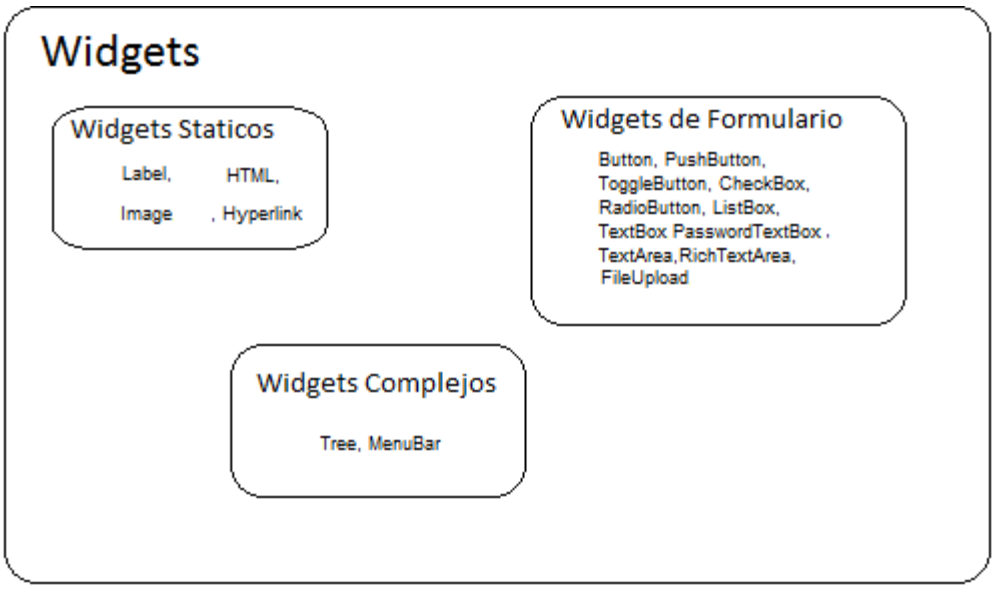
Los otros métodos de este estilo incluyen la habilidad de asignar el width (ancho), titulo (lo que se muestra cuando el mouse pasa por encima del elemento). Todos estos métodos toman parámetros tipo String.

Después de UIObject todos los Widgets, excepto TreeItem y MenuItem, deben heredar de la clase Widget, la cual proporciona todas las funcionalidades a los Widgets, incluyendo métodos que son llamados cuando un Widgets es agregado o eliminado de un panel. Esta clase también contiene la implementación por defecto del método onBrowserEvent, el cual permite a un Widgets maneje cualquier evento que tenga relacionado.

A continuación se verá los Widgets que proporciona gratis GWT, así como la manera de usarlos en una aplicación. A continuación se dividirá los diferentes tipos de Widgets en las siguientes categorías:

- ✓ **Widgets estáticos,** que son aquellos que no son muy interactivos, solo cambian de estado como resultado de alguna orden dada por la aplicación.
- ✓ **Widgets de formularios,** acá encontraremos aquellos elementos que normalmente son usados para formularios , incluyendo botones y cuadros de texto

- ✓ **Widgets complejos,** los cuales son nuevas características, brindadas por las interfaces de usuario , para navegadores los cuales son composiciones de etiquetas HTML y manejo de eventos con java script



Se tratara los Widgets en detalle en las categorías que mostró la anterior figura.

6.4.1 Widgets Estáticos

Estos Widgets no tienen ningún estado interno ni tampoco cambian dinámicamente por su cuenta. Sin embargo pueden ser parte de una interface de usuario dinámica en la cual sus propiedades pueden ser cambiadas en tiempo de ejecución por medio del código de otros controles, pero sus propiedades no cambian como resultado de acciones del usuario. Estos Widgets incluyen: Label, HTML, Image y el Hyperlink.

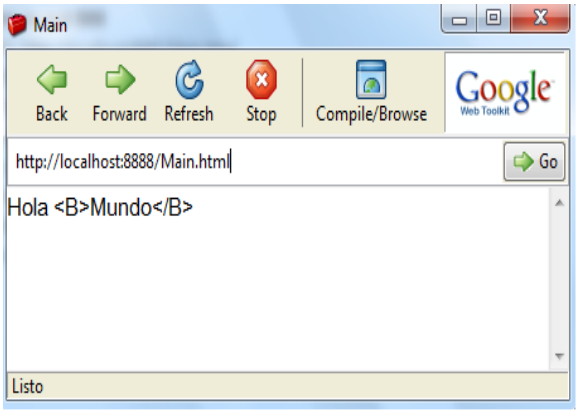
6.4.1.1 Label

Un Label contiene un texto específico, el cual es visualizado exactamente como es escrito. Esto significa que el Label creado por el código new Label ("hola <B>Mundo</B>") aparecerá en el navegador exactamente así "Hola <B>Mundo</B>, la palabra Mundo no es interpretada como HTML y no será visualizada en negrilla.

Cuando se agrega un Label a un Widget padre, simplemente se muestra una cadena la cual no es HTML. El siguiente código muestra como agregar un Label a un contenedor:

```
mipanel.add(new Label("Hola <B>Mundo</B>"));
```

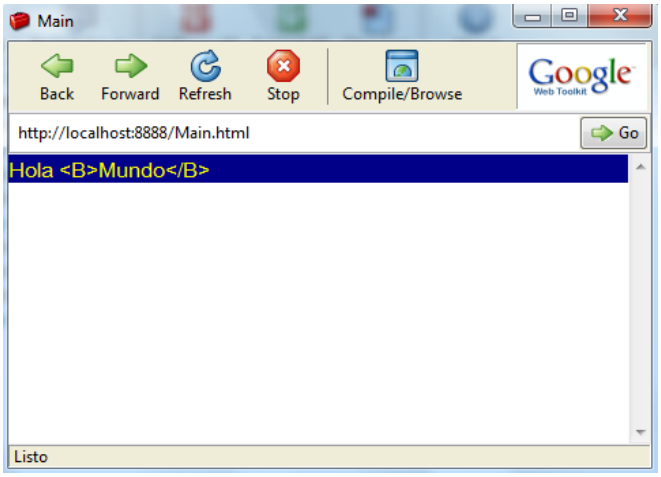
Se instancia el Label con la cadena que se desea mostrar. El contenedor padre, en este caso mipanel, determina el lugar donde colocar el Label dentro de la interface. El Label simplemente sabe cómo llenar su espacio.



Un Label tiene asociado por defecto un nombre de clase CSS: gwt-Label, el cual permite fácilmente asignar el estilo para todos los Label de nuestra

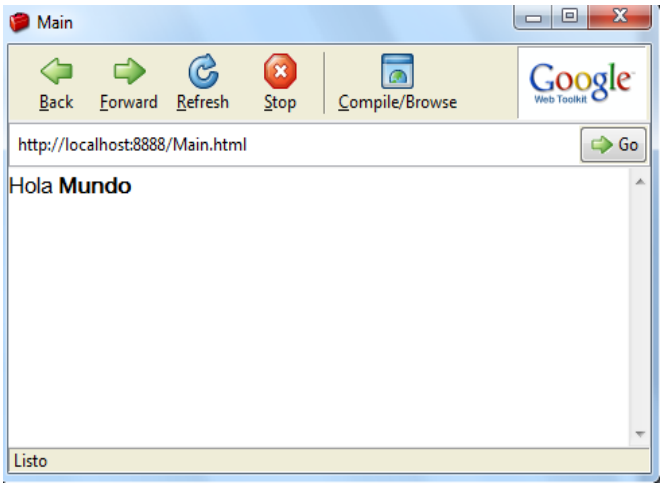
aplicación. Por ejemplo, agregando el siguiente código CSS hará que el Label tenga un fondo azul con la fuente amarilla:

```
.gwt-Label { background-color:#008; color:#FF0;}
```



6.4.1.2 HTML

El Widget HTML es muy similar al Label pero con la pequeña diferencia que puede soportar y renderizar HTML. De hecho este Widgets hereda de Label, ganando su soporte para eventos del mouse y alineación de texto. Este Widgets interpreta cualquier texto como HTML. Si en un Label el texto “Hola <B>Mundo</B>” se vería tal y cual se escribió, en un HTML se vería “Hola **Mundo**”



Su estilo de CSS predefinido es gwt-html, esto permite personalizar la apariencia a través de hojas de estilo así como el Label. Este control tiene algunas desventajas, ya que puede generar problemas de seguridad u otros debido a que puede llevar código malicioso.

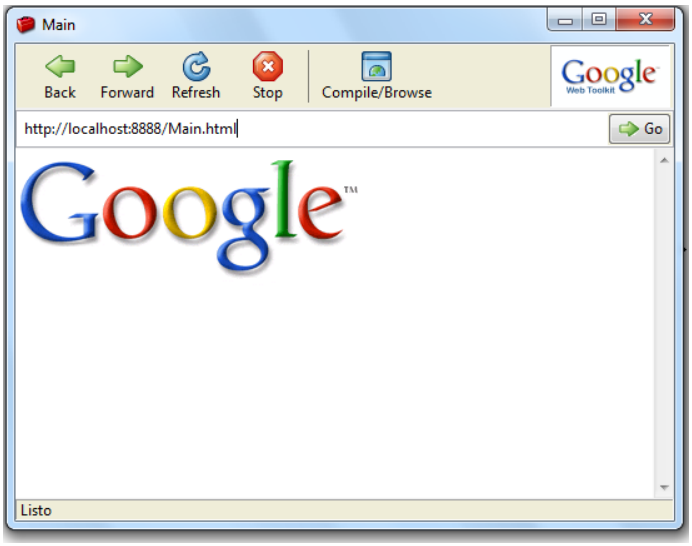
6.4.1.3 Image

El Widgets Image es estático igual que el Label y el HTML, pero en vez de aceptar y renderizar una cadena, el Image acepta una URL apuntando a un archivo de imagen y renderiza dicha imagen. El Widgets básicamente lo que permite es utilizar la etiqueta img de HTML como un Widgets del framework. El siguiente código nos permite crear una imagen:

```
Image imagen = new Image();
imagen.setUrl("archivo.jpg");
RootPanel.get().add(imagen);
```

Como se ve lo que se realiza es crear una instancia de la clase “image”, luego se pasa la url de la imagen que se desea mostrar a través del método “setUrl”, y finalmente se agrega al panel principal. El método “setUrl” acepta una cadena con la url donde se encuentra la imagen, dicha url puede ser una dirección local como en el ejemplo que se mostro donde el navegador buscará una imagen que se llama “archivo.jpg” en el mismo directorio del HTML.

Pero también se puede colocar la url de la imagen en otro servidor, por ejemplo si ahora se usa esta url “***http://www.google.com.co/intl/en_com/images/logo_plain.png***“, muestra lo siguiente.



Un aspecto interesante de este Widgets es la habilidad de agregar un loadlistener lo que permitirá ejecutar ciertas acciones si la imagen se carga correctamente u otras acciones si hubo algún error al momento de cargar la imagen en el browser. Esto se puede hacer de esta manera:

```
imagen .addLoadListener(new LoadListener() {
        public void onLoad(Widget sender) {
                //success!, do nothing
        }
        public void onError(Widget sender) {
                imagen.removeFromParent();
        }
});
```

Lo anterior lo que hace es agregar un nuevo loadlistener a la imagen que ya se había creado, al hacer esto se tiene que implementar los métodos onLoad, el cual se ejecuta cuando la imagen se cargo correctamente, y onError, que se ejecuta si hubo problemas al cargar la imagen. En el ejemplo anterior no se

hace nada cuando la imagen cargo exitosamente, y si hubo algún error al mostrar lo que se hace es removerla del panel.

6.4.1.4 Hyperlink

Este Widgets actúa como un hipervínculo dentro de la aplicación GWT. Para el usuario aparenta no ser más que un vínculo normal dentro de la página web el cual al dar clic se realizara una navegación dentro de la página. En el código esta acción lo que hace es manipular el objeto “History” de GWT para cambiar el estado de la aplicación.

Cualquier componente que use este Widgets debe implementar la interface ValueChangeHandler, que es la que maneja los eventos relacionados con cambios de estados en nuestra aplicación, e implementar el método onValueChange, que es el que se ejecuta cuando se realiza un cambio en algún estado de la aplicación, en el cual ira las acciones a tomar dentro de la aplicación cada vez que un Hyperlink es utilizado. Además hay que agregar la siguiente línea al HTML de la aplicación:

<iframe id="__gwt_historyFrame”
style="width:0;height:0;border:0"></iframe>

Para utilizar un Hyperlink creamos una instancia de la clase Hyperlink y luego agregamos el Widgets a un panel.

Hyperlink h1 = new Hyperlink("Hyperlink 1","imagen1");

El constructor del Hyperlink recibe como parámetros 2 Strings, el primero el texto del link, y el segundo una cadena que se convierte en el estado del objeto “History” q se mensiono anteriormente, al dar clic en un Hyperlink esta cadena se concatena junto con un carácter ‘#’a la url de la aplicación.

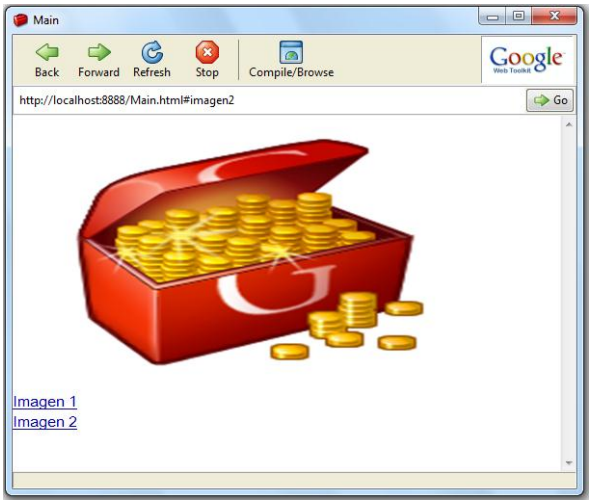
A continuación se muestra un ejemplo

```
public class Main implements EntryPoint,ValueChangeHandler<String>{

    Image imagen;
    public void onModuleLoad() {

        History.addValueChangeHandler(this);
        imagen = new Image();
        imagen.setUrl("http://fluxqubit.files.wordpress.com/2008/03/0074.jpg");

        Hyperlink h1 = new Hyperlink("Imagen 1","imagen1");
        Hyperlink h2 = new Hyperlink("Imagen 2","imagen2");

        RootPanel.get().add(imagen);
        RootPanel.get().add(h1);
        RootPanel.get().add(h2);
    }

    @Override
    public void onValueChange(ValueChangeEvent<String> event) {

        String historia = (event.getValue());

        if (historia.equals("imagen1")){
            //RootPanel.get().add(l);
            imagen.setUrl("http://fluxqubit.files.wordpress.com/2008/03/0074.jpg");
        }
        else if(historia.equals("imagen2")){
            //l.removeFromParent();
            imagen.setUrl("http://blogs.jetbrains.com/pti/images/GWT.png");
        }

    }

}
```

En la aplicación del ejemplo lo que se hace es agregar 2 Hyperlinks llamados h1 y h2 los cuales al hacer clic en cada uno de ellos cambian una imagen que se muestra en el browser. El método onValueChange recibe como parámetro la cadena que se encuentra en el estado actual del objeto History, dicho estado ya se sabe que lo cambia el Hyperlink al dar clic sobre él. Lo que se realiza en el ejemplo es algo muy sencillo, primero se captura el valor del estado y luego se compara dicho valor con el valor de los Hyperlinks y dependiendo de que

Hyperlink fue activado se muestra una u otra imagen. La aplicación se vería algo así:

Al dar clic en el Hyperlink “Imagen 2”, la url cambia y se muestra la otra imagen.



Con los Hyperlinks se puede agregar controles dinámicamente a la aplicación, eliminar paneles o Widgets, en fin permite tener una experiencia de navegación dentro del sitio web. Otra buena forma de navegación dentro de la aplicación

seria utilizar menús pero eso se verá más adelante. Los Hyperlinks funcionan muy bien para navegar dentro de nuestra aplicación, pero si lo que queremos son links hacia otros sitios sería mejor utilizar un link HTML usando el Widgets HTML que ya se vio.

Ya se terminó de dar un recorrido por los Widgets que se denominan básicos, ahora se mostraran los Widgets de formulario

6.4.2 Widgets de Formulario

Los llamados “Widgets de formulario” son aquellos que son usados normalmente en formularios HTML. Sin embargo los formularios HTML envían información al servidor y luego se muestra el resultado refrescando la página. Las aplicaciones Ajax intentan eliminar el refresco de la página y en cambio envían y reciben información del servidor de manera asíncrona. Estos “Widgets de formulario” proporcionados con GWT no necesitan ser incluidos dentro de un formulario HTML son muy flexibles y pueden ser usados de manera muy similar a Widgets de aplicaciones de escritorio. Los Widgets que se verán a continuación son los siguientes:
Button ToggleButton, PushButton, RichTextArea, CheckBox, RadioButton, ListBox, TextBox, PasswordTextBox, TextArea y FileUpload.

6.4.2.1 Button, ToggleButton y PushButton

Se hablará primero del Button. El Widgets Button se comporta como los botones clásicos de cualquier aplicación de escritorio o web, es el Widgets GWT equivalente input de tipo “Button” de un formulario HTM. El

funcionamiento de este Widgets es igual al de cualquier botón, normalmente los botones se usan en un formulario HTML para enviar información al servidor, pero en GWT se puede usar un botón para ejecutar cualquier acción que necesitemos.

Para crear un botón se debe hacer algo como lo que sigue: Button boton = new Button(“Click”)

Para poder utilizar el evento clic se tendrá que agregar al botón un “ClickHandler” que es la interface que se encarga de manejar los eventos clic de algunos Widgets. Esto se realiza así:
Botón.addClickHandler(clickhandler);

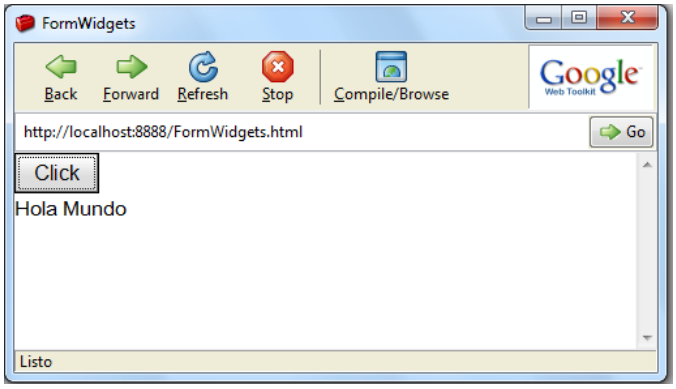
Donde clickhandler es la implementación de dicha interface, la cual se puede hacer utilizando una clase anónima, o implementando la interface en la clase que utiliza el botón, esto es algo conocido para los programadores java.

A continuación se mostrará un ejemplo en el cual se utiliza un botón

```
8  public class FormWidgets implements EntryPoint  {
9
10     public void onModuleLoad() {
11         // TODO Auto-generated method stub
12         Button boton = new Button("Click");
13         boton.addClickHandler(new ClickHandler(){
14             @Override
15             public void onClick(ClickEvent event) {
16                 // TODO Auto-generated method stub
17                 RootPanel.get().add(new Label("Hola Mundo"));
18             }});
19
20         RootPanel.get().add(boton);
21     }
22 }
```

En el ejemplo anterior se creó un botón y se agregó al panel principal, a dicho botón se le adicionó un manejador del evento click el cual se definió con una

clase anónima. El método onClick es el que se ejecutará al invocar al evento click, lo que hace el ejemplo es mostrar un “Hola Mundo” cada vez que se haga clic en el botón.

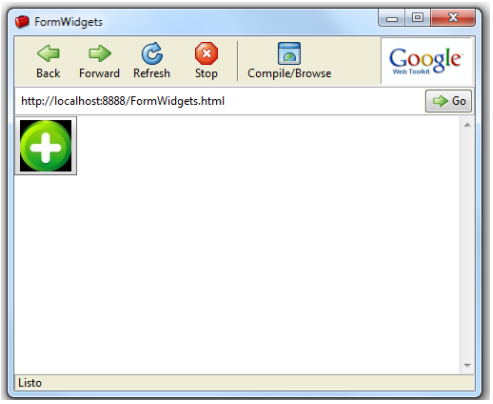


El ToggleButton y PushButton son otros 2 tipos de botones que vienen dentro de la librería de la interface de usuario que pueden ser usados de manera similar a un Widgets Button normal. Un ToggleButton se diferencia de un Button en que cuando se hace clic en estos se mantienen presionados hasta que el usuario hace clic de nuevo. Además el ToggleButton tiene varios constructores que no tiene el Button los cuales aceptan una imagen como parámetro para reemplazar la cara del botón. El PushButton también puede recibir una imagen en el constructor como el ToggleButton. Los eventos de clic y demás es igual a como definimos los del Button anteriormente.

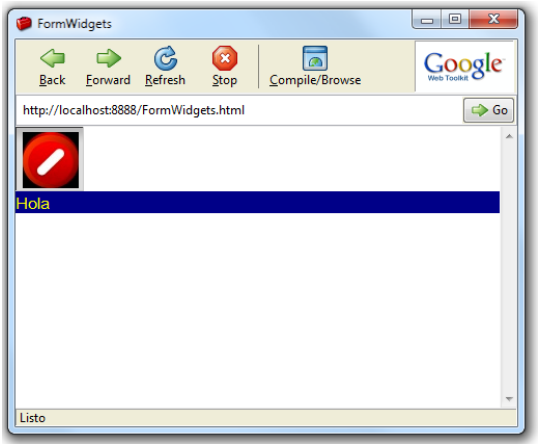
El siguiente ejemplo muestra un uso del ToggleButton

```
8  public class FormWidgets implements EntryPoint , ClickHandler {
9
10     Label lbl = new Label("Hola");
11     public void onModuleLoad() {
12
13         ToggleButton boton = new ToggleButton(new Image("mas.png"),new Image("menos.png"));
14         boton.addClickHandler(this);
15         RootPanel.get().add(boton);
16     }
17     @Override
18     public void onClick(ClickEvent event) {
19
20         if(((ToggleButton)event.getSource()).isDown()){
21             RootPanel.get().add(lbl);
22         }
23         else{
24             lbl.removeFromParent();
25         }
26     }
27 }
28
```

Lo que se ve en el ejemplo es la definición de un ToggleButton llamado botón al cual en el constructor le pasamos como parámetros 2 imágenes, la primera es la que se mostrara en el botón cuando esté en su estado normal y la segunda se mostrara cuando el botón quede presionado debido a que el usuario hizo clic en él. Los ToggleButton pueden definirse con un string que será el texto que aparecerá en el botón, con una imagen que se mostrara en el botón, o con 2 imágenes para ambos estados del ToggleButton.



ToggleButton en su estado natural.



ToggleButton presionado debido al clic del usuario

Un PushButton funciona igual que un Button con la diferencia de que el PushButton soporta algo de personalización en cuanto a estilo, igual que el ToggleButton acepta una imagen para mostrar, además comparte varios estados con el ToggleButton que pueden ser personalizables desde una hoja de estilo CSS los cuales no son soportados por un simple Button. Estos estados son seis y los describimos a continuación.

- ✓ **Button up:** es el estado normal de un PushButton o ToggleButton.

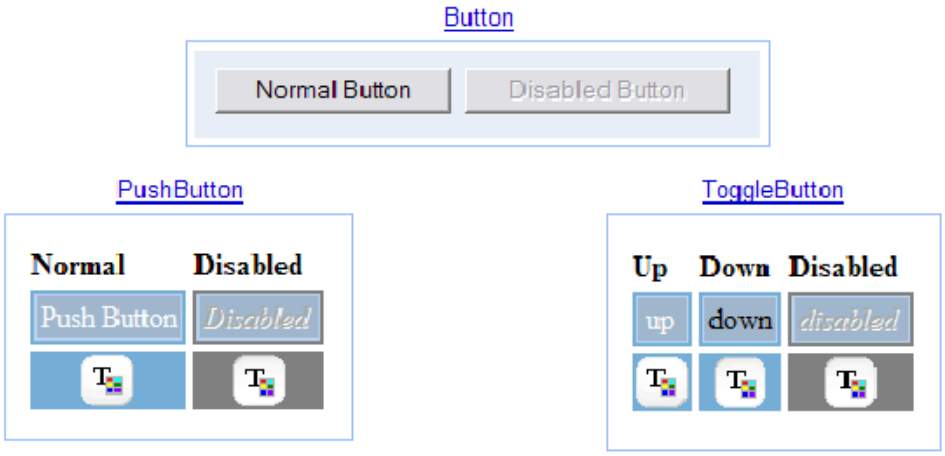
- ✓ **Button down:** el estado cuando el clic del mouse está presionando un PushButton o el estado down de un ToggleButton.

- ✓ **Button up mouse hovering:** es la misma que el Button up con la diferencia que el mouse en esos momentos se encuentra encima del PushButton o ToggleButton.

- ✓ **Button up disable:** igual que el Button up pero el botón en ese momento se encuentra deshabilitado por el método setDisable del botón.

- ✓ **Button down mouse hovering:** mismo estado que el Button down pero el mouse se encuentra encima del botón en esos momentos. Para el caso del PushButton es equivalente al Button down ya que el para que el PushButton este en estado down el mouse debe estar presionando el botón.

- ✓ **Button down disable:** este estado nada más funciona para el ToggleButton ya que es el único que puede estar deshabilitado y en el estado down al mismo tiempo.



Estado de los botones. Tomado de Manning GWT in Action Easy Ajax with the Google Web Toolkit Jun 2007

6.4.2.2 CheckBox

El Widgets CheckBox es la implementación del CheckBox estándar de HTML. En una aplicación web normal este Widgets usualmente se encontraría en una forma y su valor sería enviado al servidor cuando el formulario es enviado. En aplicaciones Ajax no se tiene estas restricciones. Este Widgets soporta eventos de cambio de foco y eventos de clic. Cuenta con un estado que nos dice si el

Widgets esta checkeado o no, que es la función getValue la cual dirá *true* si se encuentra checkeado y *false* obviamente lo contrario. A continuación se mostrara un ejemplo del uso del CheckBox haciendo una lista de acciones pendientes.

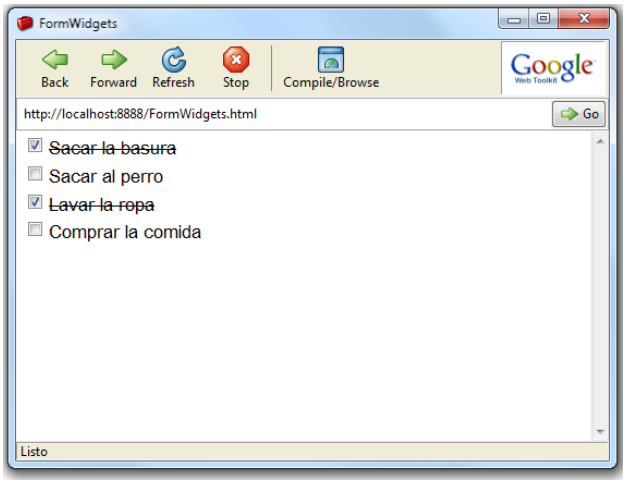
```
8  public class FormWidgets implements EntryPoint, ClickHandler  {
9
10     public void onModuleLoad() {
11
12         VerticalPanel lista = new VerticalPanel();
13
14         CheckBox item1 = new CheckBox("Sacar la basura");
15         item1.addClickHandler(this);
16
17         CheckBox item2 = new CheckBox("Sacar al perro");
18         item2.addClickHandler(this);
19
20         CheckBox item3 = new CheckBox("Lavar la ropa");
21         item3.addClickHandler(this);
22
23         CheckBox item4 = new CheckBox("Comprar la comida");
24         item4.addClickHandler(this);
25
26         lista.add(item1);
27         lista.add(item2);
28         lista.add(item3);
29         lista.add(item4);
30
30
31         RootPanel.get().add(lista);
32     }
33     @Override
34     public void onClick(ClickEvent event) {
35
36         if(((CheckBox)event.getSource()).getValue()){
37             ((CheckBox)event.getSource()).setStyleName("gwt-CheckBox-checked");
38         }
39         else{
40             ((CheckBox)event.getSource()).setStyleName("gwt-CheckBox");
41         }
42     }
43 }
```

Lo que se hace en este ejemplo es simplemente definir 4 CheckBox y agregarlos a un VerticalPanel (luego se verá cómo funciona) para que los CheckBox se agreguen uno debajo del otro. Luego como se hizo con los botones les agregamos un ClickHandler para manejar el evento cuando el usuario checkea cada ítem. En el evento onClick lo único que se hace es que

cuando el CheckBox se encuentre chequeado se le cambia el nombre del estilo CSS en el cual definimos esta propiedad:

.gwt-CheckBox-checked{text-decoration: *line-through*;}

Quedando la aplicación como se muestra a continuación:



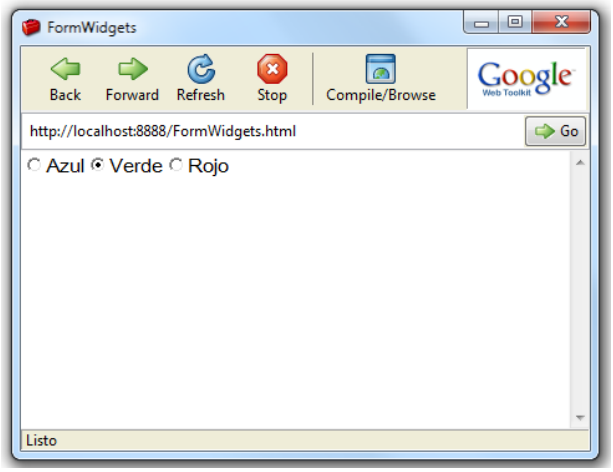
6.4.2.3 RadioButton

Este Widgets es igual de sencillo y parecido al CheckBox que se acaba de ver, con la diferencia de que varios RadioButton asociados al mismo grupo no pueden estar seleccionados a la vez. Este Widgets es útil cuando se necesita que el usuario elija un solo ítem entre un grupo de la misma categoría. Como el RadioButton hereda de CheckBox tiene la misma funcionalidad y los mismos métodos de getValue y el mismo manejo de eventos.

Un ejemplo de RadioButton

```
8  public class FormWidgets implements EntryPoint  {
9
10     public void onModuleLoad() {
11
12         RadioButton item1 = new RadioButton("Colores","Azul");
13         RadioButton item2 = new RadioButton("Colores","Verde");
14         RadioButton item3 = new RadioButton("Colores","Rojo");
15
16         RootPanel.get().add(item1);
17         RootPanel.get().add(item2);
18         RootPanel.get().add(item3);
19     }
20
21 }
```

El ejemplo muestra como se define 3 RadioButtons, el primer parámetro del constructor es el grupo al que pertenecen, el segundo es el texto que se mostrara.

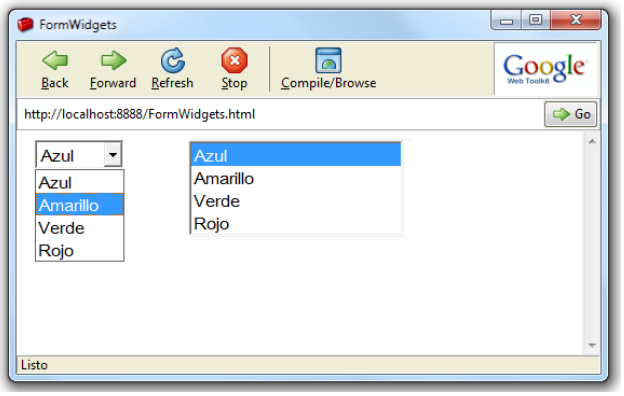


6.4.2.4 ListBox

Un ListBox es un Widgets parecido al RadioButton, es decir muestra al usuario varias opciones y lo pone a elegir una sola dentro de las múltiples opciones. La diferencia con el RadioButton es la forma de mostrar las opciones. El ListBox muestra las opciones de 2 maneras, la primera como un combo o lista

desplegable, y la segunda como un cuadro donde muestran todas las opciones y el usuario selecciona la que desee.

Esta es una muestra de cómo se ven ambas formas de mostrar el ListBox.



La primera forma muestra al ListBox como una lista desplegable que presenta solo la opción seleccionada, cuando el usuario hace clic en la flecha se despliega las demás opciones como se ve en la figura. La forma de crear este ListBox es la siguiente

```
ListBox lb = new ListBox();
lb.addItem("Azul");
lb.addItem("Amarillo");
lb.addItem("Verde");
lb.addItem("Rojo");

RootPanel.get().add(lb);
```

Instanciamos el ListBox y luego con el método addItem se agrega las opciones al ListBox.

La segunda forma para mostrar un ListBox consiste en una lista de las opciones, donde la opción que se encuentra seleccionada se encuentra resaltada (como se vio en la figura anterior). La forma de definir un ListBox de esta manera es igual a la que ya se mostró con una línea adicional donde se define el número de ítems que se desean mostrar usando el método setVisibleItemCount, si se define un número de ítems menor al total de las opciones del ListBox se agregara un scroll que permite ver las demás opciones.

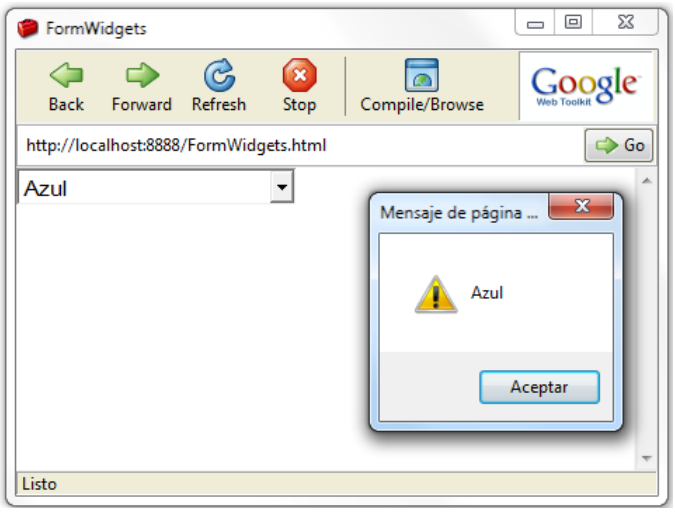
Como los Widgets de esta sección el ListBox maneja los eventos de foco y el cambio de valor del Widgets o “change event”, el cual ocurre cuando el usuario cambia de opción en el ListBox.

Esto se hace agregándole un “ChangeHandler” usando el método addChangeHandler. Esto funciona de la misma manera que los eventos de clic que hemos visto.

A continuación se mostrara un ejemplo de esto usando el ListBox que se creó anteriormente, donde el browser mostrará un aviso cada vez que se seleccione una opción del ListBox diciendo que opción se selecciono. El código sería algo así:

```
11 public class FormWidgets implements EntryPoint  {
12
13     public void onModuleLoad() {
14
15         final ListBox lb = new ListBox();
16         lb.addItem("Azul");
17         lb.addItem("Amarillo");
18         lb.addItem("Verde");
19         lb.addItem("Rojo");
20         lb.setSize("200px", "100px");
21
22         RootPanel.get().add(lb);
23
24         lb.addChangeHandler(new ChangeHandler(){
25
26             @Override
27             public void onChange(ChangeEvent event) {
28                 // TODO Auto-generated method stub
29                 int item = lb.getSelectedIndex();
30                 Window.alert(lb.getItemText(item));
31             }
32         });
33     }
34 }
```

Se observa en el código que es el mismo del ejemplo anterior con la diferencia de que acá se define el tamaño del ListBox con el método setSize, además se usa el método addChangeHandler con el que se define el evento del ListBox para que ejecute lo que se explico anteriormente. La aplicación se vería algo así:



6.4.2.5 TextBox

El TextBox es la equivalencia en GWT de la etiqueta input de tipo input de HTML, la cual es la forma estándar de capturar texto del usuario. Es un Widgets muy sencillo y fácil de usar.

Se puede crear un TextBox de la siguiente manera:

```
TextBox nombre = new TextBox();
```

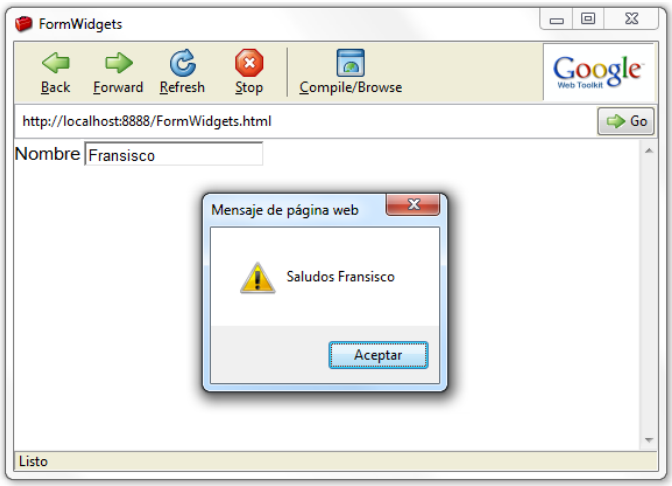
Como el ListBox soporta eventos de clic, y eventos por cambios en su valor y también un evento que se dispara cuando el usuario presiona una tecla en el llamado KeyPressEvent el cual se puede manejar con un KeyPressHandler.

A continuación se muestra un ejemplo de una aplicación donde se pide el nombre al usuario y al presionar la tecla enter el browser muestra una ventana con el nombre que digito el usuario, el código seria este:

```
14 public class FormWidgets implements EntryPoint, KeyPressHandler  {
15
16     public void onModuleLoad() {
17
18         Label lblnombre = new Label("Nombre");
19         TextBox txtnombre = new TextBox();
20         txtnombre.addKeyPressHandler(this);
21
22         RootPanel.get().add(lblnombre);
23         RootPanel.get().add(txtnombre);
24
25     }
26
27     @Override
28     public void onKeyPress(KeyPressEvent event) {
29         // TODO Auto-generated method stub
30
31         char a = event.getCharCode();
32         String nombre = ((TextBox)event.getSource()).getText();
33
34         if (a == KeyCodes.KEY_ENTER){
35
36             Window.alert("Saludos "+nombre);
37
38         }
39     }
40 }
```

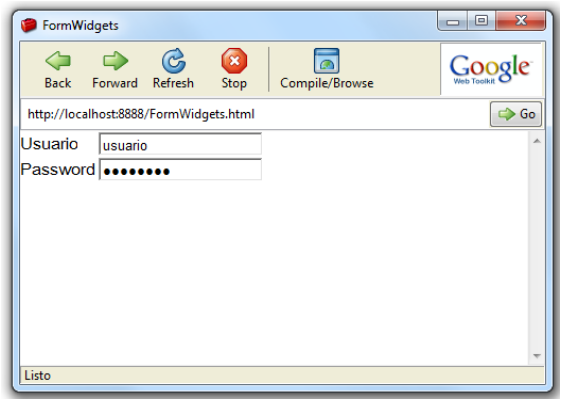
En este código se puede observar cómo se le agrega un “KeyPressHandler” el cual es la misma clase que se implementó de la interface KeyPressHandler. En

el método onKeyPress se observa la acción que se ejecuta al dispararse el evento, lo que hace es que al presionar la tecla ENTER en el TextBox se manda un mensaje. La aplicación quedaría algo como esto:



6.4.2.6 PasswordTextBox

Este Widgets funciona exactamente igual que el TextBox que se acaba de ver con la diferencia que su contenido está oculto del usuario para proteger datos como contraseñas. Normalmente este Widgets se usa en un formulario de login. A continuación se muestra como se vería en una aplicación el PasswordTextBox.



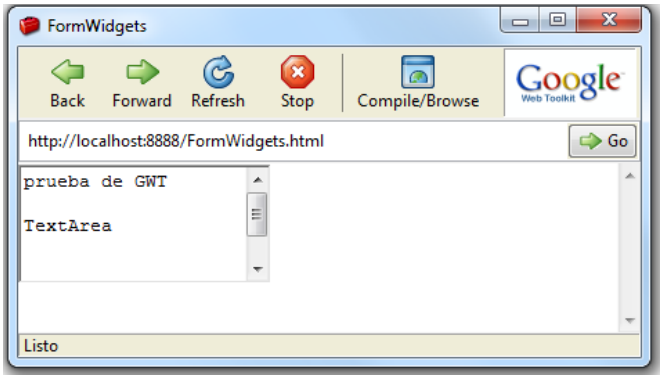
6.4.2.7 TextArea

El TextArea es otro Widgets que permite capturar texto del usuario, es parecido al TextBox que ya se vio con la diferencia que permite textos de múltiples líneas mientras que el TextBox nada más nos permite una sola línea.
A continuación se verá un ejemplo del uso de un TextArea.

```
14 public class FormWidgets implements EntryPoint  {
15
16     public void onModuleLoad() {
17
18         TextArea blog = new TextArea();
19
20         blog.setCharacterWidth(20);
21         blog.setVisibleLines(5);
22
23         RootPanel.get().add(blog);
24
25     }
26 }
```

En el código anterior se observa cómo se define un TextArea llamado blog, luego se utiliza 2 métodos de este Widgets. El método setCharacterWidth permite fijar el número de caracteres que recibe el TextArea a lo ancho. Con el método setVisibleLines se decide cuantas líneas serán visibles en el TextBox, si el usuario ingresa más líneas de las que se define el Widgets mostrara un scroll con el que se puede ver el texto.

El ejemplo luciría algo así:



El TextArea soporta los mismos eventos que el TextBox y se puede usar de la manera que ya se mostró con el TextBox.

6.4.2.8 RichTextArea

Los anteriores Widgets de captura de texto permitían usar texto simple, es decir texto plano sin formato, pero hay caso en los que se quiere usar texto en negrilla o de algún color en especial u otros tipos de formato. Para esto se utiliza el Widgets RichTextArea, que como su nombre lo indica es un TextArea pero que maneja texto enriquecido podríamos decir, se maneja de manera similar al TextArea, tiene los métodos getText y setText que permiten recuperar o asignar texto plano, pero además tiene 2 métodos adicionales que son getHTML y setHTML que permiten usar texto en formato HTML para así poder utilizar texto formateado. Este es un ejemplo de un RichTextArea:

El RichTextArea es solo el cuadro de texto que se ve debajo de la barra de herramientas, eso ya es un control aparte, como se nota hay parte del texto que se encuentra en negrilla y parte subrayado, en el ejemplo el RichTextArea junto con la barra de herramientas es similar a un procesador de texto.

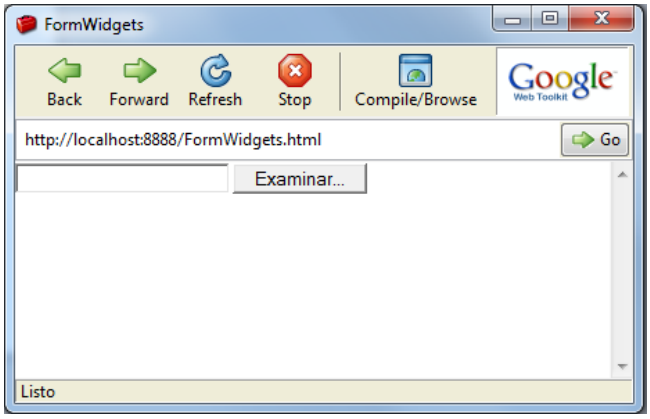
6.4.2.9 FileUpload

El FileUpload Widgets es el que da la posibilidad de permitirle al usuario seleccionar un archivo de su PC. Este Widgets solo funciona del lado del cliente por lo que permite al usuario seleccionar un archivo mas no enviarlo al servidor, ya eso depende de nosotros implementar la funcionalidad que no nos da el Widgets. Lo máximo que se puede obtener de este Widgets es el nombre del archivo que seleccionó el usuario con el método getFileName.

La mayoría de los navegadores tienen muchas restricciones de seguridad en cuanto al envió de archivos se refiere, por ejemplo nada más permiten enviar el archivo seleccionado por el usuario y no otros que uno haya intentado enviar mediante programación, además el archivo solo es enviado al servidor por el browser cuando el formulario es enviado. Esto último implicaría un refresco de la página, pero la idea de GWT es que todo se Ajax por lo que brinda un panel llamado FormPanel, cuyo funcionamiento se verá más adelante, que permite incluir el FileUpload y enviar archivos al servidor de manera asíncrona y

cumpliendo con la metodología Ajax. Más adelante se observará la manera de enviar los archivos al servidor.

Un FileUpload se vería más o menos así:



6.4.3 Widgets Complejos

Cuando los navegadores empezaron a aceptar JavaScript y HTML dinámico, se empezó a ver interfaces de usuario compuestas por objetos los cuales no tenían una etiqueta HTML equivalente. Estos objetos fueron creados por la composición de etiquetas y el manejo de los eventos fue realizado con JavaScript. Estos Widgets se pueden crear con GWT también, además que ya vienen algunos con el framework, estos son a los que llamamos Widgets complejos. En esta categoría se encontrara al Tree y al MenuBar

6.4.3.1 Tree

El Tree es parecido a los controles Tree de aplicaciones de escritorio, permite mostrar una vista jerárquica de datos que se puede contraer y colapsar. Los

elementos que se muestran en un Tree pueden ser TreeItem, a los que se le pueden llamar ramas del árbol. Se puede agregar TreeItem a los TreeItem para agregar subniveles al árbol. También se puede agregar una cadena u otros Widgets al árbol

Instanciar un Tree Widgets y agregarlo a un panel es igual de sencillo que los demás Widgets:

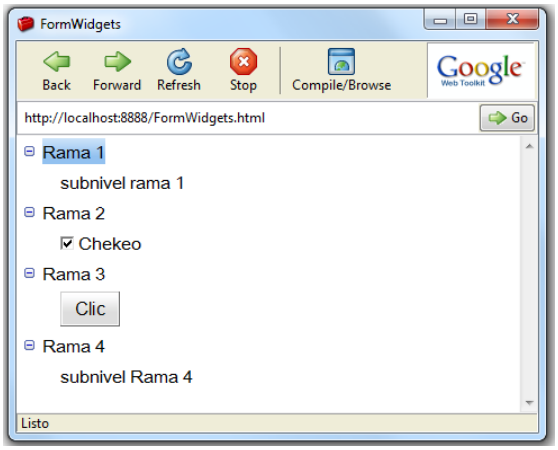
```
Tree arbol = new Tree();
RootPanel.get().add(arbol);
```

Mencionamos anteriormente que se puede agregar TreeItem a nuestro Tree, pero ¿qué son los TreeItem? los TreeItem, como se dijo antes, pueden considerarse como Widgets que tienen el papel de ramas en el árbol. Se le puede agregar texto, imágenes u otros Widgets a cada TreeItem, además que se puede agregar subniveles con más TreeItems.

A continuación se muestra un ejemplo agregando los ítems del Tree manualmente, pero se puede hacer esto dinámicamente con datos que se traigan de una base de datos que se encuentra en el servidor.

```
18      public void onModuleLoad() {
19
20          Tree arbol = new Tree();
21          TreeItem rama1 = new TreeItem("Rama 1");
22          rama1.addItem("subnivel rama 1");
23
24          TreeItem rama2 = new TreeItem("Rama 2");
25          rama2.addItem(new CheckBox("Chekeo"));
26
27          TreeItem rama3 = new TreeItem("Rama 3");
28          rama3.addItem(new Button("Clic"));
29
30          TreeItem rama4 = new TreeItem("Rama 4");
31          rama4.addItem(new TreeItem("subnivel Rama 4"));
32
33          arbol.addItem(rama1);
34          arbol.addItem(rama2);
35          arbol.addItem(rama3);
36          arbol.addItem(rama4);
37
38          RootPanel.get().add(arbol);
39      }
40  }
```

En el ejemplo se observa que en la línea 20 se instancia el Tree, luego se define 4 TreeItem cada uno de los cuales se le agrega un ítem ya sea una cadena, un Widgets u otro TreeItem. El resultado de ese ejemplo sería el siguiente:



Si se está usando el Tree que muestra diferentes vistas u opciones dependiendo del ítem que se seleccione o alguna operación que tenga que ver con selección de ítems se necesitara que el Widgets diga cuando se hizo una

selección de un determinado ítem y cual ítem es el que se encuentra seleccionado. Para esto el Tree soporta varios eventos que son SelectionHandler (para manejar el evento de selección de ítems del árbol), OpenHandler (se dispara cuando se abre un ítem o rama del árbol, es decir cuando se colapsa una opción) y el CloseHandler (que funciona cuando se cierra o contrae un ítem o rama del árbol).

6.4.3.2 MenuBar

Otro Widgets que entra en la categoría de Widgets complejos es este llamado MenuBar. Como su nombre lo supone, es la equivalencia a los menús de las aplicaciones de escritorio. Tiene cierta similitud con el Tree que se mostró, con la diferencia que solo se puede ver una subcategoría al tiempo. Normalmente se usa este Widgets para mostrar una lista de opciones o aplicaciones dentro de la aplicación.

Para instanciar un MenuBar solo falta una línea:

```
MenuBar menu = new MenuBar();
```

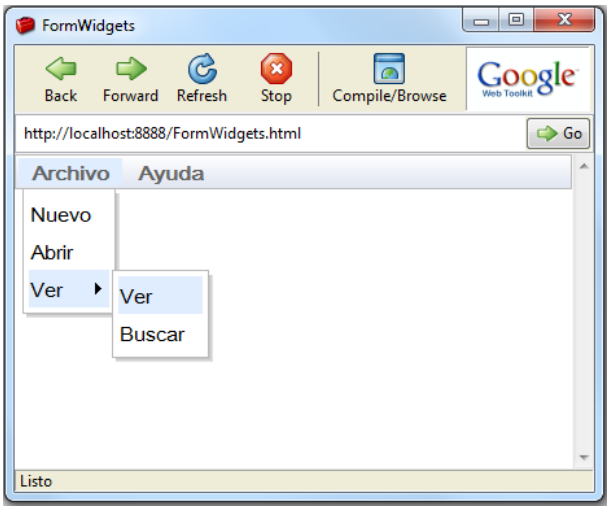
Si en el constructor no recibe ningún parámetro o el valor booleano false, el menú se muestra horizontalmente, mientras que si se inicializa con true se muestra vertical. Se maneja de la misma forma que el Tree que ya se vio, para agregar un ítem al menú se usa el método addItem. Con la diferencia que se puede usar este método de varias maneras.

Se puede pasar como parámetros una cadena con el texto que se muestra en el ítem del menú y un objeto de tipo Command, el cual ejecutará un código cada

vez que se selecciona ese ítem. También se puede en vez del objeto tipo Command, pasarle como parámetro otro MenuBar para llamar a un submenú. Además también se puede una clase llama MenuItem que funciona de manera similar al TreeItem que ya se vio. Ahora se observa un ejemplo de un menú sencillo.

```
18 public class FormWidgets implements EntryPoint  {
19
20     public void onModuleLoad() {
21
22         Command cmd = new Command() {
23             public void execute() {
24             //code to handle the command goes here
25
26             }
27         };
28
29         MenuBar menu = new MenuBar();
30
31         MenuBar ver = new MenuBar(true);
32         ver.addItem("Ver",cmd);
33         ver.addItem("Buscar", cmd);
34
35         MenuBar archivo = new MenuBar(true);
36         archivo.addItem("Nuevo", cmd);
37         archivo.addItem("Abrir", cmd);
38         archivo.addItem("Ver",ver);
39
40         MenuBar ayuda   = new MenuBar(true);
41         ayuda.addItem("Acerca de ", cmd);
42
43         menu.addItem("Archivo",archivo);
44         menu.addItem("Ayuda",ayuda);
45
46         RootPanel.get().add(menu);
47     }
48 }
```

En este ejemplo se define un objeto de tipo Command vacío, es decir con el método execute que es el que se ejecuta cuando se selecciona un ítem del menú, además se define el menú principal llamado “menú". Luego se define los submenús que se le van a agregar al menú principal, se observa que los submenús se instancian con el parámetro true para que se despliegue de manera vertical. Finalmente se agrega el menú principal al panel. El resultado sería la siguiente aplicación:

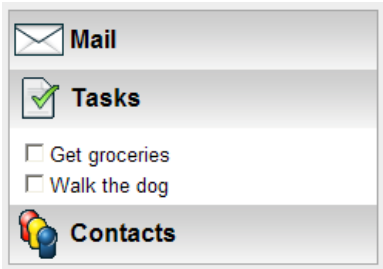


6.5 QUÉ ES UN PANEL

Los paneles están diseñados para contener Widgets dentro y son usados para determinar como la interface de usuario será distribuida sobre el navegador utilizando códigos en java del lado del cliente.

6.6 PANELES DE GWT

6.6.1 StackPanel

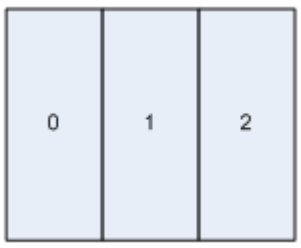


Se encarga de organizar los elementos en una determinada dirección. Mediante el uso de sus propiedades el contenido puede fluir de forma horizontalmente o verticalmente (configuración por defecto).

Implementa el IScrollInfo que es una interface lógica de apoyo a desplazarse.

```
<table class="gwt-StackPanel" cell-spacing="0" cell-padding="0">
  <tbody>

    <tr>
      <td class="gwt-StackPanelItem" height="1px">text/html</td>
    </tr>
    <tr>
      <td height="100%" valign="top">
        content -- a widget
      </td>
    </tr>
  </tbody>
</table>
```

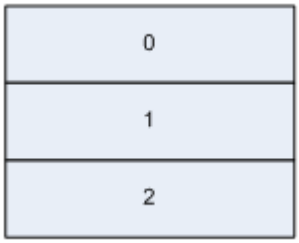
6.6.2 HorizontalPanel



Es un cuadro basado en grupos alineados horizontalmente con celdas y van organizados de izquierda a derecha.

```
<table cell-spacing="0" cell-padding="0">
  <tbody>
    <tr>
      <td style="display: static; vertical-align: top;" align="left">Item 1</td>
      <td style="display: static; vertical-align: top;" align="left">Item 2</td>
    </tr>
  </tbody>
</table>
```

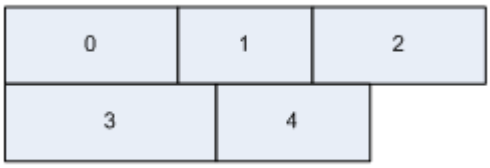
6.6.3 VerticalPanel



Es un panel que establece todos sus módulos en una columna vertical.

```
<table cell-spacing="0" cell-padding="0">
  <tbody>
    <tr><td style="display: static; vertical-align: top;" align="left">Item
1</td></tr>
    <tr><td style="display: static; vertical-align: top;" align="left">Item
2</td></tr>
  </tbody>
</table>
```

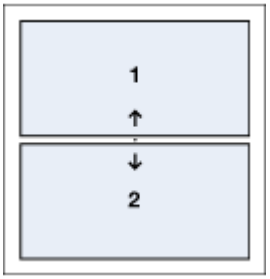
6.6.4 FlowPanel



Representa un DIV con el atributo display en inline.

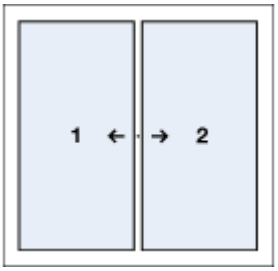
```
<div style="display: inline;">content</div>
```

6.6.5 VerticalSplitPanel



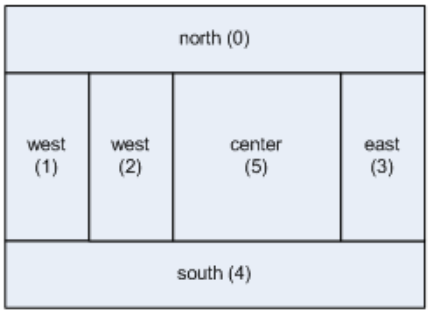
Es un grupo que se encarga de organizar dos Widgets en una sola columna vertical permitiendo al usuario cambiar interactivamente la proporción de la altura propuesta para cada uno de los dos Widgets. Los Widgets dentro de este VerticalSplitPanel pueden ser decorados con barras de desplazamiento si es necesario.

6.6.6 HorizontalSplitPanel



Es un grupo el cual está encargado de organizar dos Widgets en una sola fila horizontal permitiendo al usuario cambiar de forma interactiva la proporción de la anchura propuesta para cada uno de los dos Widgets. Los Widgets dentro de este HorizontalSplitPanel pueden ser decorados con barras de desplazamiento si es necesario.

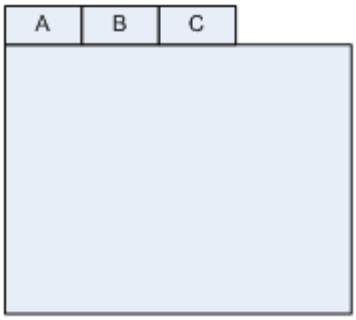
6.6.7 DockPanel



Es un grupo que establece los Widgets en forma acoplada en sus bordes exteriores permitiendo al último Widgets tomar el espacio del centro. Por defecto tiene cell-spacing y cell-padding en cero.

```
<table cell-spacing="0" cell-padding="0">
  <tbody>
  </tbody>
</table>
```

6.6.8 TabPanel

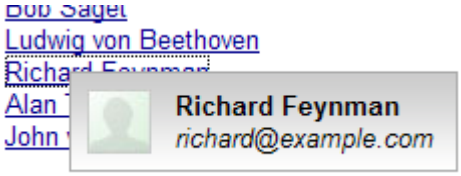


Es un grupo que representa a un conjunto de paginas con pestañas en la cual cada una posee un elemento. Cada una de las pestañas puede contener HTML arbitrario. Estos elementos no son un grupo de por si, sino un compuesto que contiene un TabBar y un DeckPanel.

```
<table class="gwt-TabPanel" cell-spacing="0" cell-padding="0">
  <tbody>
    <tr>
      <td>

        <table class="gwt-TabBar" style="width: 100%;" cell-spacing="0" cell-
padding="0">
          <tbody>
            <tr>
```

```
            <td class="gwt-TabBarFirst" style="height: 100%;"><div
class="gwt-HTML" style="height: 100%;">&amp;nbsp;</div></td>

            <td class="gwt-TabBarRest" style="width: 100%;"><div class="gwt-
HTML" style="height: 100%;">&amp;nbsp;</div></td>
          </tr>
        </tbody>
      </table>
    </td>
  </tr>
  <tr>
    <td>
      <div class="gwt-TabPanelBottom">

  </div>
    </td>
  </tr>
 </tbody>
</table>
```

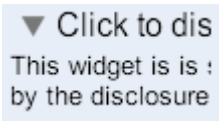
6.6.9 PopupPanel



Es un grupo que puede emerger con respecto a otros Widgets. La anchura y la altura del PopupPanel no se puede establecer explícitamente ya que se

encuentran definidos por el PopupPanel del Widgets. Un PopupPanel no debería añadirse a otros grupos, más bien, debería ser expuesto y ocultos utilizando los métodos show() y hide().

6.6.10 DisclosurePanel



Dicho panel consta de una cabecera y de un contenido. DisclosurePanel revela su contenido al momento que un usuario hace clic sobre su cabecera.

7 CREAR WIDGETS COMPUESTOS

Ya se ha observado el uso de varios de los Widgets que provee GWT en su framework. Ahora se verá lo que son los Widgets compuestos o “composite Widgets”. Los “composite Widgets” son elementos potentes de GWT los cuales son Widgets creados combinando los Widgets y paneles que ya se ha visto en capítulos anteriores. Esto es muy útil si se quiere crear controles de usuario personalizados y reutilizables. Esto puede resultar familiar para aquellos desarrolladores que crean este tipo de controles reutilizables en java o .Net. La idea es la misma.

Se puede considerar a estos Widgets compuestos como mini aplicaciones GWT, ya que están compuestas por Widgets y proporcionan una funcionalidad dada por el creador del control.

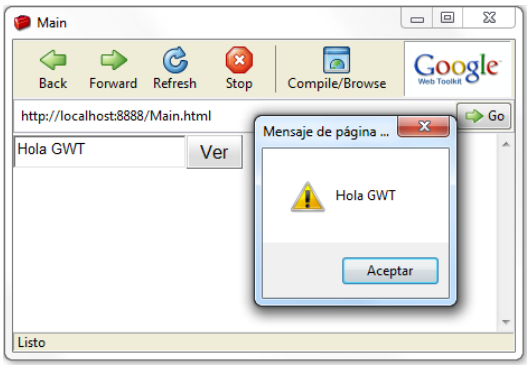
Para crear un Widgets compuesto lo que se hace es crear una clase y hacer que herede de “Composite”, la implementación de interfaces depende de los eventos que se vaya a manejar en el Widgets. Luego como atributos de la clase agregar los Widgets que se vaya a utilizar.

A continuación se observará un ejemplo de un Widgets que estará compuesto de un TextBox y un Button, y que al dar clic en el botón mostrará lo escrito en el TextBox en un mensaje del browser.

```
8  public class WidgetCompuesto extends Composite implements ClickHandler {
9
10     private TextBox texto = new TextBox();
11     private Button  btnver = new Button("Ver");
12
13     public WidgetCompuesto(){
14
15         texto.setHeight("30px");
16
17         btnver.addClickHandler(this);
18         btnver.setHeight("30px");
19
20         FlowPanel panel = new FlowPanel();
21         panel.add(texto);
22         panel.add(btnver);
23
24         this.initWidget(panel);
25     }
26
27     @Override
28     public void onClick(ClickEvent event) {
29         // TODO Auto-generated method stub
30         Window.alert(texto.getText());
31     }
32
33 }
```

En el ejemplo vemos que el Widgets hereda de Composite e implementa la interface ClickHandler para manejar el evento del botón. Como atributos de la clase se tiene un TextBox y un Button a los cuales se les asigna un valor al heigh, aunque eso se puede hacer por CSS. Ambos Widgets se agregan a un FlowPanel y finalmente con el método initWidget se le dice que el Widgets que va a mostrar el composite será el panel que contiene a los demás objetos. El manejo del evento clic del botón ya se ha visto así que no es nada nuevo para nosotros.

La aplicación se vería algo así:



Crear un Widgets compuesto es como si se realizara una mini aplicación, así que se debe resolver varias preguntas como las que se hace al construir otras aplicaciones: que Widgets usar y como se les quiere mostrar, que paneles sirven para el diseño que se quiere montar, que interfaces debe implementar el Widgets, el manejo de eventos que se debe utilizar. Para esto se aconseja una serie de pasos para armar los Widgets compuestos los cuales son los siguientes:

- ✓ **Identificar los Widgets:** como se sabe los Widgets compuestos son hechos de varios Widgets los cuales pueden ser los estándares de GWT u otro Widgets compuesto, por eso es bueno que antes de construir el Widgets se identifique los Widgets que se va a incluir en él.

- ✓ **Elegir el diseño:** después de elegir los Widgets que se usarán se debe definir el diseño a usar, es decir la disposición de los Widgets dentro del control.

- ✓ **Identificar interfaces:** esto se refiere a las interfaces que se va a implementar en el Widgets para el manejo de los eventos, depende de las necesidades y de las funcionalidades que se vaya a implementar. Se implementará las interfaces que sean útiles para el propósito.

- ✓ **Construcción del Widgets:** como se imaginaran por el nombre, es la construcción del Widgets como tal, ya que se ha definido la estructura es hora de implementar dicha estructura. Este paso tiene sub-etapas que son:

    - ➢ Implementar las interfaces identificadas.
    - ➢ Implementar métodos adicionales a los requeridos por las interfaces que se implementan.
    - ➢ Crear la estructura del Widgets y establecer el manejo de eventos.
    - ➢ Implementar el código que manejara los eventos esperados.

- ✓ **Establecer los estilos:** en esta etapa más que todo lo que se hace es establecer una convención para los nombres de estilo de cada componente Widgets o panel.

- ✓ **Pruebas:** como todo desarrollo de software la etapa final es la prueba del Widgets. la idea es que el Widgets se comporte bien y haga lo que tiene que hacer para evitar sorpresas más tardes.

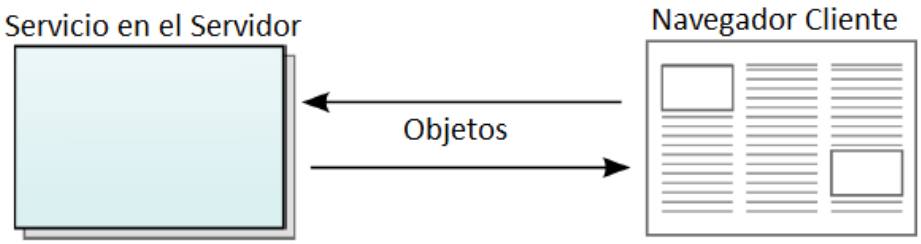
8 RPC

Hasta ahora lo que se ha visto es la construcción de interfaces gráficas, manejo de eventos pero todo esto del lado del cliente. La mayoría de las aplicaciones web necesitan la comunicación con el servidor, ya sea para recibir o enviar información. Para esto GWT implemento un método de comunicación con el servidor llamado GWT-RPC, para los que han trabajado en java esto les parecerá algo conocido. Este mecanismo permite ejecutar métodos que se encuentran en otras computadoras y traer el resultado. Ya se ha descrito algunas cosas sobre RPC en el capítulo 1.

Se puede decir que hay 3 partes o piezas que se debe ensamblar para poner a funcionar una aplicación que use RPC. La primera es el servicio que corre en el servidor, la segunda es el cliente en el browser que llama al servicio, y la tercera son los objetos que son transportados entre el cliente y el servidor. Tanto el cliente como el servidor tienen la habilidad de serializar y deserializar datos, por lo que los objetos pueden ser enviados entre ambos como texto ordinario.

El código del lado del servidor que es invocado desde el cliente es frecuentemente llamado “servicio”, por lo que el “llamar procedimientos remotos” es comúnmente llamado como invocación de servicios. No debe confundirse estos llamados a RPC con los web-services, estos servicios del RPC de GWT no es lo mismo que el SOAP (Simple Object Access Protocol).

En la siguiente imagen se muestra un bosquejo de una comunicación entre un servidor y el cliente.



8.1 Estructura RPC

A continuación se muestra las partes básicas requeridas para invocar un servicio. Cada servicio tiene una pequeña familia de interfaces y clases. Algunas de esas clases, como el servicio Proxy, son generados automáticamente y generalmente nunca te percatarás de que existen. El patrón de clases es idéntico para todos los servicios que implementas, por lo que es una buena idea familiarizarse un poco con los términos y el propósito de cada capa, en el procesamiento de llamadas al servidor. Si estás familiarizado con el mecanismo tradicional de RPC, probablemente ya conocerás la mayoría de la terminología. Se verá estas partes más en detalle luego.

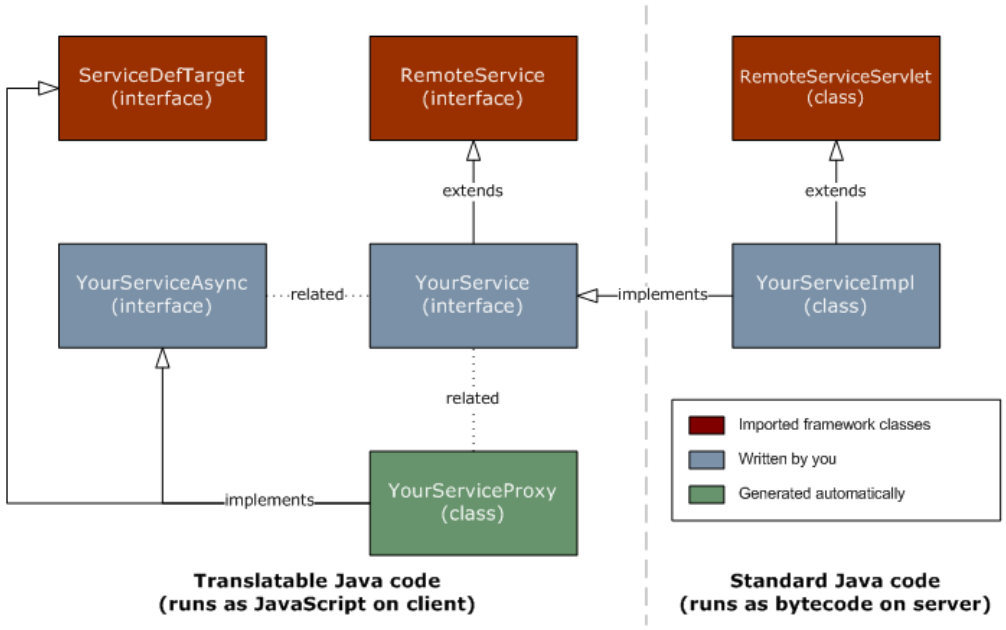


Grafico tomado de la pagina oficial de google web toolkit

Como ya se dijo antes las 3 partes básicas del mecanismo de RPC son el servicio, el cliente y los objetos que viajan entre ellos. Se va a explicar estas partes, comenzando por los objetos y explicando un poco de la serialización.

Se empezará explicando con los objetos porque los datos son los que le dan vida al GWT-RPC. Como ya se dijo RPC consiste en llamar métodos que se encuentran en un servicio en el servidor el cual enviara el resultado de la ejecución del método hacia el cliente. Como cualquier método de Java se puede pasar parámetros a los métodos los cuales pueden ser tipos primitivos u objetos. Ya se mencionó anteriormente que los objetos se pasan serializados entre el cliente y el servicio así que hay límites entre los tipos de datos que se pueden enviar, los cuales están en la siguiente tabla:

Tipos primitivos	boolean, byte, char, double, float , int, long , shot
Contenedores de los tipos	Boolean, Byte, Character, Double, Integer,

primitivos	Float…
Subconjuntos de objetos java	Solo ArrayList, Date, HashMap, HashSet, String, Vector
Clases definidas por el usuario	Cualquier clase que implemente IsSerializable
Arreglos	Arreglos de cualquier tipo serializable

Los tipos de datos propios de java son limitados y son los incluidos en la librería de GWT que emula la JRE de java.

Con respecto a los otros tipos de datos definidos por el usuario se deben implementar la interface IsSerializable, la cual no tiene métodos que implementar y que se utilizan para que GWT sepa que ese objeto podrá ser serializado. Las clases implementaran la interface com.google.gwt.user.client.rpc.IsSerializable, todos los atributos que no tengan el modificador transient son serializables, y además la clase tendrá un constructor sin parámetros o no tendrá constructor.

El próximo paso para usar RPC es definir e implementar el servicio que será ejecutado en el servidor. Esto consistirá en una interface de java, la cual describiría el servicio y la implementación del mismo.

Para definir el servicio se necesita crear una interface y hacerla heredar de la interface que provee GWT llamada RemoteService. Esto es tan fácil como suena. Sería algo como esto:

```
 9 public interface Servicio extends RemoteService {
10
11     public String getFecha();
12
13 }
```

Como se ve es tan sencillo como definir el nombre de la interface, heredar de la interface com.google.gwt.user.client.rpc.RemoteService, definir los métodos que se desea en el servicio teniendo en cuenta que los parámetros y los valores de retorno deben ser serializables. Esta interface se debe colocar en el packete “cliente” de el modulo. Aunque si se usa algún IDE de desarrollo como eclipse con algún plugin este se encargara de todo esto, ya veremos después como hacer esto con eclipse.

Ya se ha creado la interface para el servicio, ahora es momento de implementar sus métodos. Se realiza esto creando un servlet que herede de RemoteServiceServlet y que implemente la interface de servicio. El código sería algo como esto:

```
6  public class ServicioImpl extends RemoteServiceServlet implements Servicio {
7
8      @Override
9      public String getFecha() {
.0         // TODO Auto-generated method stub
.1         return "01/01/2010";
.2     }
.3
.4 }
```

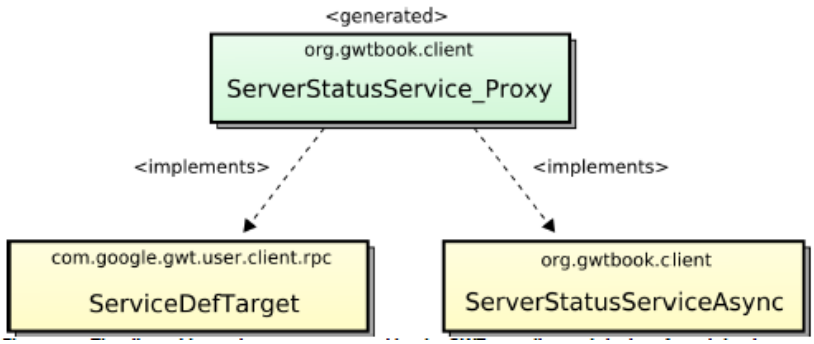
No hay mucho que explicar, en el servlet se implementara los métodos del servicio que se desea crear. Se observa que el nombre del servlet es el mismo que el de la interface con el sufijo Impl el cual es notación de GWT para reconocer a ese servlet como la implementación del servicio que define la interface que creamos anteriormente.

Cuando se llama al servicio desde el cliente GWT hace la mayor parte del trabajo por nosotros, sin embargo todavía se necesita crear una última interface. Esta interface será usada por el compilador de GWT cuando este genere el objeto proxy del servicio. Un objeto proxy es la instancia de un objeto que reenvía la petición a otro objetivo. En este caso se estará llamando a un método local y el

objeto proxy será el encargado de serializar los parámetros, llamar al servicio remoto, y manejar la deserialización del resultado. Todo esto lo hace GWT por nosotros. En el código del lado del cliente se crea el objeto proxy con la siguiente línea:

```
GWT.create(Servicio.class);
```

Aquí se llama al método estático create de la clase com.google.gwt.core.client.GWT pasándole como parámetro la clase de la interface del servicio. Esto devuelve un objeto proxy el cual se puede usar para asignar la URL del servicio y llamar los métodos del servicio remoto. El objeto proxy que se obtiene implementara 2 interfaces, una que debe ser creada y otra que provee GWT.



En la figura se ve un ejemplo con un servicio llamado ServeStatus, la interface ServiceDefTarget es la que viene con GWT, incluye el método setServiceEntryPoint para especificar la URL del servicio. La segunda es la que se define y nos provee métodos asíncronos para llamar el servicio remoto. Como se observa es el nombre del servicio con el sufijo Async. Los métodos en esta interface deben coincidir con los nombres de los métodos de nuestra interface de

servicio pero cambiando algo en la firma de los métodos, el valor de retorno debe ser void y además se debe agregar un parámetro de tipo com.google.gwt.user.client.rpc.AsyncCallback. En esta figura se muestra un ejemplo de esto:

Service interface	Asynchronous service interface
String methodOne (int i);	void methodOne (int i, AsynCallback cb);
List methodTwo ();	void methodTwo (AsynCallback cb);
boolean methodThree (int a, int x);	void methodThree (int a, int x, AsynCallback cb)

Esta interface es usada del lado del cliente por lo que debe ser ubicada en el paquete “cliente”.

Ahora que se ha definido todas las interfaces necesarias podemos realizar el llamado al servicio desde el cliente y para esto se debe hacer lo siguiente:

- ✓ Instanciar un objeto proxy como ya vimos.
- ✓ Especificar la URL del servicio.
- ✓ Crear un método callback para manejar el resultado de la llamada asíncrona del método.
- ✓ Llamar al método.

Conozcamos estos 4 pasos más detalladamente.

Instanciar el objeto proxy:

Se realiza esto llamando al método estático GWT.create() pasándole como parámetro la clase de la interface del servicio que creamos. Luego se necesita realizar un casting de este objeto a la interface asíncrona que ya se ha creado anteriormente.

```
ServerStatusServiceAsync serviceProxy =
  (ServerStatusServiceAsync) GWT.create(ServerStatusService.class);
```

Especificar la URL del servicio:

Para este paso se necesita realizar otro casting del objeto proxy que se capturó al tipo ServiceDefTarget para de esta manera definir la URL del servicio remoto.

```
ServiceDefTarget target = (ServiceDefTarget) serviceProxy;
target.setServiceEntryPoint(GWT.getModuleBaseURL() + "server-status");
```

La URL debe ser la misma que está en la definición del servlet en el web.xml.

Crear un objeto callback:

En este paso se creará un objeto callback, este objeto implementará la interface com.google.gwt.user.client.rpc.AsyncCallback y será ejecutado cuando el resultado sea recibido del servidor. Como ya recordarán se ha añadido un parámetro AsyncCallback en los parámetros de los métodos de la interface asíncrona que se definió, este objeto será el parámetro extra que enviaremos en el llamado a la función. Aquí se creará un objeto anónimo que implementa la interface AsyncCallback.

```
AsyncCallback callback = new AsyncCallback() {

            public void onFailure (Throwable caught) {
              GWT.log("RPC error", caught);
            }

           public void onSuccess (Object result) {
             GWT.log("RPC success", null);
           }
};
```

La interface AsyncCallback tiene 2 métodos que deben ser implementados que son onFailure que se ejecuta cuando hay algún error en la ejecución del método. Y el método onSuccess cuando el método fue ejecutado exitosamente y el resultado fue recibido del servidor, en este caso el resultado es recibido por el método en el parámetro result el cual se puede usar para cualquier operación que se desee realizar.

Llamando al servicio remoto:

El último paso es llamar al servicio, lo cual es bastante simple y consiste en una sola línea que sería:

```
serviceProxy.getStatusData(callback);
```

9 ENTORNOS DE DESARROLLO EN GWT

Para el desarrollo de GWT se necesita de un entorno para poder implementar las aplicaciones Java correspondientes, usar entornos de desarrollos integrados (IDE) y depurar dicha aplicación usando las librerías necesarias para GWT. Para esto existe una gran variedad de entornos de desarrollo integrados. Los más conocidos son:

NETBEANS:

Es una plataforma para el desarrollo de aplicaciones de escritorio que utiliza Java. Trabaja con módulos los cuales permiten que las aplicaciones sean desarrolladas. Para NetBeans existe un plugin llamado Gwt4nb que permite desarrollar proyectos GWT dentro del IDE.

ECLIPSE:

Es una plataforma de software de código abierto.

El pulgin que utiliza Eclipse es GWT+Designer el cual permite la aplicación de creaciones Web con tecnología AJAX por medio de la uso de GWT. Por su fácil manejo para los componentes dentro del área de diseño visual encargándose de escribir el código correspondiente a cada componente adicionado lo hace el más utilizado. También actualiza el área de diseño a medida que se escribe el código manualmente. Permite depurar la parte cliente y servidor de la aplicación como si fuera una aplicación swing/swt stand-alone, en "hosted mode", a una velocidad espectacular desde la versión 1.2.

JDEVELOPER:

Fue creado por Oracle Corporation, es un entorno integrado de desarrollo para lenguajes Java, HTML, XML, SQL, PL/SQL, JavaScript, entre otros.

JDK (java development kit):

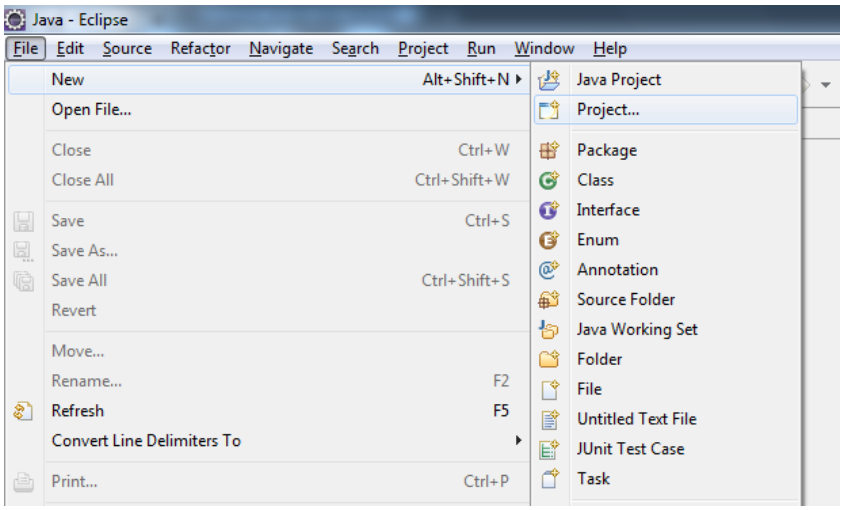
Es un compilador y un conjunto de herramientas de desarrollo que sirve para crear programas independientes y applets java.

10 CONSTRUYENDO UNA APLICACIÓN GWT CON ECLIPSE

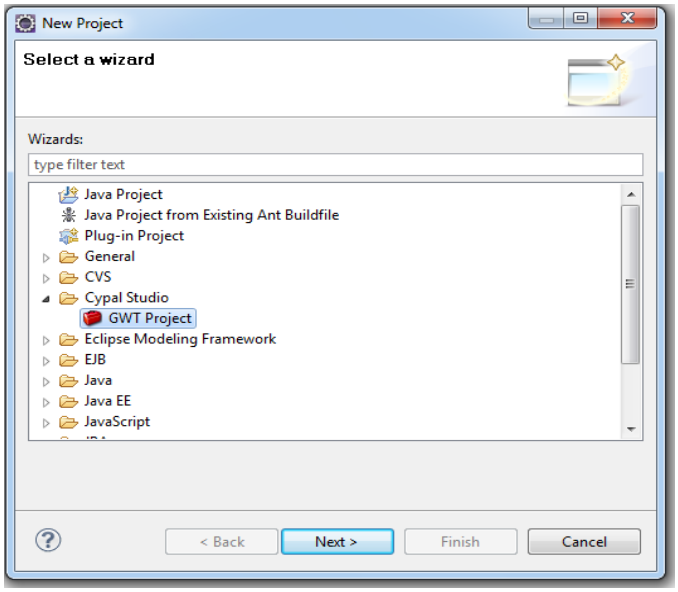
En capítulos anteriores se vieron los elementos básicos para crear una aplicación usando GWT, además se ilustro en los capítulos iniciales como configurar el plugin para eclipse llamado cypalstudio. A continuación se ilustrará como unir todas esas bases de capítulos anteriores y armar una aplicación web sencilla en GWT utilizando eclipse como IDE de desarrollo y el plugin cypalstudio para manejar proyectos de GWT usando este IDE.

Para ilustrar el uso de las herramientas se realizará un simulador de un blog en GWT. Un blog es un espacio donde se opina sobre un tema en específico que publica el autor del blog, se coloca el contenido y se da la posibilidad a los visitantes de dejar sus opiniones. ¿Porque un simulador? Porque la información publicada por los visitantes será almacenada en la memoria del servidor y no físicamente en una base de datos. Luego se verá cómo se va construyendo la aplicación pero primero se detallará como crear el proyecto en eclipse para empezar a trabajar en la aplicación.

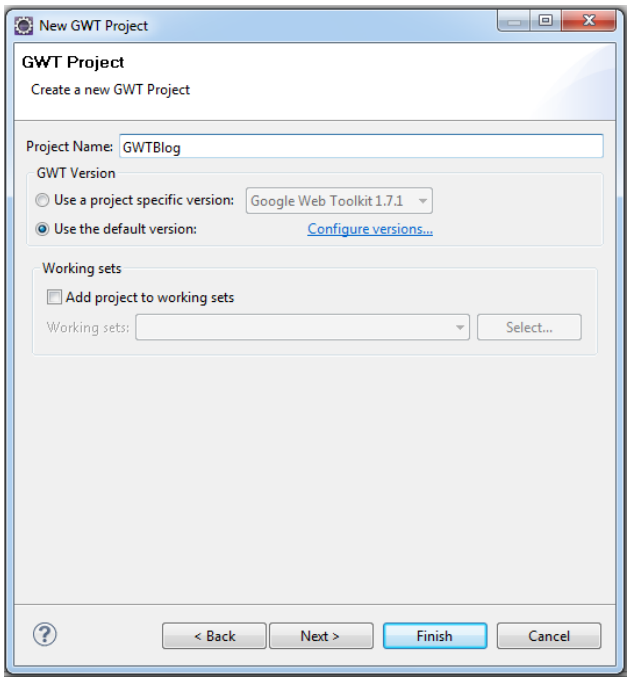
Lo primero a realizar será crear un nuevo proyecto en eclipse como se indica en la siguiente imagen. Se selecciona nuevo, proyecto



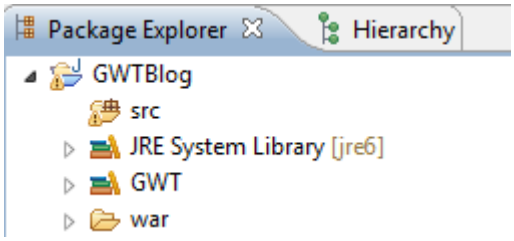
Luego se escoge la opción GWT Project.



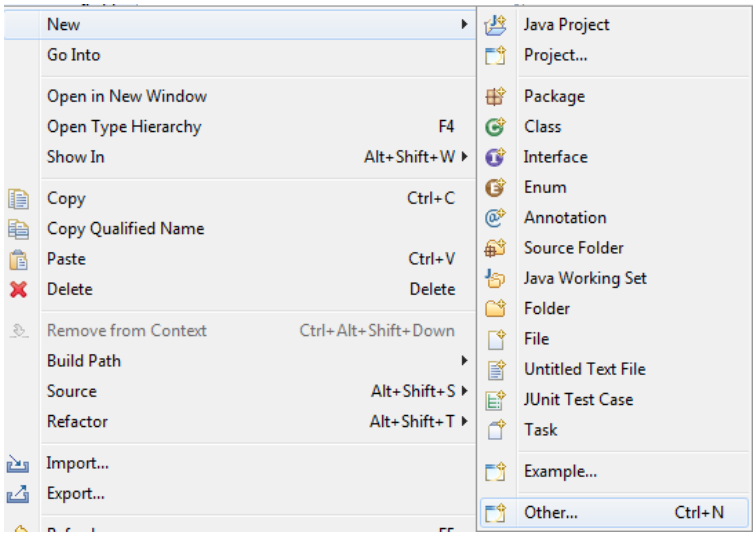
El siguiente paso es escoger el nombre del proyecto y escoger la versión de GWT con la que queremos trabajar, por defecto trabajaremos con la versión 1.7.1.



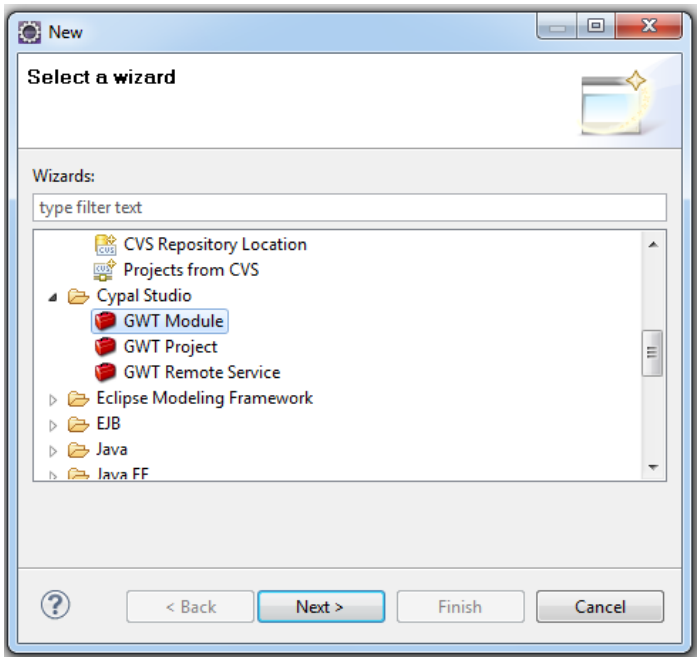
Finalizado el proceso eclipse nos creará la estructura de directorio del proyecto



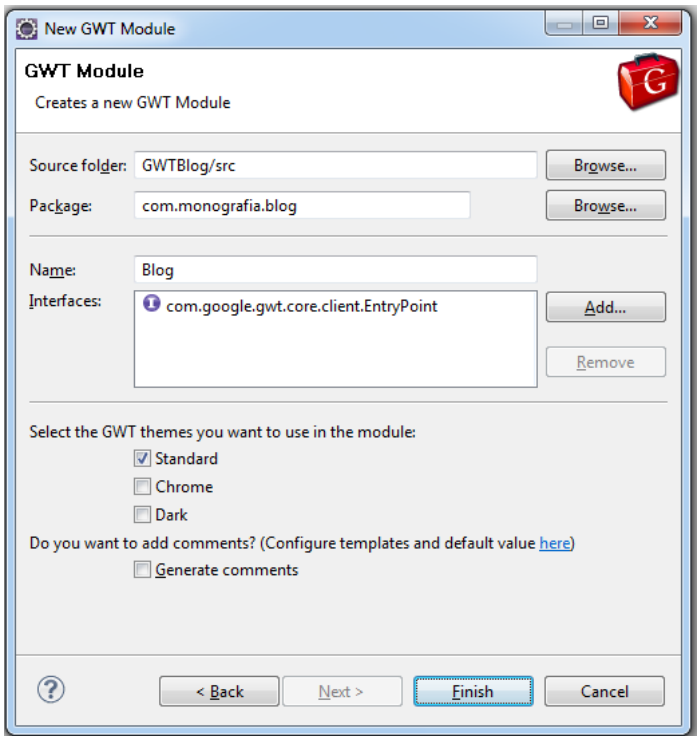
A continuación procederemos a crear un módulo para empezar a construir nuestra aplicación, esto se hace haciendo clic derecho sobre el proyecto y seleccionando nuevo -> otro.



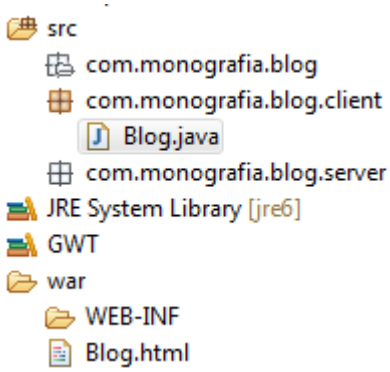
Esto nos mostrará una opción en donde escogeremos Cypal Studio -> GWT Module



Luego nos pedirá que ingresemos el paquete donde colocaremos el módulo y su nombre. Además de esto nos pedirá que seleccionemos un tema para nuestra aplicación. Por defecto esta seleccionado el standard y es el que se usará en este ejemplo.



Esto genera la siguiente estructura de directorios

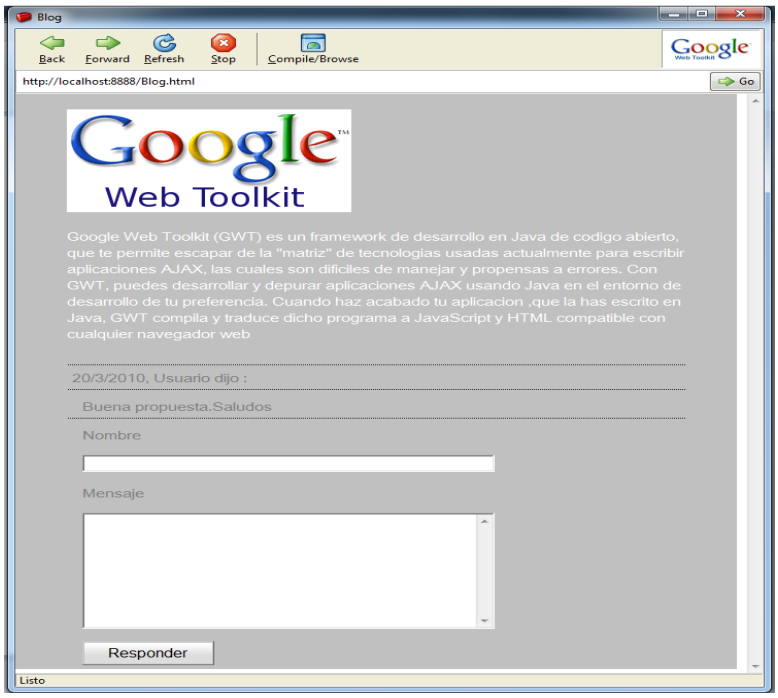


Se ve como generó el archivo Blog.java y Blog.html, el contenido del Blog.java es el siguiente:

```
package com.monografia.blog.client;

import com.google.gwt.core.client.EntryPoint;

public class Blog implements EntryPoint {

    public void onModuleLoad() {
        // TODO Auto-generated method stub
    }

}
```

Se puede ver que la clase implementa la interface EntryPoint e implementa el método onModuleLoad el cual es el que se ejecuta cuando el módulo Blog es “ejecutado” y que esta enlazado directamente con el archivo Blog.html.

Ahora si después de creado el proyecto empezaremos con el diseño de la aplicación. Este será la imagen de como quedara el simulador del blog al terminar

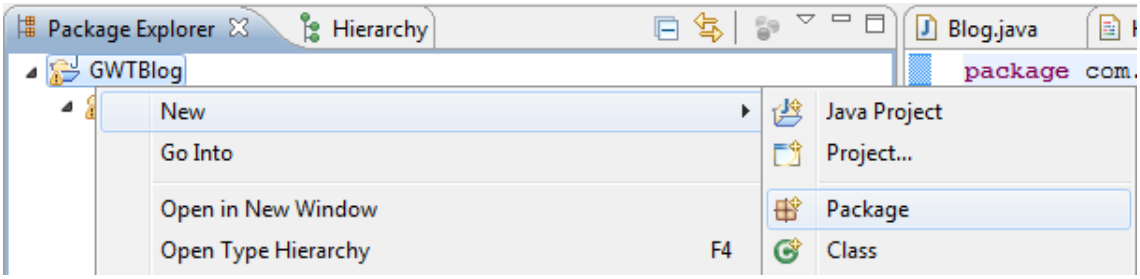


Como muestra la figura es un diseño sencillo, tiene 4 elementos básicos:

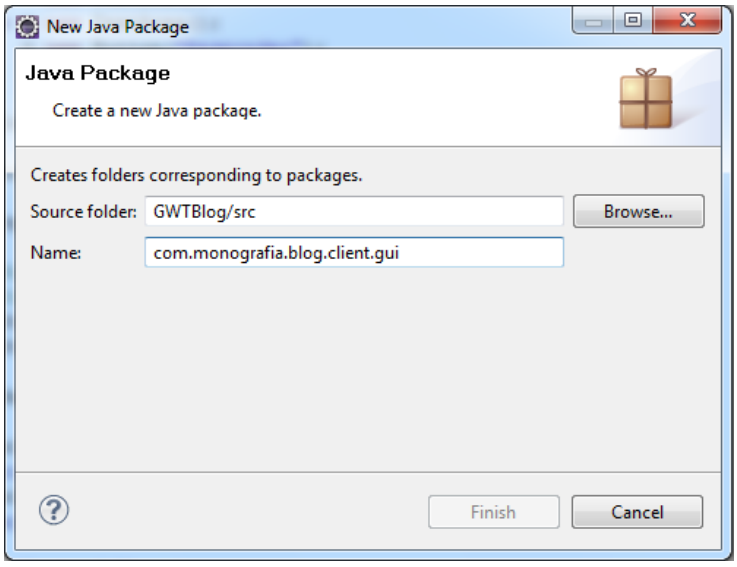
1. Imagen de GWT.
2. Un breve párrafo haciendo una introducción de lo que es GWT.
3. Sección de comentarios de algunos lectores.
4. Sección para que los lectores publiquen sus comentarios.

Con esta información se puede aplicar lo del capítulo donde se hizo referencia a la creación de Widgets personalizados. Aprovechando que se está analizando estos datos se puede proceder a crear la interfaz gráfica de la aplicación.

Para mantener la estructura del proyecto organizadas se crearán algunos paquetes, a continuación en la siguiente imagen se muestra como crear un nuevo paquete en eclipse.

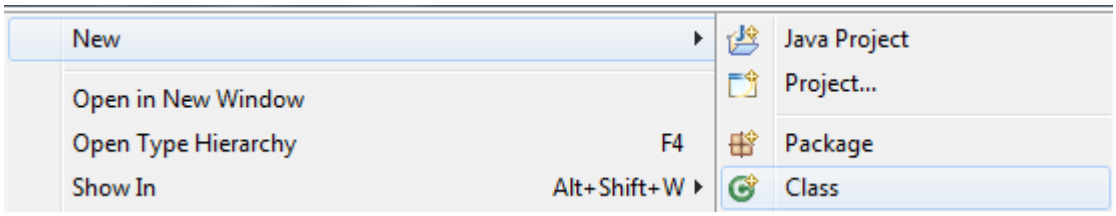


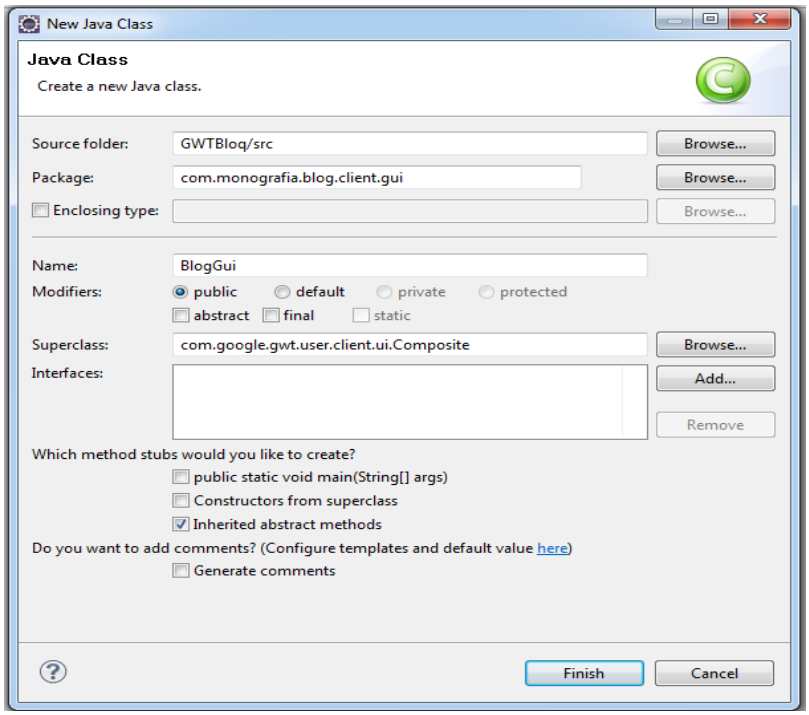
Se selecciona la opción que se ve en la figura, luego se muestra otra ventana donde elegimos el nombre del paquete y la carpeta donde se creará.



En este paquete se crearán las clases de la interface gráfica del blog.

Para comenzar se creará una clase llamada BlogGui que será el contenedor de toda la interface. Para hacer esto se agrega una clase como se muestra a continuación.





Como se muestra en la figura se escribe el nombre de la clase, en este caso BlogGui, se selecciona el paquete donde se va a crear, y se puede seleccionar la clase de la que heredara, como se va a crear un Widgets personalizado se escoge que herede de composite, ya en capítulos anteriores se explicó este procedimiento de creación de Widgets.

Ya se analizó anteriormente la estructura de la interface gráfica, y de este análisis se puede ver que los 2 primeros ítems son Widgets sencillos, un Image y un HTML, pero los otros 2 pueden ser tratados como otros subwidgets. La sección de los comentarios dejados consta de un título con la fecha y el nombre del usuario que lo dejó y abajo el comentario. En la imagen del blog vemos uno solo, pero si son varios comentario será una estructura que se repetirá tantos comentarios existan, por lo tanto podemos crear un Widgets para esta tarea, lo llamaremos BlogItem. El código sería más o menos el siguiente:

```
public class BlogItem extends Composite {


    SimplePanel titulo = new SimplePanel();
    ScrollPanel pmensaje = new ScrollPanel();

    public BlogItem(){

        VerticalPanel vp = new VerticalPanel();

        titulo.setStyleName("tituloentrada");
        pmensaje.setStyleName("mensajeentrada");

        vp.add(titulo);
        vp.add(pmensaje);
        vp.setStyleName("item");
        this.initWidget(vp);

    }
}
```

Es muy sencillo el diseño, se utiliza un SimplePanel para el título (se usó así para utilizar el div generado por el panel para efectos de la CSS), y un scrollPanel para el mensaje. Por último se agregan dichos Widgets en un VerticalPanel para obtener la organización vertical como se vio en el diseño. Además se le colocan nombres de estilo a los 2 componentes para efectos de CSS que se verá más adelante.

El último elemento de la interface es la parte donde se publican los comentarios. Se puede considerar otro Widgets el cual está compuesto por la parte donde se pide el nombre del que deja el comentario, y donde se deja el comentario. Este Widgets se llamara BlogEntry. El código del Widgets sería algo así:

```
public class BlogEntry extends Composite implements ClickHandler {

    TextBox  txtusuario = new TextBox();
    TextArea txtmensaje = new TextArea();
    Button   btnok      = new Button("Responder");
    BlogGui  padre;

    public BlogEntry(BlogGui padre){

        this.padre = padre;

        VerticalPanel vp = new VerticalPanel();

        txtusuario.setStyleName("txtusuario");
        txtmensaje.setStyleName("txtmensaje");

        btnok.addClickHandler(this);

        vp.add(new Label("Nombre"));
        vp.add(txtusuario);
        vp.add(new Label("Mensaje"));
        vp.add(txtmensaje);
        vp.add(btnok);
        vp.setStyleName("entrada");

        this.initWidget(vp);

    }


public void onClick(ClickEvent event) {

}
```

Como se ve el Widgets contiene el textbox donde se escribirá el nombre del usuario, un TextArea donde se escribirá el mensaje, el botón para mandar el mensaje y una instancia de BlogGui llamada padre que se referirá a la interface desde donde se está llamando a este Widgets. Además de heredar de Composite, la clase BlogEntry implementa la interface ClickHandler para manejar el evento del botón.

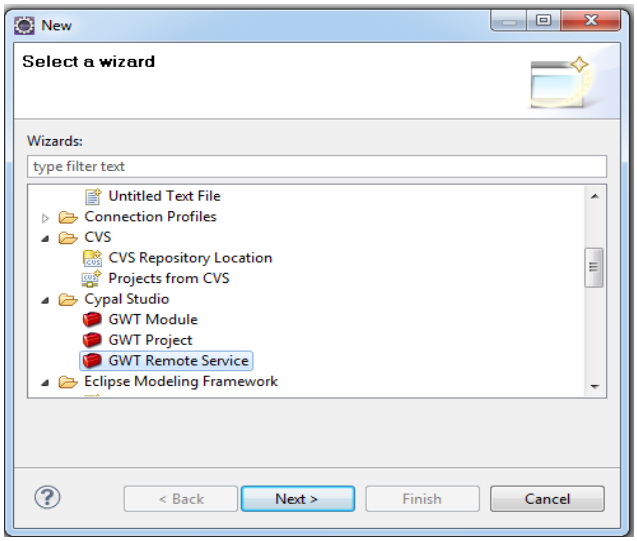
Ya que se tienen todos los elementos para armar la interface principal se puede empezar a construir BlogGui. El código de BlogGui sería el siguiente:

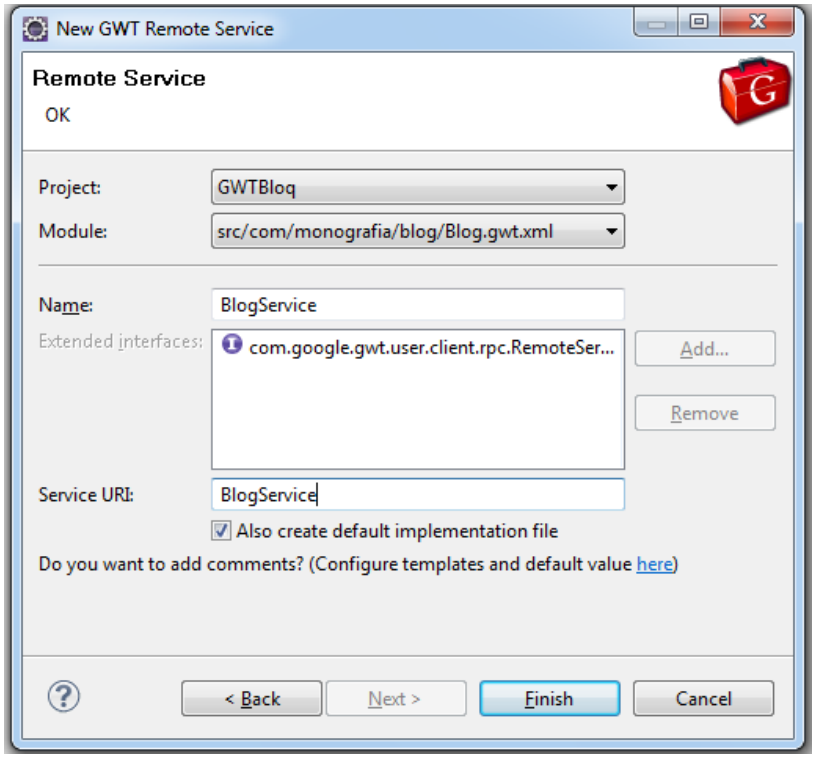
```
public class BlogGui extends Composite {

    VerticalPanel pblog = new VerticalPanel();
    VerticalPanel pmensajes = new VerticalPanel();
    public BlogGui( ){


        SimplePanel ptexto = new SimplePanel();
        ptexto.add(new HTML("Google Web Toolkit (GWT) es un framework de desarrollo en Java de codigo abierto,"
                            +"que te permite escapar de la \"matriz\" de tecnologias usadas actualmente para escribir"
                            +"aplicaciones AJAX, las cuales son dificiles de manejar y propensas a errores. Con GWT, "
                            +"puedes desarrollar y depurar aplicaciones AJAX usando Java en el entorno de desarrollo "
                            +"de tu preferencia. Cuando haz acabado tu aplicacion ,que la has escrito en Java, GWT compila "
                            +"y traduce dicho programa a JavaScript y HTML compatible con cualquier navegador web"));

        pblog.setStyleName("blogpanel");
        ptexto.setStyleName("intro");
        pmensajes.setStyleName("mensajes");

        pblog.add(new Image("http://img205.imageshack.us/img205/3061/logogwtan3.png"));
        pblog.add(ptexto);
        pblog.add(pmensajes);
        pblog.add(new BlogEntry(this));


        this.initWidget(pblog);
    }
}
```

Como se puede observar, la interface principalmente cuenta con 2 VerticalPanel, pblog que será el panel principal y en el cual irán todos los Widgets de la interface, y pmensajes donde estarán los mensajes dejados por los usuarios. Luego en el constructor se inicializan los diferentes Widgets de la interface y se van agregando a pblog, se crea un SimplePanel para ubicar el párrafo sobre GWT en un div, luego se agregan los ítems a pblog en el orden vertical en el que van a mostrarse en la interface, primero un Image con la imagen de GWT, luego el SimplePanel del párrafo, seguido del VerticalPanel de los mensajes y finalmente una instancia del Widgets BlogEntry parapermitirle al usuario dejar comentarios.

Ya se vio la parte del cliente que es la interface gráfica, ahora se continúa con el lado del servidor. Para seguir aplicando lo que se ha visto en capítulos anteriores

se usará RPC para la comunicación con el servidor. Para esto se agrega un nuevo servicio al proyecto, esto se hace de la siguiente manera, se agrega un nuevo ítem de la misma manera que se han agregado los otros, pero se selecciona GWT Remote Service.





Como se muestra en la figura se escoge el nombre del servicio y la URI con la que se va a llamar al servicio posteriormente cuando se use el RPC para la comunicación con el servidor. Al crear el servicio se crean las interfaces y la clase necesarias para la implementación del RPC como ya se vio en el capítulo de RPC. En este apartado no se pretende explicar RPC así que se limitará a explicar el código del lado del servidor.

Antes de continuar a explicar lo que se hace en el servidor primero se mostrara la manera en la que tratará la información de los comentarios dejados por los usuarios. Se creara una entidad llamada BlogObject que tendrá como atributos los datos de los comentarios de los usuarios, la clase BlogObject será la siguiente:

```
public class BlogObject implements IsSerializable {

    private String usuario;
    private String  fecha;
    private String mensaje;


    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
    public String getUsuario() {
        return usuario;
    }
    public void setFecha(String fecha) {
        this.fecha = fecha;
    }
    public String getFecha() {
        return fecha;
    }
    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }
    public String getMensaje() {
        return mensaje;
    }
```

Como se ve la clase contiene el nombre del usuario, la fecha en que fue dejado el comentario (String por conveniencia), y el mensaje de dicho comentario, y además sus getters y setters correspondientes. Además implementa la interface IsSerializable de GWT para poder usarla en el RPC.

Ahora si se analizara el funcionamiento del servicio, básicamente se necesitan los siguientes requisitos del servicio:

1. Almacenar los comentarios.
2. Agregar un comentario.
3. Devolver los comentarios que se encuentran almacenados.

Basándose en esos requisitos se construye el servicio. Gracias al plugin cada método que se cree en una de las interfaces o en la clase se reflejara en las demás interfaces o en la clase de la arquitectura del RCP que necesite actualizarse, así que solamente se centrará en mostrar la implementación del servicio en el servidor que es la siguiente:

```
public class BlogServiceImpl extends RemoteServiceServlet implements BlogService {

    private static final long serialVersionUID = 1L;

    private ArrayList<BlogObject> blogentradas = new ArrayList<BlogObject>();

    @Override
    public void agregarEntrada(BlogObject entrada) {


        Calendar calendar = Calendar.getInstance();
        String fecha = "" + calendar.get(Calendar.DAY_OF_MONTH) + "/"
                          + calendar.get(Calendar.MONTH)        + "/"
                          + calendar.get(Calendar.YEAR);

        entrada.setFecha(fecha);
        blogentradas.add(entrada);

    }

    @Override
    public ArrayList<BlogObject> obtenerEntradas() {
        return blogentradas;
    }

}
```

Se puede observar que como atributo tiene una colección con las entradas o comentarios del blog, y 2 métodos. El primero agregarEntrada que recibe como

parámetro un BlogObject y simplemente lo agrega a la colección agregándole primero la fecha del servidor en la que fue recibido el comentario. El segundo método obtenerEntradas devuelve todas las entradas o comentarios que se encuentran en el servidor.

Ya que se tiene la parte del servidor se necesita algo para manejar la conexión con el servidor y hacer las llamadas a los eventos. Para esto se crea una clase llamada BlogManejador. Esta clase lo que nos va a permitir es realizar los llamados a las funciones del servlet que se encuentran en el servidor por medio de RPC.

La clase BlogManejador es la siguiente:

```
public class BlogManejador {

    BlogServiceAsync servicioblog;

    public BlogManejador(){
        servicioblog = BlogService.Util.getInstance();
        ServiceDefTarget  target = (ServiceDefTarget)servicioblog;
        target.setServiceEntryPoint(GWT.getModuleBaseURL()+"blogservice");

     }

     public void agregarMensaje(BlogObject entrada){

     public void obtenerEntradas(){
}
```

Es una clase muy sencilla, tiene un atributo servicioblog que será el que se utilice para realizar el RPC, en el constructor se ve como se inicializa este atributo para poder ejecutar el RPC, este procedimiento ya se explicó en el capítulo de RPC por lo que no se volverá a explicar aquí. Los 2 métodos agregarMensaje y obtenerEntradas se verán luego.

Ya se tiene el servidor, se tiene el manejador, se tiene la interface, ¿Qué hace falta?, una clase que sea la aplicación como tal y que contenga la interface gráfica y el manejador. Dicha clase se ha llamado BlogAplicacion. En esta clase se tendrán 2 atributos que serán la interface gráfica y el manejador además de unos métodos que se encargaran de funcionar como puente entre el manejador y la interface gráfica. La clase BlogAplicacion sería la siguiente:

```
public class BlogAplicacion {

    private BlogGui gui ;
    private BlogManejador manejador;

    public BlogAplicacion(){

        setGui(new BlogGui(this));
        setManejador(new BlogManejador(this));

    }

    void setGui(BlogGui gui) {

    public BlogGui getGui() {

    private void setManejador(BlogManejador manejador) {

    private BlogManejador getManejador() {

    public void obtenerEntradas(){
            getManejador().obtenerEntradas();
    }

    public void agregarEntradas(ArrayList<BlogObject> entradas){

        gui.limpiarEntradas();
        for (int i = 0 ; i< entradas.size() ; i++){
            gui.agregarEntrada(entradas.get(i));
        }

    }

    public void enviarEntrada(BlogObject entrada){
        getManejador().agregarMensaje(entrada);
    }

    public void mostrarError(String mensaje){
        Window.alert(mensaje);
    }


}
```

Se observan los 2 atributos, los setters y getters de los atributos y 4 métodos que son los siguientes:

1. obtenerEntradas: como su nombre lo dice obtiene las entradas o comentarios del servidor, lo hace llamando al método obtenerEntradas del manejador.

2. agregarEntradas: recibe una colección de BlogObject y agrega el contenido de estos en la interface gráfica.

3. enviarEntrada: recibe como parámetro un BlogObject y lo envía al servidor.

4. mostrarError: muestra un mensaje si sucede algún error.

Si observa bien se notan algunas diferencias en el uso de los objetos BlogGui y BlogManejador teniendo el código que se ha mostrado de ellos anteriormente. Una de ellas es que en el constructor reciben como parámetro un objeto de tipo BlogAplicacion, y otra es unos métodos de BlogGui que no se habían visto. A continuación se hace la actualización de las 2 clases:

```
public class BlogGui extends Composite {

    VerticalPanel pblog = new VerticalPanel();
    VerticalPanel pmensajes = new VerticalPanel();
    BlogAplicacion app;

    public BlogGui(BlogAplicacion app){

        this.app = app;

        SimplePanel ptexto = new SimplePanel();
        ptexto.add(new HTML("Google Web Toolkit (GWT) es un framework de desarrollo en Java de codigo abierto,"
                            +"que te permite escapar de la \"matriz\" de tecnologias usadas actualmente para escribir"
                            +"aplicaciones AJAX, las cuales son dificiles de manejar y propensas a errores. Con GWT, "
                            +"puedes desarrollar y depurar aplicaciones AJAX usando Java en el entorno de desarrollo "
                            +"de tu preferencia. Cuando haz acabado tu aplicacion ,que la has escrito en Java, GWT compila "
                            +"y traduce dicho programa a JavaScript y HTML compatible con cualquier navegador web"));

        pblog.setStyleName("blogpanel");
        ptexto.setStyleName("intro");
        pmensajes.setStyleName("mensajes");
        pblog.add(new Image("http://img205.imageshack.us/img205/3061/logogwtan3.png"));
        pblog.add(ptexto);
        pblog.add(pmensajes);
        pblog.add(new BlogEntry(this));

        this.initWidget(pblog);
        app.obtenerEntradas();

    }

    public void agregarEntrada(BlogObject entrada){

        BlogItem item = new BlogItem();
        item.setInfo(entrada);
        pmensajes.add(item);

    }

    public void enviarMensaje(BlogObject entrada){
        app.enviarEntrada(entrada);
    }

    public void limpiarEntradas(){

        pmensajes.clear();

    }
}
```

Aquí se ve que tiene un atributo adicional y es la aplicación. Además de 3 métodos que son los siguientes:

1. agregarEntrada: agrega un mensaje en la interface gráfica recibiendo como parámetro un BlogObject.
2. enviarMensaje: envía el mensaje que el usuario ingreso a la interface gráfica.
3. limpiarEntradas: borra el panel donde se encuentran los mensajes dejados por los usuarios.

Con estos cambios también se actualizó la clase BlogEntry quedando como sigue:

```
public class BlogEntry extends Composite implements ClickHandler {

    TextBox  txtusuario = new TextBox();
    TextArea txtmensaje = new TextArea();
    Button   btnok      = new Button("Responder");
    BlogGui  padre;

    public BlogEntry(BlogGui padre){

        this.padre = padre;
        VerticalPanel vp = new VerticalPanel();

        txtusuario.setStyleName("txtusuario");
        txtmensaje.setStyleName("txtmensaje");
        btnok.addClickHandler(this);

        vp.add(new Label("Nombre"));
        vp.add(txtusuario);
        vp.add(new Label("Mensaje"));
        vp.add(txtmensaje);
        vp.add(btnok);
        vp.setStyleName("entrada");
        this.initWidget(vp);
    }

public void onClick(ClickEvent event) {
        // TODO Auto-generated method stub
        if (txtusuario.getText().trim().isEmpty()){

            Window.alert("debe escribir un nombre");
            txtusuario.setFocus(true);
        }
        else if(txtmensaje.getText().trim().isEmpty()){

            Window.alert("debe escribir un mensaje");
            txtmensaje.setFocus(true);
        }
        else{

            BlogObject entrada = new BlogObject();
            entrada.setMensaje(this.txtmensaje.getText());
            entrada.setUsuario(this.txtusuario.getText());
            padre.enviarMensaje(entrada);
            this.txtusuario.setText("");
            this.txtusuario.setFocus(true);
            this.txtmensaje.setText("");

        }
    }

}
```

El cambio que se ve es el método del evento clic del botón donde se crea un BlogObject de los datos ingresados por el usuario y se llama al método enviarMensaje de BlogGui, que luego lo envía a la aplicación para enviar el mensaje al servidor a través del manejador.

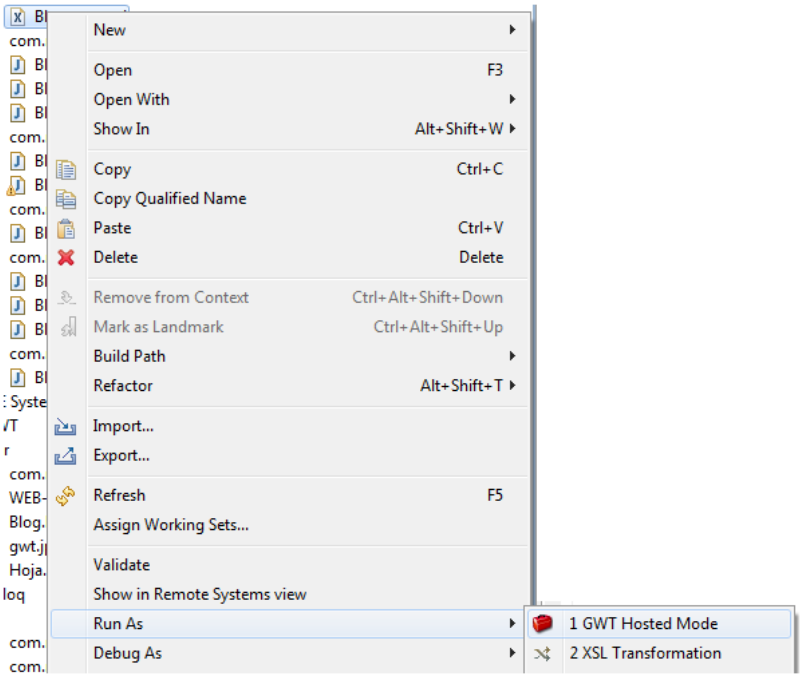
Ahora se mostrarán los métodos del manejador:

```
public class BlogManejador {

    BlogAplicacion app ;
    BlogServiceAsync servicioblog;

    public BlogManejador(BlogAplicacion app){

        this.app = app;
        servicioblog = BlogService.Util.getInstance();
        ServiceDefTarget  target = (ServiceDefTarget)servicioblog;
        target.setServiceEntryPoint(GWT.getModuleBaseURL()+"blogservice");
    }

    public void agregarMensaje(BlogObject entrada){
        AsyncCallback callback = new AsyncCallback(){
            @Override
            public void onFailure(Throwable caught) {
                // TODO Auto-generated method stub
                app.mostrarError(caught.getMessage());
            }
            @Override
            public void onSuccess(Object result) {
                // TODO Auto-generated method stub
                app.obtenerEntradas();

            }
        };

        servicioblog.agregarEntrada(entrada, callback);
    }

    public void obtenerEntradas(){
        AsyncCallback callback = new AsyncCallback(){
            @Override
            public void onFailure(Throwable caught) {
                // TODO Auto-generated method stub
                app.mostrarError(caught.getMessage());

            }

            @Override
            public void onSuccess(Object result) {
                // TODO Auto-generated method stub
                ArrayList<BlogObject> entradas = (ArrayList<BlogObject>)result;
                app.agregarEntradas(entradas);
            }
        };

        servicioblog.obtenerEntradas(callback);
    }
}
```

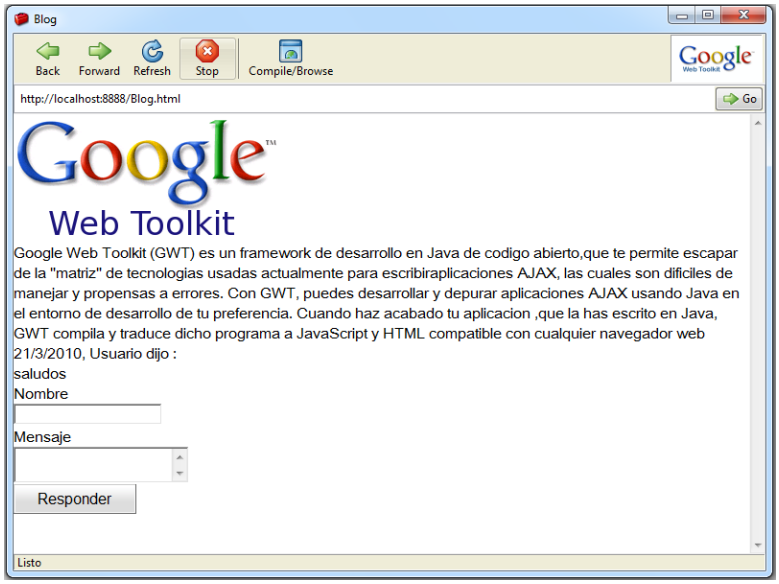
A continuación se explican ambos métodos:

1. agregarMensaje: recibe un BlogObject y crea un callback para utilizar el RPC. Como ya se sabe por el capítulo de RPC el callback tiene 2 métodos: onFailure y onSuccess. Si el RPC fue exitoso se ejecutara onSuccess, en este caso se llama al método obtenerEntradas de la aplicación que se encargará de refrescar los mensajes en la interface gráfica. Si en cambio hubo un error se ejecutara onFailure, en el cual se llama al método de la aplicación mostrarError donde se mostrara el error ocurrido. Para agregar el mensaje en el servidor se llama al método agregarEntrada de servicioblog pasando como parámetro el BlogObject y el callback que se creó recién.
2. obtenerEntradas: en este método se traen todos los mensajes que se encuentran en el servidor. En onFailure del callback se vuelve a llamar al método mostrarError de la aplicación. En onSuccess se realiza un casting del resultado que nos devuelve el servidor, que es un tipo Object , a una colección de BlogObject y luego se llama al método agregarEntradas de la aplicación que se encargara de refrescar la interface gráfica con los mensajes. La llamada al servidor se hace llamando al método obtenerEntradas de servicioblog pasándole como parámetro el callback que se acabó de crear.

Ya la aplicación esta lista para funcionar. Para correr nuestra aplicación desde eclipse podemos correrla en modo hosted de la siguiente manera: se selecciona el módulo que se desea correr, en este caso blog y se da clic derecho y se selecciona Run -> GWT Hosted Mode



Lo que daría como resultado lo siguiente:



Pero si se tiene en cuenta la primera imagen de la aplicación y se mira la imagen anterior se nota cierta diferencia. Y esto es porque no se creado ninguna hoja de

estilo para manejar la apariencia de la aplicación. No es competencia de este documento hablar sobre hojas de estilo CSS así que se explicara brevemente lo que se hizo en la siguiente CSS

```
.blog{
    padding-left:50px;
    padding-right:50px;
    padding-top:20px;
    width:700px;
    background-color:silver;
    margin:auto;
}

.intro{
    margin-top:20px;
    margin-bottom:20px;
    width:600px;
    color:white;
    font-style:bold;
}
.item{
    margin-top:10px;
    border-style:dotted;
    border-width:1px;
    border-color:black;
    border-left-width:0px;
    border-right-width:0px;
    width:600px;
}
```

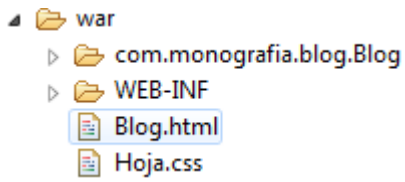
```
.tituloentrada{
    border-style:dotted;
    border-top-width:0px;
    border-left-width:0px;
    border-right-width:0px;
    border-bottom-width:1px;
    border-color:black;
    margin-top:5px;
    padding-bottom:5px;
    padding-left:5px;
    color:gray;
    font-style:bold;
}
.mensajeentrada{
    padding-top:10px;
    padding-left:15px;
    padding-right:5px;
    padding-bottom:5px;
    heigh:auto;
    color:gray;
    font-style:bold;
}
.entrada{
    color:gray;
    font-style:bold;
    padding-top:10px;
    padding-left:15px;
    padding-right:5px;
    padding-bottom:5px;
}
```

```
.txtusuario{
    width:400px;
}
.txtmensaje{
    width:400px;
    height:150px;
}
```

Todas esas clases que se ven ahí fueron los nombres de estilos que se les agrego a los objetos en la construcción de la interface gráfica. Básicamente lo que se hace es ajustar un tamaño de ancho para los divs, usar un color de fondo que será plateado, se juega con los padding para tener un cierto espaciamiento entre los objetos y su contener y no estén todos muy juntos. En el párrafo informativo sobre GWT se coloca la letra de color blanco, mientras que en la sección donde se muestran los mensajes la letra es de color gris y en negrita. Además se usa como separador el borde de color negro y punteado en cada div de los mensajes. Finalmente se le coloca un tamaño al TextBox y al TextArea.

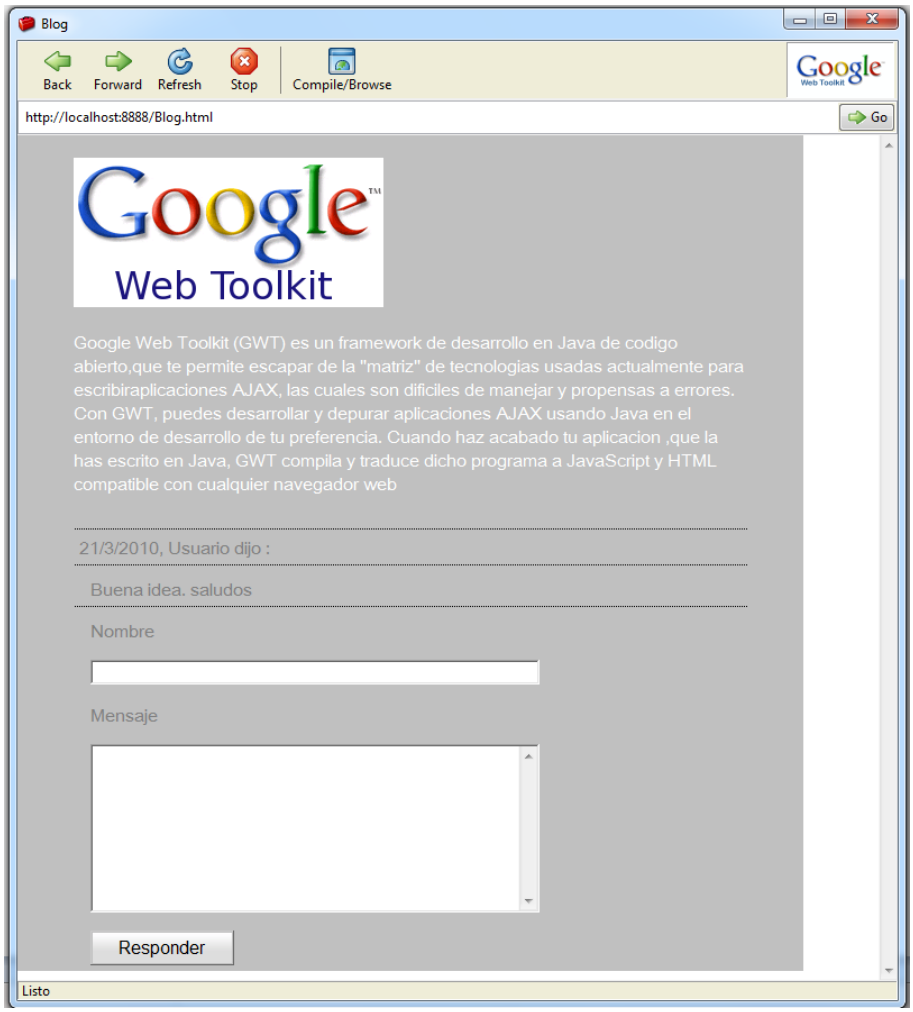
Ya que se tiene la hoja de estilo el siguiente paso es agregarla a la página HTML. Al inicio de se dijo que por cada módulo se creaba una página HTML, esta página

es donde vamos a colocar la referencia a la hoja de estilo. La página se ve en la estructura de directorios del proyecto.



Se abre dicha página y se agrega el enlace a la hoja de estilo como sigue.

Luego corremos de nuevo la aplicación y el resultado será el esperado:


Blog
Back
Forward
Refresh
Stop
Compile/Browse
Google
Web Toolkit
http://localhost:8888/Blog.html
Go
Google
Web Toolkit
Google Web Toolkit (GWT) es un framework de desarrollo en Java de codigo abierto,que te permite escapar de la "matriz" de tecnologias usadas actualmente para escribiraplicaciones AJAX, las cuales son dificiles de manejar y propensas a errores. Con GWT, puedes desarrollar y depurar aplicaciones AJAX usando Java en el entorno de desarrollo de tu preferencia. Cuando haz acabado tu aplicacion ,que la has escrito en Java, GWT compila y traduce dicho programa a JavaScript y HTML compatible con cualquier navegador web
21/3/2010, Usuario dijo :
Buena idea. saludos
Nombre
Mensaje
Responder
Listo

11 CONCLUSIÓN

Al hablar de Google Web Toolkit se logró conocer sus características, tales como su compatibilidad con la mayoría de navegadores, que es de codfigo abierto, RPC fácil, entre otras; así como sus principales componentes y la importancia de esta herramienta al momento de crear páginas web.

Como se mostró en el capítulo 5, la configuración e instalación de Google Web Toolkit es sencilla y rápida permitiendo desarrollar aplicaciones WEB.

Google Web Toolkit incluye los Widgets y Paneles comúnmente utilizados en las aplicaciones RIA, con una característica adicional importante y es que permite desarrollar Widgets complejos basándose en los básicos.

En el capítulo 8 se observó que Google Web Toolkit implementó un método de comunicación con el servidor llamado GWT-RPC, el cual permite ejecutar métodos que se encuentran en otras computadoras y traer sus resultados, permitiendo desarrollar aplicaciones web de varias capas y niveles.

Las aplicaciones RIA con GWT se pueden desarrollar haciendo uso de los IDE más populares en la comunidad como: NETBEANS, ECLIPSE Y JDEVELOPER. En el anterior trabajo se trabajo con ECLIPSE pero esto no quiere decir que sea el mejor y cada persona escoge el que más conozca.

Al momento de realizar la aplicación que se muestra como ejemplo, se observó lo fácil que es trabajar con Google Web Toolkit si se tiene buenos conocimientos en JAVA y sin necesidad de ser expertos en HTML y JavaScript.

12 RECOMENDACIONES

Google Web Toolkit posee varios componentes, en el anterior trabajo solo se mostraron los principales. Se puede realizar un estudio más a fondo acerca de todos los componentes de Google Web Toolkit y así poder conocer más acerca de ellos.

Explorar más a fondo los paneles y Widgets de Google Web Toolkit así como creación de nuevos paneles y Widgets y mostrar ejemplos tanto fáciles como complejos.

En base a este manual se puede crear aplicaciones más complejas utilizando todo lo aprendido, así como la utilización de otros entornos de desarrollo.

13 GLOSARIO

A

AJAX (JavaScript And XML): Es una técnica de desarrollo web para crear aplicaciones interactivas.

ATRIBUTOS: Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

B

BROWSER: Es un programa que permite visualizar la información que contiene una página web.

BYTECODES: Código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina .

C

COMPILADOR: Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.

CSS: Es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML

D

DOM (**Document Object Model):** Es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

F

FRAMEWORK: Es una estructura conceptual y tecnológica de soporte definida, normalmente con módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado

G

GWT: Framework creado por Google que permite ocultar la complejidad de varios aspectos de la tecnología AJAX.

H

HIPERVÍNCULOS: Es un enlace, normalmente entre dos páginas web de un mismo sitio, pero un enlace también puede apuntar a una página de otro sitio web, a un fichero, a una imagen, etc.

HTML (HyperText Markup Language): Lenguaje de marcado predominante para la construcción de páginas web.

I

INTERFACE: Colección de métodos abstractos. En ellas se especifica qué se debe hacer pero no su implementación.

J

JAVASCRIPT: Es un lenguaje de scripting basado en objetos, utilizado para acceder a objetos en aplicaciones. Principalmente, se utiliza integrado en un navegador web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas.

JVM (Java Virtual Machine): Es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial, el cual es generado por el compilador del lenguaje Java.

P

PANELES: Están diseñados para contener Widgets dentro y son usados para determinar como la interfaz de usuario va a ser distribuida sobre el navegador.

R

RPC (Remote Procedure Call): Es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.

S

SCRIPTS: Es un programa usualmente simple, que generalmente se almacena en un archivo de texto plano. Los script son casi siempre interpretados, pero no todo programa interpretado es considerado un script. El uso habitual de los scripts es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario.

SCROLL:

SERVIDOR: Es una computadora que, formando parte de una red, provee servicios a otras computadoras denominadas clientes. Aplicación informática o programa que realiza algunas tareas en beneficio de otras aplicaciones llamadas clientes.

SERVLET: Es un programa que se ejecuta en un servidor. El uso más común de los servlets es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

W

WIDGETS: Son los componentes y partes visuales de una aplicación.

X

XML (Extensible Markup Language): Es una manera de definir lenguajes para diferentes necesidades.

14 BIBLIOGRAFIA

- ✓ Google Web Toolkit Aplications, Ryan Dewsbury, Prentice Hall 2007.
- ✓ Google Web Toolkit Solutions, David Geary-Rob Gordon, Prentice Hall 2007.
- ✓ GWT in action. Easy Ajax with the Google Web Toolkit, Robert Hanson-Adam Tacy, Manning 2007.
- ✓ GWT in practice, Robert T. Cooper-Charlie E. Collins , Manning 2008
- ✓ http://code.google.com/intl/es-ES/webtoolkit/doc/latest/tutorial/
- ✓ http://esgooglewebtoolkit.blogspot.com/
- ✓ http://www.cypal.in/studiodocs
- ✓ http://casidiablo.net/tutorial-de-introduccion-al-google-web-toolkit/