

### Objetivo general del taller

El objetivo general de este taller es dar una introducción a Java y al ambiente de desarrollo que se usará dentro del curso y asegurar que cada estudiante cuenta con un ambiente de trabajo en el que pueda desarrollar todas las actividades que requieran programar.

### Objetivos específicos del taller

Durante el desarrollo de este taller se buscará que los estudiantes alcancen los siguientes objetivos:

1. Tener un ambiente de trabajo instalado y configurado en su computador con una distribución para desarrollo de Java instalada (OpenJDK) y un IDE (Eclipse).
2. Entender los elementos que hacen parte de una aplicación Java y un proyecto en Eclipse.
3. Poder compilar y ejecutar una aplicación Java desde Eclipse.
4. Modificar el código fuente de una aplicación Java y ver el resultado de la modificación.

### Instrucciones generales

1. Descargue de Bloque Neón el archivo nTaller1-Gasolinera\_esqueleto.zip y descomprímalo en una carpeta dentro de su computador a la que pueda llegar con facilidad.
2. Lea con cuidado este documento y vaya realizando las actividades una por una. Lea el documento al tiempo que vaya resolviendo el taller: en este documento encontrará información importante para completar el programa mientras que va entendiendo lo que está haciendo.
3. No se preocupe si hay términos en este documento que no conozca o si no entiende algunos aspectos particulares del programa sobre el que va a trabajar. Este taller pretende dar sólo una primera mirada a Java e ir introduciendo la terminología relevante. Al final del semestre deberían haber quedado resueltas todas las dudas.

**Este taller debe desarrollarse individualmente.**

### Parte 1: Instalación del ambiente de trabajo

Para realizar las actividades de este taller y del curso, usted necesita tener un ambiente de trabajo completo en su computador. Esto incluye los siguientes dos elementos principales:

- Una máquina virtual de Java (JVM), para poder ejecutar las aplicaciones. Sin embargo, no puede ser una versión SE, debe tener un JDK (Java Development Kit) para poder compilar las aplicaciones y tener disponibles las librerías básicas del lenguaje.
- Un ambiente de desarrollo (IDE) para poder editar las aplicaciones y realizar otras tareas relacionadas.

Para cubrir el primer punto existen varias alternativas, pero hay dos principales: Oracle JDK y Open JDK. La primera alternativa es la “oficial”, pero está especialmente dirigida a empresas (de hecho, para poder aprovecharla al máximo se debería pagar una licencia). La segunda es la versión abierta y, aunque no es oficial, no tiene ninguna limitación real que nos interese a nosotros, así que es la que usaremos.

Con respecto a la versión, le recomendamos usar una de las últimas oficialmente disponibles y no una de las que están en versión *Early Access*.

El ambiente de desarrollo que vamos a utilizar es **Eclipse**, o más específicamente el paquete llamado “Eclipse IDE for Java Developers” (que muchas veces simplemente es llamado “Eclipse”). En realidad, Eclipse es una plataforma sobre la cual se pueden desplegar componentes basados en OSGI, que permiten desarrollar una infinidad de funciones sin tener que cambiar la plataforma. Por ejemplo, con los componentes adecuados es posible usar Eclipse para desarrollar aplicaciones en Java, C, C++, Python, PHP, entre otros. Otros componentes permiten también construir modelos y diagramas, e incluso hay juegos (sencillos) desarrollados sobre esta plataforma.

Existen alternativas para Eclipse, como IntelliJ, pero ninguna tiene el mismo balance tan positivo entre funcionalidades y costo de licencia (en IntelliJ las licencias son gratuitas para cosas académicas, pero no lo son si luego van a usar esa herramienta para algún asunto profesional). Finalmente, todos los esqueletos de proyectos que nosotros les demos durante el semestre van a estar configurados para Eclipse. Se podrían portar a otras herramientas, pero eso sería su responsabilidad y además ustedes tendrían que portarlos de regreso a Eclipse para hacer la entrega.

## Actividades

1. Descargue e instale una versión de desarrollo de Java en su computador. Puede visitar este sitio web para encontrar los archivos de instalación: <https://jdk.java.net/> Asegúrese de descargar un JDK.
2. Abra la línea de comandos de su equipo (cmd en Windows o la terminal de Linux o Mac OS) y ejecute el siguiente comando para asegurar que todo esté correctamente configurado:

```
javac -version
```

Si todo está bien, usted debería ver un mensaje con el número de la versión que tiene instalada. Si no funciona (le dice que el comando `javac` no existe), revise por qué no quedaron bien configuradas las rutas de ejecución (el *PATH* del sistema).

3. Descargue e instale Eclipse en su computador. Puede encontrar los últimos instaladores en <https://www.eclipse.org/downloads/>.
4. Ejecute Eclipse desde donde haya quedado instalado.
5. Eclipse va a preguntarle en qué carpeta quiere tener el *workspace* o espacio de trabajo. Seleccione una nueva carpeta que pueda ubicar con facilidad, para dejar ahí todos los proyectos en los que trabaje con Eclipse. Le recomendamos utilizar siempre el mismo *workspace* porque ahí también se almacenarán la configuración personalizada que usted haga del IDE (por ejemplo, los tipos de letra, los estándares para organizar el código, etc.).

## Parte 2: Caso de estudio

En esta parte del taller vamos a cargar en Eclipse un proyecto existente construido con Java y estudiar algunos de sus componentes principales.

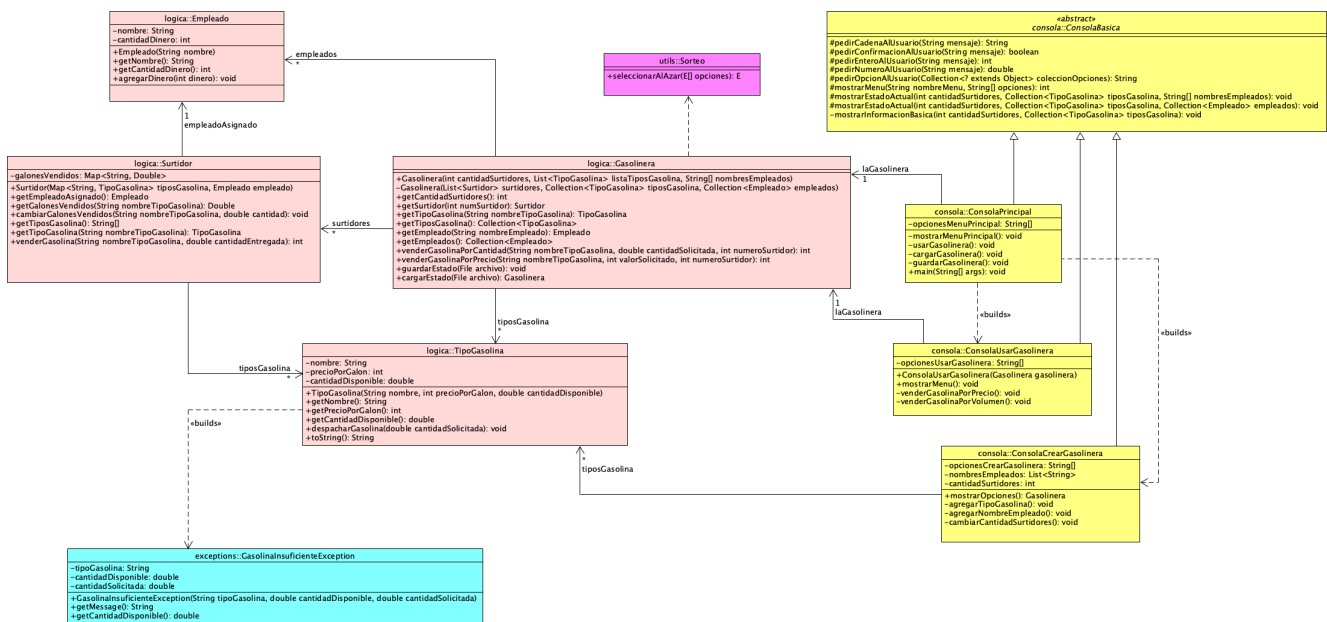
## Descripción del caso de estudio: Gasolinera

El caso de estudio que vamos a utilizar para esta parte del taller modela una gasolinera que tiene varios empleados y varios surtidores. El código de la aplicación está dividido en dos carpetas: **src**, donde se encuentran las fuentes principales, y **tests**, donde se encuentra la implementación de las pruebas automáticas de la aplicación.

Dentro de src, los archivos están divididos en 4 paquetes:

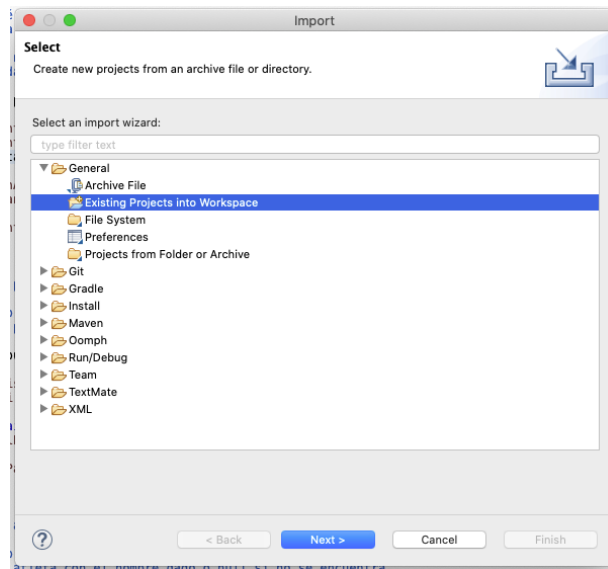
- **console**, donde están las clases que sirven para mediar en la interacción con el usuario.
- **exceptions**, donde está una clase que representa un error lógico que puede presentarse en el programa.
- **logica**, donde se encuentran las clases que implementan la lógica de la aplicación.
- **utils**, donde hay una clase que se usa en la aplicación, pero no está relacionada con el caso de estudio

El siguiente diagrama muestra todas las clases que hacen parte del proyecto (dentro de la carpeta docs encontrará una versión de mejor calidad). En este diagrama pueden verse las relaciones entre las clases. Estas relaciones las estudiaremos con mucho detalle durante el curso. Cada uno de las clases que aparecen en este diagrama corresponde a un archivo java dentro del proyecto. Cada uno de los paquetes corresponde a una carpeta.



## Cargar el proyecto en Eclipse

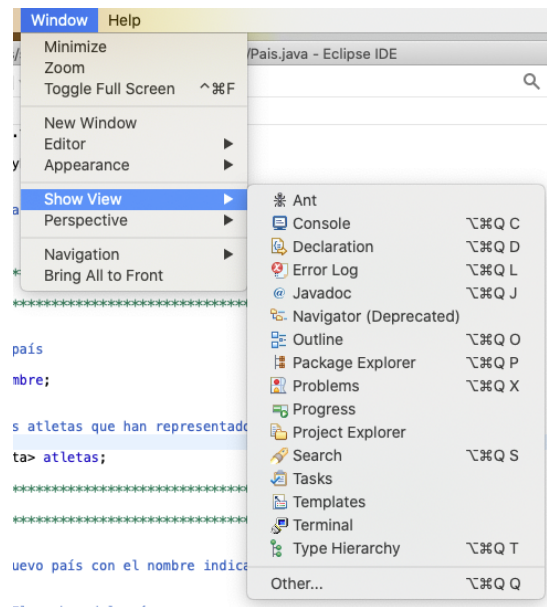
En Eclipse, escoja la opción para importar algo a su espacio de trabajo (File – Import) e indique que quiere importar un proyecto existente.



Seleccione la carpeta donde descomprimió el archivo nTaller1-Gasolinera\_esqueleto.zip y seleccione el proyecto para importar.

Si selecciona la opción “Copy projects into workspace”, los archivos se copiarán al espacio de trabajo de Eclipse para que se utilicen desde allí. En general es mejor tener todos los proyectos de Eclipse dentro del workspace.

En Eclipse, cada uno de los espacios del editor se denomina una vista (view). A través del menú Window – Show View usted puede seleccionar qué vistas quiere tener disponibles en cualquier momento.



Si no ha hecho ningún cambio, usted debería estar viendo en este momento una ventana en la cual aparecen las vistas predeterminadas:

1. Esta vista se denomina el Package Explorer y es el espacio en el cual se puede ver el contenido de los proyectos que actualmente están cargados en el workspace. Cada proyecto tiene su propia estructura interna que estudiaremos a continuación.

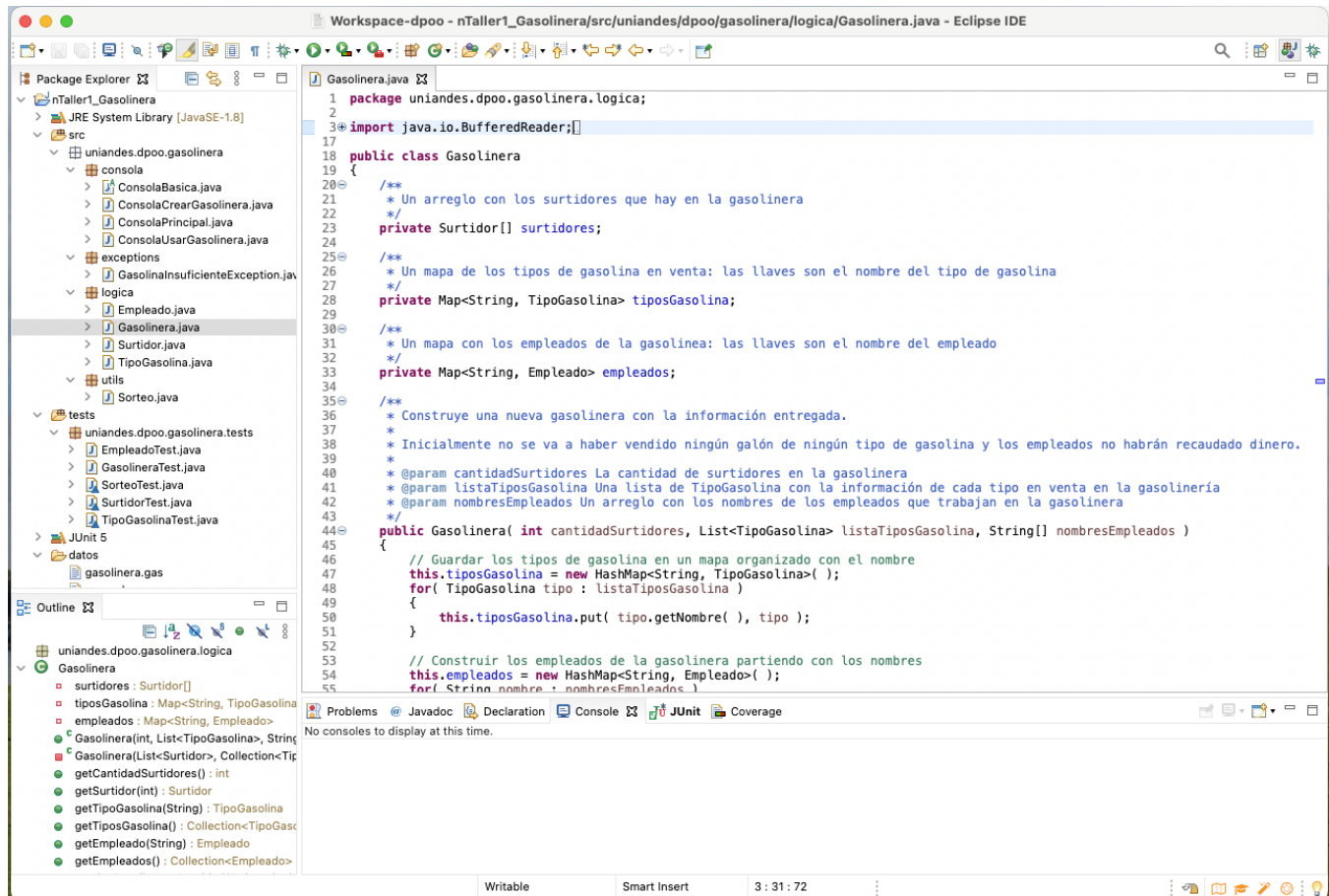
2. Esta vista es un editor de Java. En este caso, estamos viendo el contenido del archivo Gasolinera.java.

3. En la vista de problemas aparece la información de todos los **Errores** y **Warnings** de compilación que haya detectado Eclipse sobre todos los proyectos que estén abiertos en un momento dado. Los errores son problemas que impiden que un proyecto se compile y por tanto tienen que ser solucionados antes de continuar. Los warnings o alertas son problemas que no son serios y que no impiden la compilación. Sin embargo, Eclipse nos avisa de estos problemas potenciales para que los solucionemos lo más pronto posible.

4. La vista de Consola donde se puede ver el resultado de la ejecución de un programa y se puede ingresar información si es necesario.

5. El outline, que muestra la estructura del recurso que tenemos seleccionado en este momento.

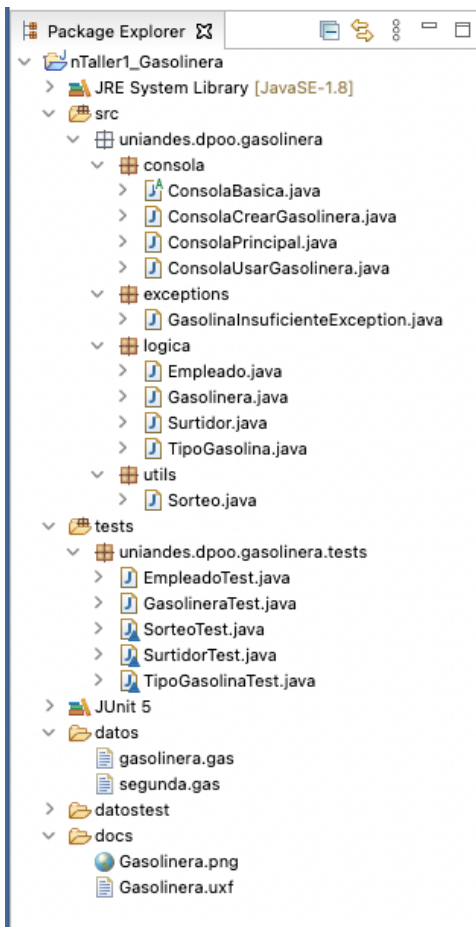
En la captura de pantalla se pueden ver también otras vistas que van a ser importantes como la vista de Junit y la vista de recubrimiento (Coverage)



## Estudiar la estructura del proyecto

La estructura del proyecto que se ve en el explorador de paquetes permite identificar los principales elementos que hacen parte del proyecto. A continuación, explicamos qué significa cada uno de ellos.

1. **nTaller1\_Gasolinera**: este es el nivel superior y corresponde al proyecto entero.
2. **JRE System Library**: esto muestra cuál es la versión del JDK que se está usando para construir y correr la aplicación.
3. **src**: el ícono de esta carpeta nos muestra que esta carpeta está configurada para que su contenido sea compilado (note que el ícono es diferente al de las carpetas **datos** y **docs** que se encuentran más abajo. Es



decir que la carpeta `src` hace parte de lo que Eclipse llama “Build path”: todos los elementos que se utilizan durante el proceso de compilación.

4. `uniandes.dpoo.gasolinera.consola`: el ícono de esta carpeta también es diferente y nos indica que es un paquete, es decir una unidad de Java dentro de la cual se encuentran clases.

5. `ConsolaPrincipal.java`: este es un archivo y tanto el ícono como la extensión nos indican que se trata de un archivo de Java. Lo normal es que dentro de este archivo encontremos una clase llamada `ConsolaPrincipal`.

6. A continuación, encontraremos otros tres paquetes (`exception`, `logica` y `utils`) con los archivos java que cada uno contiene.

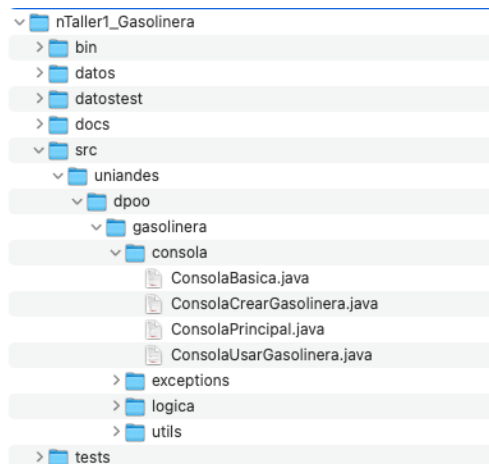
7. `datos`: en esta carpeta tenemos la información de gasolineras que vamos a cargar (el archivo `gasolineras.gas`).

8. `datostest`: en esta carpeta tenemos la información de gasolineras que utilizan las pruebas automáticas para verificar la implementación.

9. `docs`: en esta carpeta tenemos documentación adicional sobre la aplicación.

**Nota importante:** El nombre de una de las clases de la aplicación es `Empleado` y se encuentra en un archivo con el nombre `Empleado.java`. Esto no es una casualidad. Una clase pública **DEBE** declararse en un archivo que tenga exactamente el mismo nombre de la clase y la extensión `.java`. Una consecuencia de esto es que no se pueda tener un archivo en el cual se declaren dos clases públicas.

Finalmente, abra el Explorador de Archivos, Finder o una herramienta equivalente en su computador para observar la estructura de la carpeta `src` de su proyecto. Debería encontrarse con algo similar a lo siguiente:



Esto muestra que los paquetes en Java son equivalentes a las carpetas y que una clase se encuentra en un paquete dependiendo de dónde esté ubicado el archivo dentro de esta estructura.

## Estudiar la estructura de un archivo Java: Gasolinera.java

Estudiemos ahora los elementos que hacen parte de un archivo Java para identificar sus principales componentes. Note que los archivos del proyecto están documentados de forma extensiva para que usted entienda con facilidad el rol de cada elemento.

Estudiaremos el archivo Gasolinera.java, descomponiéndolo para ir explicando algunos de los bloques de instrucciones que lo componen.

```
package uniandes.dpoo.gasolinera.logica;
```

Esta línea declara que el archivo Gasolinera.java está ubicado dentro de la carpeta uniandes/dpoo/gasolinera/logica. Note que, en los paquetes, los nombres de las carpetas se separan usando el carácter ‘.’

**¡Tenga cuidado con los nombres de carpetas!** El nombre de un paquete no puede tener ni espacios ni casi ningún símbolo, así que límitese a letras y números.

**¡Tenga cuidado con los Punto y Coma!** Este caracter se usa en Java para terminar cualquier instrucción, así que lo encontraremos al final de casi cualquier línea.

```
import java.io.File;
import java.io.FileReader;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
```

Estas líneas declaran que en este archivo se utilizarán estas clases definidas dentro de los paquetes ‘java.util’ y ‘java.io’. El mecanismo de importar una clase es muy similar al mecanismo de Python.

```
/**
 * Esta clase tiene la información de una Gasolinera incluyendo los tipos de gasolina,
 * los empleados y los surtidores.
 */
```

Estas líneas sólo tienen un comentario, que en este caso describe el objetivo de la clase Gasolinera. En Java, un comentario largo puede incluirse usando las cadenas /\* y \*/ para delimitar de dónde a dónde debe considerarse que va el comentario.

```
public class Gasolinera
{
```

Estas líneas inician la declaración de la clase Gasolinera, la cual termina al final del archivo con el carácter { . Es importante notar que, en este caso, como en la mayoría, indicamos que la clase Gasolinera va a ser pública.

En Java, todos los bloques de instrucciones se delimitan con la pareja de caracteres { y }, por lo cual la **indentación** ni es obligatoria ni significa nada. La estructura que en Python se construye a través de la indentación, en Java se logra a través del uso de llaves. Sin embargo, es una muy buena práctica indentar el código para facilitar su lectura.

```
// *****
// Atributos
// *****
```

Estas líneas ilustran la otra forma que ofrece Java para introducir un comentario en un programa: con los caracteres // se indica que el resto de la línea es un comentario.



```

/**
 * Un arreglo con los surtidores que hay en la gasolinera
 */
private Surtidor[] surtidores;

/**
 * Un mapa de los tipos de gasolina en venta: las llaves son el nombre del tipo de gasolina
 */
private Map<String, TipoGasolina> tiposGasolina;

```

Estas líneas muestran la definición de dos atributos en la clase `Gasolinera` llamados `surtidores` y `tiposGasolina`. Aquí debemos resaltar, en primer lugar, que los atributos se definen como privados. Esto es muy importante en relación con el concepto de encapsulamiento que estudiaremos más adelante.

En segundo lugar, observe que estamos especificando el tipo de cada uno de estos atributos: los surtidores de la gasolinera están organizados en un arreglo de objetos de tipo `Surtidor` (note los caracteres `[]` después del nombre de la clase); en cambio decimos que `tiposGasolina` es un mapa de elementos de tipo `TipoGasolina`. Si miramos con atención, veremos que `Map` es el nombre de algo que importamos del paquete `java.util` al principio del archivo. `TipoGasolina` no la encontramos importada al inicio, pero como se encuentra en el mismo paquete que nuestra clase `Gasolinera`, podemos utilizarla sin problema.

```

public Gasolinera( int cantidadSurtidores, List<TipoGasolina> listaTiposGasolina, String[]
nombresEmpleados )
{
    ...
}

```

Este bloque de instrucciones corresponde a la declaración de un método constructor de la clase `Gasolinera`. Más abajo encontraremos otros métodos, pero este es especial porque es el que permite construir instancias de una `Gasolinera`. Es decir que, cuando queramos construir una nueva `Gasolinera`, tendremos que invocar este método. Sabemos que este método es un método constructor por dos razones: se llama igual que la clase y no indica el tipo de retorno.

La implementación de este método se encuentra entre los caracteres `{ y }`.

La documentación de este método se encuentra justo antes del método y tiene una estructura que estudiaremos más adelante.

```

/**
 * Retorna la cantidad de surtidores que hay en la gasolinera
 * @return Cantidad de surtidores
 */
public int getCantidadSurtidores( )
{
    return surtidores.length;
}

```

Ahora encontramos la declaración, implementación y documentación del método `getCantidadSurtidores`. A diferencia del método anterior, este método no es un método constructor (no se puede usar para crear nuevas gasolineras) y en cambio debe utilizarse para preguntarle a una gasolinera por la cantidad de surtidores que tiene. La signatura del método ( `public int getCantidadSurtidores()` ) nos indica varias cosas importantes:

- El método es público, para que pueda ser invocado desde otras clases.
- El método retorna algo de tipo `int`.
- El método se llama `getCantidadSurtidores`.
- El método no espera ningún parámetro.

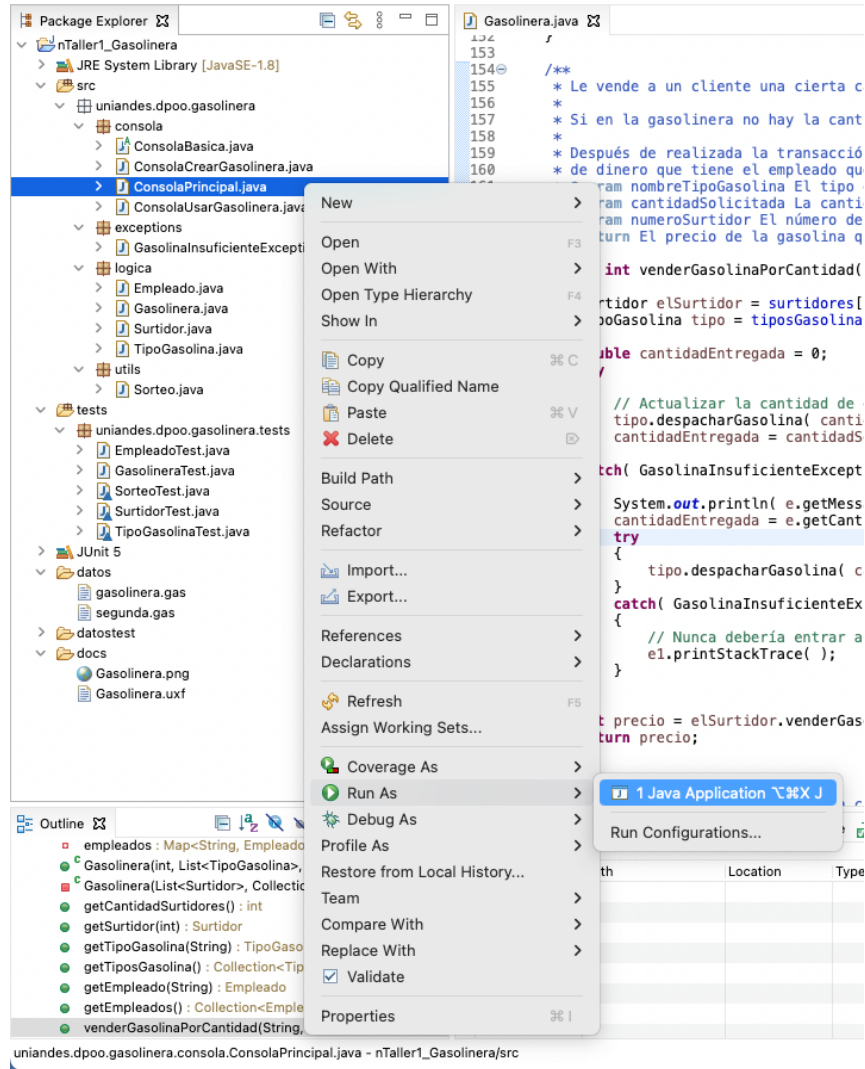


## Correr la aplicación

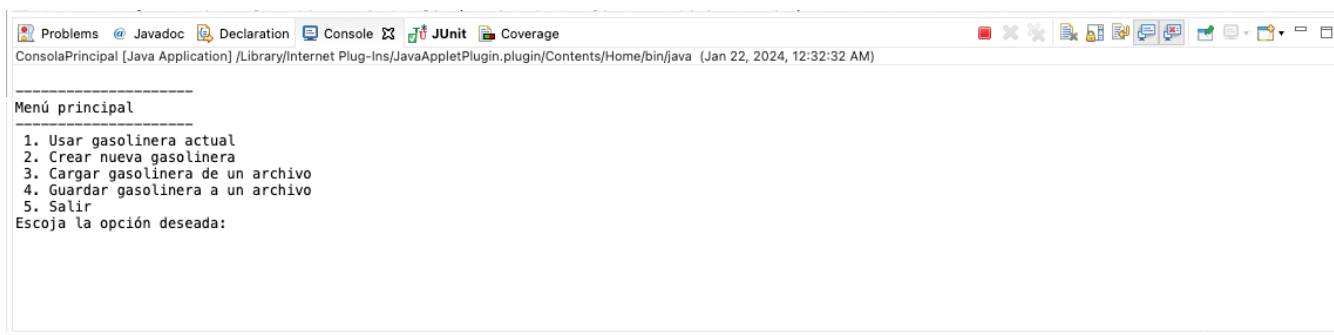
Ubique el archivo con la clase `ConsolaPrincipal` y ábralo. Al final del archivo encontrará un método con la signatura:

```
public static void main(String[] args)
```

Esta es la signatura que **DEBE** tener el método `main` de una aplicación Java, es decir el método que se puede utilizar para lanzar una aplicación. Ahora haga click derecho sobre el archivo que contiene la clase y seleccione la opción para correrlo como una aplicación Java.



La aplicación debería empezar a correr y mostrar los siguientes mensajes en la consola:



```
Problems Javadoc Declaration Console JUnit Coverage
ConsolaPrincipal [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java (Jan 22, 2024, 12:32:32 AM)

Menú principal
1. Usar gasolinera actual
2. Crear nueva gasolinera
3. Cargar gasolinera de un archivo
4. Guardar gasolinera a un archivo
5. Salir
Escoja la opción deseada:
```

Pruebe algunas de las opciones y finalmente seleccione la opción 5 para salir, o haga click en el cuadro rojo de la parte superior derecha de la consola para terminar la ejecución del programa.

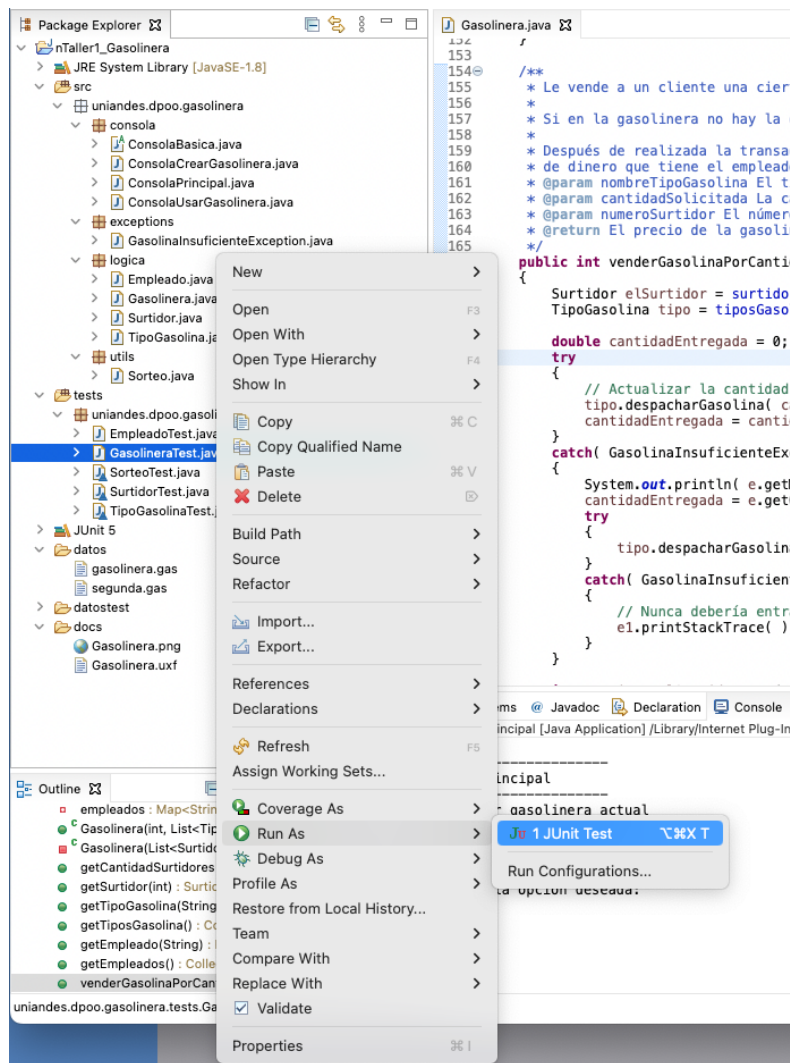
Note que, si hace click derecho sobre una clase que no tenga un método `main`, no le aparecerá la opción para correr la clase como una aplicación Java.

Puede volver a correr la clase `ConsolaPrincipal` haciendo click sobre el botón que se encuentra en el menú superior, el cual correrá el último programa ejecutado.

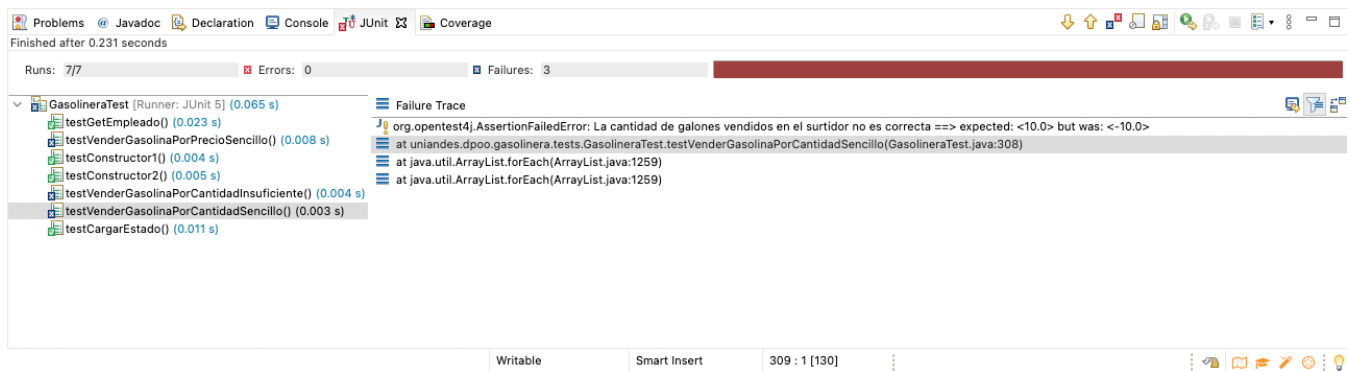


## Correr la aplicación

Ubique el archivo con la clase `GasolineraTest`. Este archivo no se encuentra dentro de la carpeta `src` sino dentro de la carpeta `tests`, la cual contiene otro conjunto de clases que se están usando para verificar que la aplicación haya quedado bien implementada. Corra la clase `GasolineraTest`, pero no como una aplicación independiente sino como un conjunto de pruebas, seleccionando la opción “JUnit Test”



Al correr las pruebas, debería abrirse la vista con el título JUnit. Si no es así, ábrala y revise su contenido – que debería verse similar a la siguiente imagen.



En esta vista se puede ver que dentro de la clase GasolineraTest se implementaron 7 pruebas y que al correrlas 4 fueron exitosas (las que están en verde) mientras que 3 presentaron algún problema (las que tienen un ícono azul). En la sección Failure Trace puede verse un mensaje que explica cuál fue el problema que se evidenció y al hacer click sobre la línea siguiente se puede ir a la línea exacta donde se presentó el error dentro del programa de pruebas.

## Parte 3: Modificar el programa

En esta parte ahora usted tendrá que modificar el programa.

### Modificación 1: Corregir los errores del programa

Corra todas las pruebas de la aplicación. Para esto, en lugar de seleccionar cada una de las clases de prueba, puede seleccionar el paquete que contiene todas las pruebas.

Revise el resultado de las pruebas y corrija los errores que se presenten, hasta que logre que todas las pruebas sean exitosas. NO DEBE MODIFICAR el código de las pruebas.

### Modificación 2: Utilizar la aplicación

Después de haber corregido los errores, usted debe utilizar la aplicación para crear un archivo para una gasolinera: ejecute la consola principal y ejecute todas las acciones necesarias para crear una nueva gasolinera, utilizarla y finalmente salvarla. La nueva gasolinera debe tener 4 tipos de gasolina, 5 empleados y 4 surtidores.

El archivo con su nueva gasolinera debe quedar dentro de la carpeta `datos` en el momento de la entrega.

## Entrega

GitHub es una plataforma en la nube que permite administrar un proyecto de software, bien sea que se trabaje de forma individual o en grupo. En este curso, las entregas de talleres y proyectos deben realizarse a través de esta plataforma.

Para continuar con el laboratorio deben tener una cuenta en GitHub, en caso de no poseerla deben registrarse **con su correo de Uniandes** en la plataforma utilizando el enlace <https://github.com/join>. Si ya tienen una cuenta personal con otro correo, debe crear una nueva cuenta con el correo de la Universidad o cambiar el correo de registro de que ya existe.

Construya un repositorio **público** nuevo para el laboratorio y deje en este todo el trabajo que haya realizado para el taller.

Entregue a través de Bloque Neón el URL del repositorio en la actividad designada como “**Taller 1**”.