PRINCIPIOS DE DESARROLLO DE SOFTWARE

2019-2020

Universidad Carlos III de Madrid

Ejercicio Guiado - 2

2019/2020

NORMATIVAS, PROPIEDAD COLECTIVA DE CODIGO, INTEGRACIÓN AUTOMATIZADA

Ejercicio Guiado 2

Universidad Carlos III de Madrid. Escuela Politécnica Superior

Javier García Guzmán

Sección 1

Objetivos

Este ejercicio guiado tiene como propósito:

- Establecer un estándar de codificación, aceptado e implantado por todo el equipo, para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada.
- Aplicar ese estándar de código utilizando una herramienta sobre el entorno integrado de desarrollo.
- Poner en práctica los principios básicos de integración automatizada.
- Aprender las buenas prácticas de propiedad colectiva de código que será necesario aplicar durante el desarrollo de un proyecto siguiendo los principios de las metodologías ágiles.

Necesidades acerca del componente de software a desarrollar:

Transport4Future es una empresa que está especializada en el desarrollo de tecnologías para vehículos autónomos.

Como parte de los mecanismos de seguridad que se deben implementar en un vehículo autónomo, es necesario asegurar que solamente sensores y actuadores autorizados pueden proporcionar la información necesaria para la percepción del vehículo y para la ejecutar las órdenes de conducción.

A fin de garantizar que solamente sensores y actuadores debidamente autorizados y registrados pueden formar parte de la red de un vehículo autónomo, se ha decidido desarrollar un componente de software que emita las autorizaciones (tokens) a los sensores y actuadores previamente registrados en la configuración operativa del vehículo.

Los ingenieros de software de Transport4Future comenzaron la tarea de implementar dicho componente, pero dadas las dificultades del mismo decidieron lanzar un contrato de desarrollo a un grupo de ingenieros de software independientes (vosotros), por una suma tan grande de *dinero* como para permitir un lujoso retiro en las playas del Caribe.

El sistema de seguridad de conexión de dispositivos a la red de un vehículo autónomo debe proporcionar los mecanismos necesarios para generar tokens de acceso a la red del vehículo a partir de datos en formato JSON que sirvan para verificar que un dispositivo está autorizado a conectarse a la red del vehículo.

Este componente se desarrollará en Java, plataforma J2EE, y se entregará en forma de un JAR que proporcione acceso a los métodos de la interfaz de programación que permita su integración en los servicios de verificación implementados en los distintos vehículos autónomos. Asimismo, se proporcionará un manual de usuario que permita a otros programadores realizar las tareas de integración de este componente en los sistemas mencionados.

Sección 2

Pasos para aplicar los principios de propiedad colectiva de código

GitLab es un servicio de control de versiones on-line para facilitar la colaboración en proyectos de desarrollo de software ofreciendo herramientas para la gestión de requisitos, diseño, control de código fuente, integración continua, pruebas, y seguimiento de proyectos. Esta sesión se centrará en las funcionalidades para la gestión de cambios en el código fuente (control de versiones) y la integración continua de soluciones.

Herramientas para utilizar en la práctica

GITLAB

GitLab es una plataforma que es capaz de unificar en un solo lugar la mayoría de las necesidades actuales de los desarrolladores de software. Posee versiones tanto públicas que permiten alojar nuestros repositorios en sus servidores, como descargar una copia del software y desplegarlo en nuestros propios servidores. Este es el caso en el que nos encontraremos a lo largo del ejercicio.

El servidor al que nos referimos en este caso se encuentra alojado dentro del campus, y se ha dotado al mismo de acceso desde direcciones IPv4 correspondientes al ámbito geográfico español.

La plataforma dispone de acceso a través de una web, accesible desde la siguiente URL: https://pds.sel.inf.uc3m.es

Al acceder a esta página, se deberá crear, en primera instancia, un usuario por alumno de la asignatura. En clase se describirá el procedimiento.

SOURCETREE

Como cliente GIT se recomienda el uso de SourceTree. Dicho cliente se encuentra instalado en las aulas. SourceTree presenta una versión para Windows y otra para MacOS. En el caso de Linux se puede utilizar otro cliente gráfico similar, como GitKraken. No obstante, siempre está disponible la opción de trabajar (en cualquiera de los sistemas operativos mencionados) desde la línea de comandos.

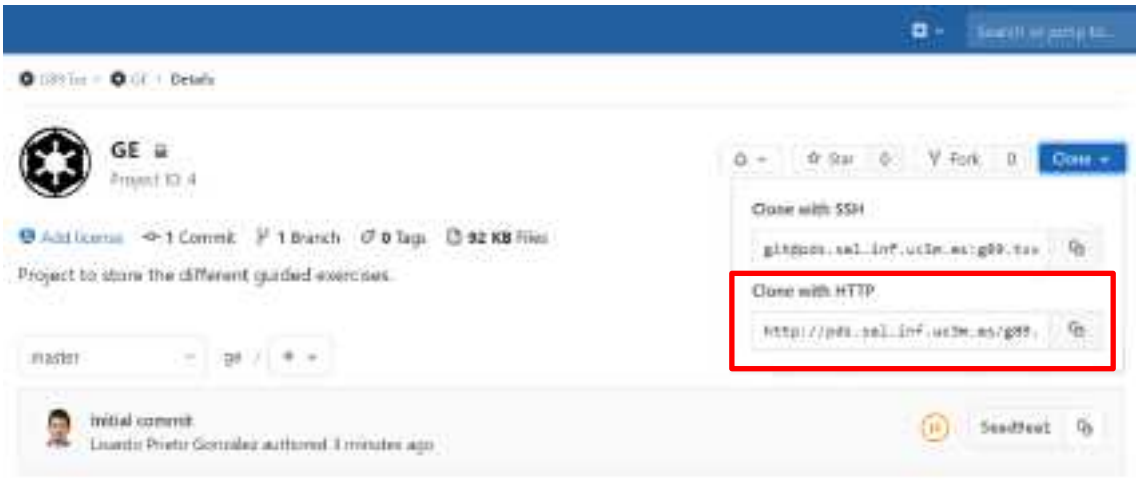
ECLIPSE

Como IDE se empleará ECLIPSE, también instalado en las aulas. Se permite utilizar cualquier otro entorno en el que el equipo de desarrollo se sienta a gusto y sea de su preferencia, sin embargo, el soporte de herramientas que proporcionaremos será únicamente para este IDE.

Paso 1: Preparar el ambiente de trabajo y proyecto en GitLab

El primer paso será vincular el repositorio en GitLab que contendrá los ejercicios guiados y la práctica final. Estos repositorios estarán asignados a cada grupo de prácticas, y se pueden consultar accediendo a través de la interfaz web de GitLab.

A efectos de trabajo, la configuración de SourceTree se realizará empleando el protocolo de comunicación SSH. La dirección a utilizar en SourceTree se podrá consultar desde la página de cada repositorio, desde el sitio web de GitLab. Por ejemplo:



Para establecer comunicación mediante SSH primero es necesario generar y agregar la clave pública de cada usuario en su correspondiente perfil (dentro de GitLab), estas claves públicas serán las que autentifiquen a un cliente como SourceTree como autorizado a subir datos en nombre del desarrollador, a los proyectos en los que este participe.

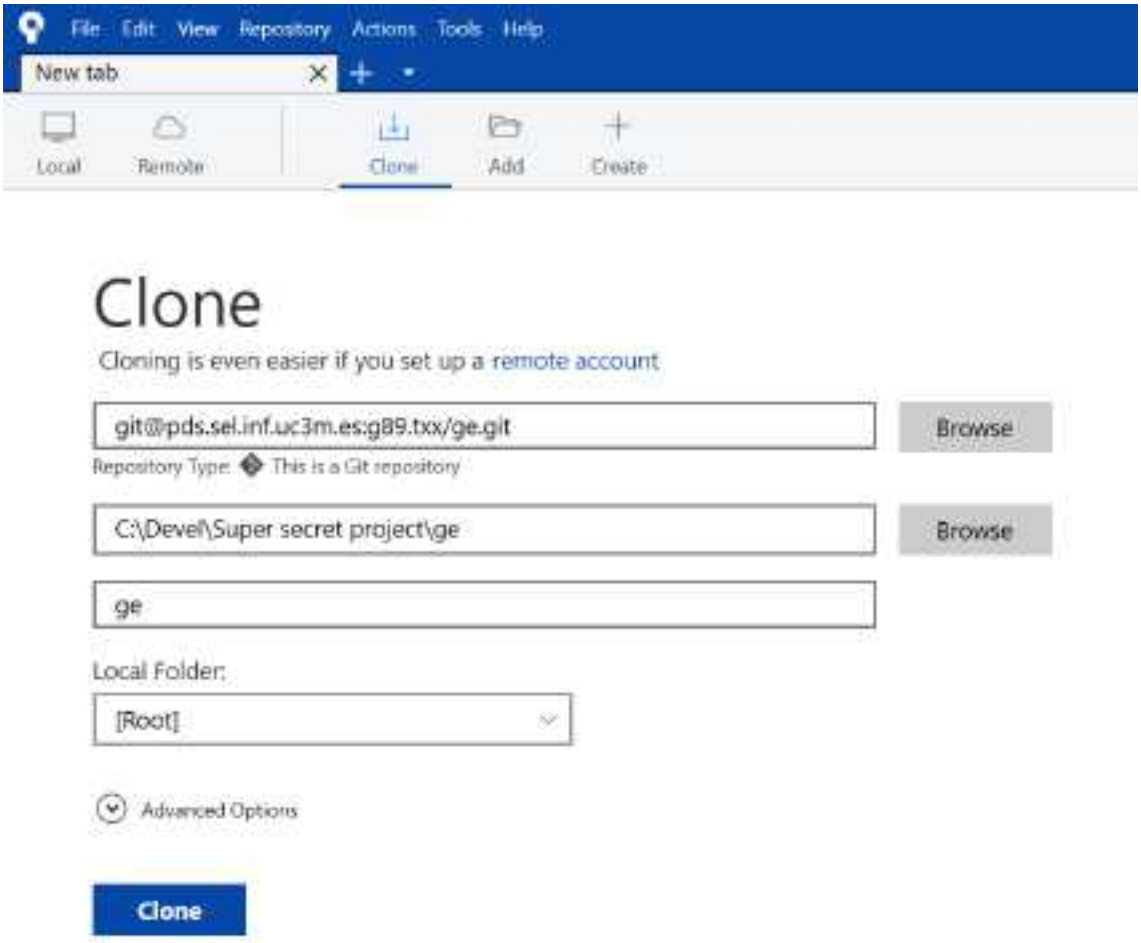
Las claves SSH generadas deberán ser agregadas en la sección correspondiente de GitLab, accesible en https://pds.sel.inf.uc3m.es/profile/keys . Las indicaciones para generar dichas claves se pueden encontrar en https://docs.gitlab.com/ee/ssh/ . Asimismo, en sistemas Windows se puede recurrir al uso de la herramienta gratuita y de código abierto **PuTTYgen** (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html). Tutorial de uso en https://www.ssh.com/ssh/putty/windows/puttygen

En caso de utilizar SourceTree en Windows, se pedirá el agregar la clave privada al programa **Pageant** (*PuTTY authentication agent*) que es instalado junto con SourceTree para realizar la comunicación mediante SSH empleando el esquema de autenticación clave-pública-privada. Tras configurar esto se puede proceder con los siguientes pasos. Las claves SSH-RSA generadas son **únicas y vinculantes**, por lo que cada miembro del equipo tendrá su propia clave para poder subir las soluciones a sus repositorios asignados. No es necesario que ambos miembros suban la solución, ya que se sobreentiende que es un equipo de trabajo y puede colaborar tanto un miembro como otro en un momento determinado.

Paso 2: Clonar el repositorio para empezar a trabajar

En la vista principal del proyecto se pueden encontrar las indicaciones pertinentes para poder empezar a utilizar el repositorio y si se han seguido los pasos previos correctamente ya estará listo para usar. Se recomienda el uso de SourceTree (u otro cliente gráfico) ya que dispone de asistentes de configuración para el clonado (y gestión) de los repositorios. Toda la información necesaria para cualquiera de los casos estará siempre disponible en la página principal de cada proyecto.

En SourceTree bastará con acceder al menú "**File → Clone/New (Ctrl+N)**" e introducir los datos correspondientes al proyecto. Para el caso correspondiente a la anterior captura de pantalla:

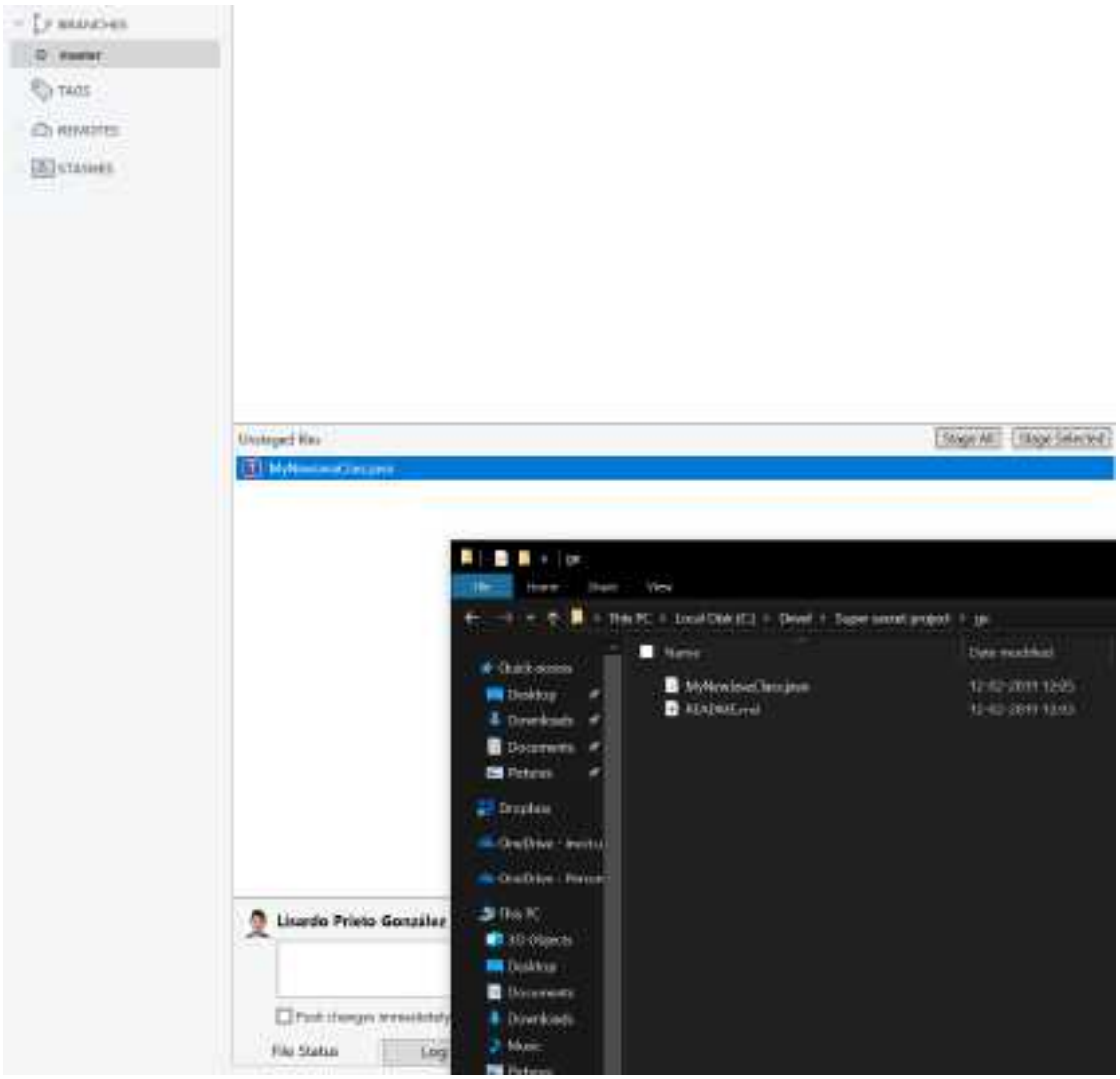


Tras introducir la URL del proyecto (accesible como puede verse en la primera captura de pantalla), se seleccionará el directorio de destino (directorio de trabajo, donde se copiarán los archivos del proyecto y las soluciones), y posteriormente se pulsará el botón "**Clone**".

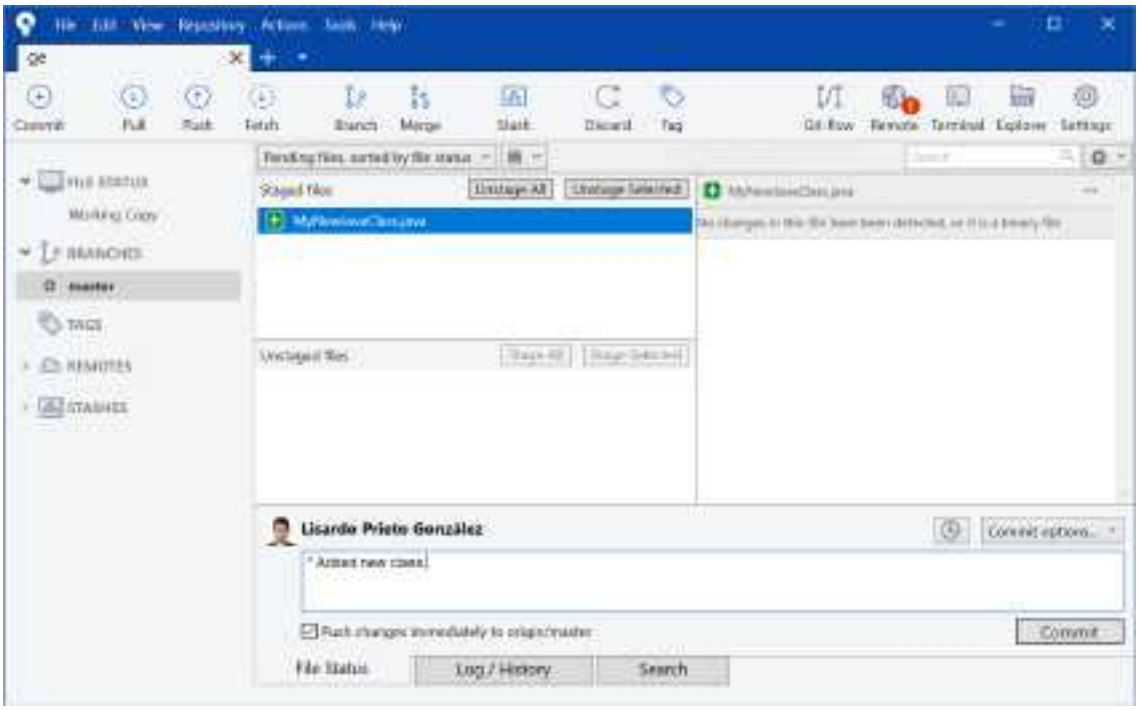
Tras este punto, el repositorio local ya se encuentra enlazado con el remoto y por consiguiente cualquier cambio que hagamos en los archivos locales será registrado y propagado a todas las versiones en el momento de subirlo/descargarlo en otro computador enlazado al proyecto.

Paso 3: Agregar código al repositorio

En este momento se puede subir el código al servidor. Para ello, se puede abrir el Explorador de archivos del sistema operativo en la ruta donde se ha descargado el repositorio (directorio de trabajo) y se copiarán los ficheros deseados dentro de dicho directorio. A partir de este momento, SourceTree detectará la existencia de datos nuevos (o modificados si se han cambiado ficheros existentes) en el directorio del proyecto, y propondrá subir los cambios pendientes (pestaña inferior "**File Status**").

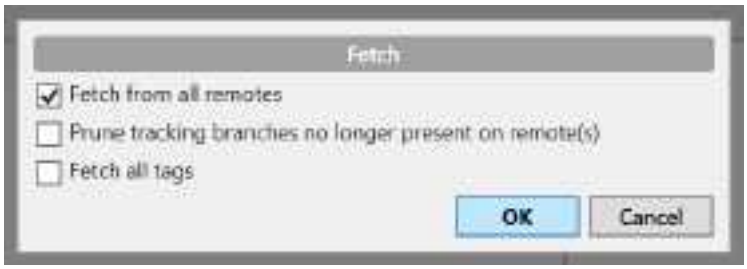


Para completar este paso, será necesario pulsar el botón "**Stage All**" en la sección "**Unstaged files**", opcionalmente escribir una breve descripción de los cambios llevados a cabo en el cuadro de texto inferior y finalmente pulsar el botón "**Commit**".

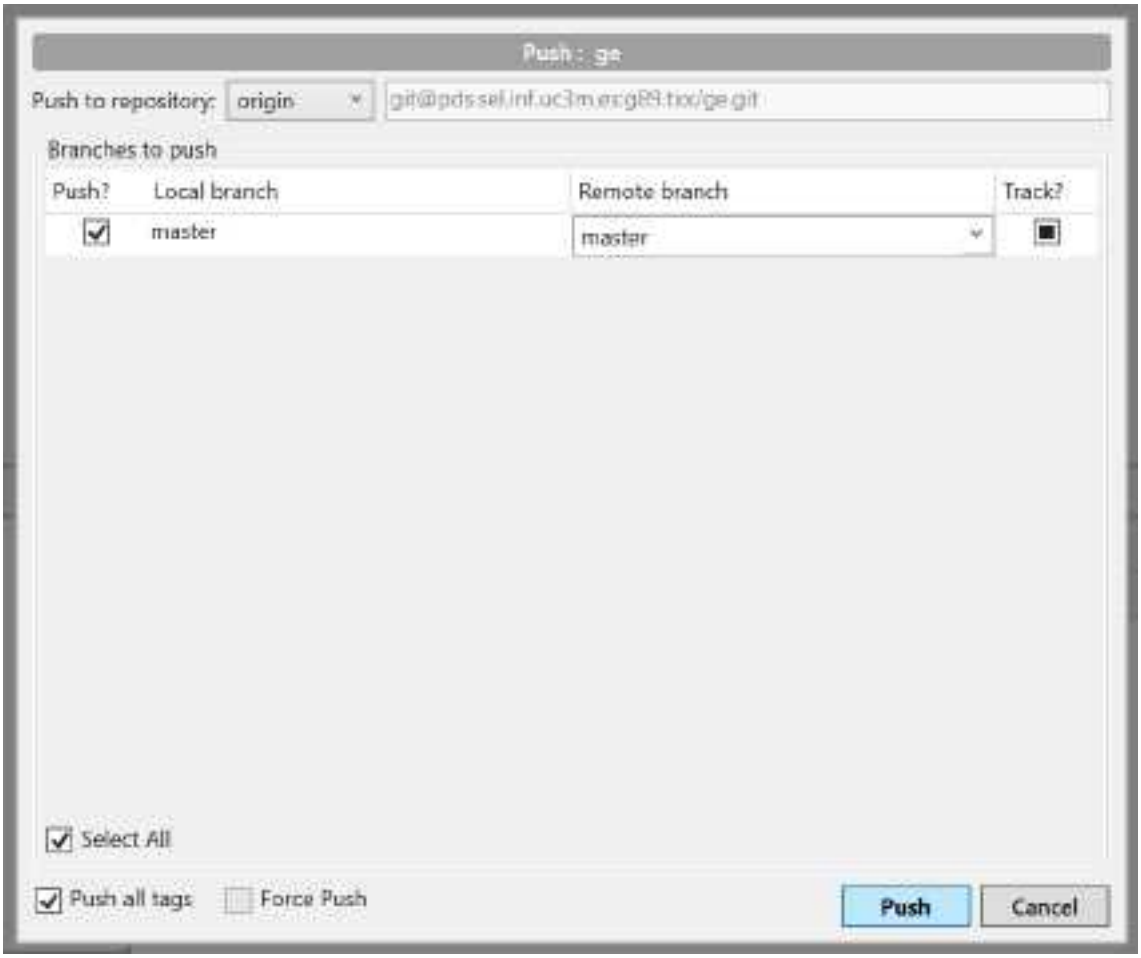


Importante: **GIT** es un sistema de control de versiones descentralizado, por lo que sería posible estar trabajando con diversas máquinas conectadas al mismo repositorio trabajando todas en paralelo, y con sus propias copias del repositorio en almacenamiento local. Para hacer efectivos los cambios es necesario enviarlos al servidor (en el presente caso) GitLab.

Una buena práctica es realizar siempre un "**Fetch**" como primer paso, a fin de determinar si ha habido cambios en el repositorio antes de proceder a hacer un "**Push**" (sube los cambios a los que se han aplicado el "**Commit**" al repositorio del servidor). En caso de existir cambios, estos deberán ser descargados con un "**Pull**", y fusionar los cambios obtenidos del servidor remoto con los cambios locales a través de un nuevo "**Commit**" y finalmente proceder con un "**Push**" al servidor.




File Edit View Repository Actions Tools Help
ge
Commit
Pull
Push
Fetch
Branch
Merge
Stash
Discard
Tag
All Branches
Show Remote Branches
Date Order
Graph
FILE STATUS
Working Copy
BRANCHES
master
TAGS
REMOTES
STASHES
master
* Another class.
origin/master
origin/HEAD
* Added new class.
Initial commit


Push : ge
Push to repository:
origin
Branches to push
Push?
Local branch
Remote branch
Track?
master
master
Select All
Push all tags
Force Push
Push
Cancel

En cada uno de los PCs del integrante del equipo de desarrollo que tenga SourceTree vinculado contra el repositorio se recibirá la notificación de que hay nuevos cambios en el servidor que deben ser descargados. Como se ha recomendado anteriormente, será conveniente hacer un "**Fetch**" seguido de un "**Pull**" para actualizar la versión local antes de continuar con el desarrollo de las tareas del proyecto.

La versión incompleta del componente (*Licensing*), utilizada como medio de partida, se podrá descargar del entregador de Aula Global. El componente es un proyecto de Eclipse, comprimido en un fichero de nombre "Empire_Licensing_v1.0.zip". Se recomienda su extracción en el mismo directorio utilizado en el punto "*Paso 3: Agregar código al repositorio*" y realizar un primer "**Commit**" + "**Push**" del mismo. A partir de ahí, se continuará con el desarrollo del ejercicio guiado.

Sección 3

Configuración de un script de integración automatizada con Maven

Maven se basa en el concepto de Modelo de Objetos de Proyecto (POM) el cual permite que dichos objetos (dependencias, meta data del proyecto y plugins) puedan ser explícitamente declarados para estandarizar la construcción y empaquetado de proyectos Java que en conjunto forman una aplicación o sistema. El archivo de configuración que maven utiliza para este propósito es llamado ***pom.xml***.

Estructura básica de un archivo pom.xml para efectos prácticos la podemos dividir en tres partes:

1. Información general de nuestro proyecto

   Contiene la información que hace único al artefacto que el proyecto genera, la información necesaria es:

   - groupId Identificación del grupo al que pertenece el proyecto.
   - id Nombre de nuestro proyecto.
   - packaging Tipo de empaquetado (por ejemplo jar o war).

2. Sección de dependencias

   Contiene la lista de dependencias que el proyecto utilizará. En este punto podemos incluso indicar el scope (ámbito) de cada dependencia, por ejemplo indicar que la dependencia Junit no forme parte del empaquetado final de nuestro proyecto pero que sí esté presente durante la fase de la ejecución de pruebas unitarias.

3. Sección de plugins

Contiene los plugins que extienden las funcionalidades básicas de maven. Esta característica ha hecho muy popular a maven dado que oficialmente da soporta a más de 50 plugins y la lista de desarrollados por terceros es amplia. Existen plugins para empaquetado, generación de reportes, análisis estático de código etcétera.

Las partes del ciclo de vida principal del proyecto Maven son:

1. compile: Genera los ficheros .class compilando los fuentes .java
2. test: Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
3. package: Genera el fichero .jar con los .class compilados
4. install: Copia el fichero .jar a un directorio de nuestro ordenador donde maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos maven en el mismo ordenador.
5. deploy: Copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de los comandos maven, por ejemplo, si ejecutamos mvn install, maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando solo aquellas que no se hayan ejecutado previamente.

Los pasos que se deben seguir para crear un script de integración automatizada con Maven se describen a continuación

Paso 1: Creación del Proyecto

En este paso, iniciará Eclipse y creará un proyecto de Maven. Añadirá las dependencias necesarias y creará el proyecto. La compilación producirá un archivo .jar, que es el paquete de implementación.

1. Cree un nuevo proyecto de Maven en Eclipse.
   a) En el menú File, elija New, seguido de Project.
   b) En la ventana New Project, elija Maven Project.
   c) En la ventana New Maven Project, elija Create a simple project y deje el resto de selecciones con sus valores predeterminados.

d) En las ventanas New Maven Project y Configure project, escriba la siguiente información para Artifact:
   - Group Id: Transport4Future
   - Artifact Id: TokenManagement
   - Version: 0.0.1-SNAPSHOT
   - Packaging: jar

Paso 2. Configurar las propiedades de Maven

A veces, es necesario compilar un proyecto con una versión específica. El comando *javac* puede aceptar dicho comando usando *-source* y *-target*. El complemento del compilador también se puede configurar para proporcionar estas opciones durante la compilación.

Por ejemplo, si desea utilizar las funciones de lenguaje Java 8 *(-source 1.8)* y también desea que las clases compiladas sean compatibles con JVM 1.8 *(-target 1.8)*, puede agregar las siguientes propiedades:

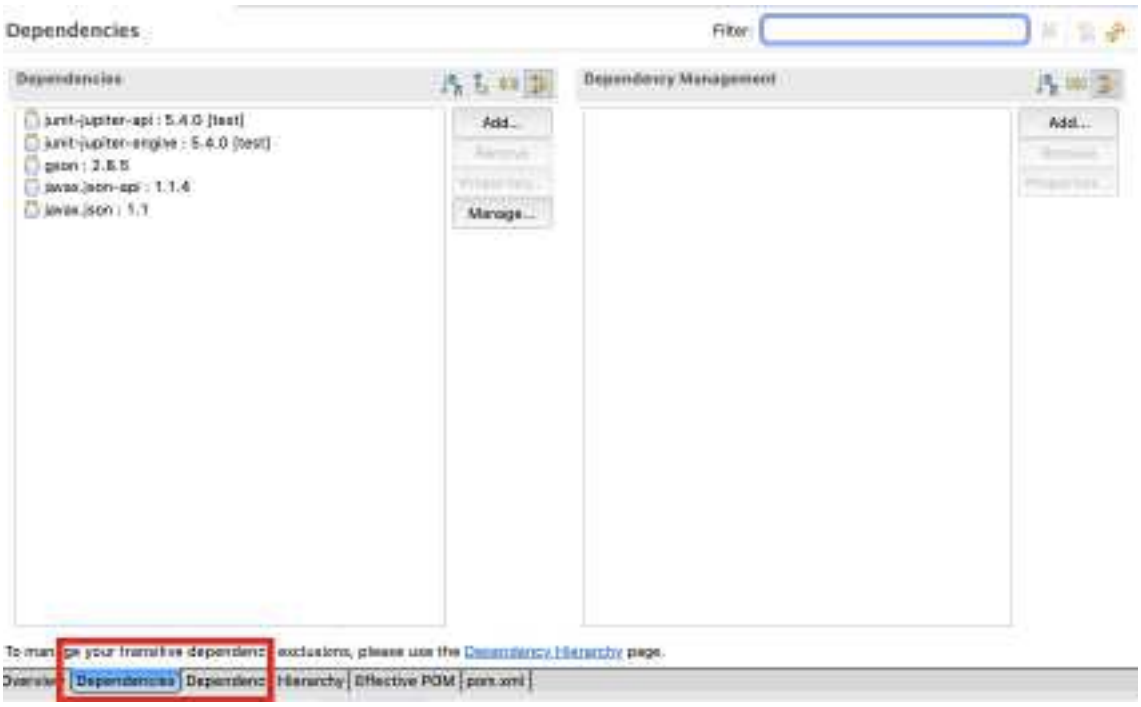
```
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
```

Paso 3. Añadir las dependencias

En este paso será necesario añadir las dependencias a las librerías externas que se utilizan en la complicación, construcción y empaquetado del componente.

a) En primer lugar, es necesario añadir las dependencias al framework de JUnit 5 que permitirá la ejecución de las pruebas unitarias como primer paso de la integración automatizada después de la compilación.

   Para ello será necesario pulsar en la pestaña dependencias del Pom.xml y pulsar el botón Add.



Para facilitar la compilación y ejecución de los casos de prueba es necesario contemplar las siguientes dependencias:

- junit-jupiter-api. Los valores a colocar son:
  - groupId: org.junit.jupiter
  - artifactId: junit-jupiter-api
  - versión: 5.4.0
  - scope: test; este valor indica que esta dependencia solo se tiene en cuenta para compilar y ejecutar las pruebas unitarias.
- junit-jupiter-engine. Los valores a colocar son:
  - groupId: org.junit.jupiter
  - artifactId: junit-jupiter-engine
  - versión: 5.4.0
  - scope: test

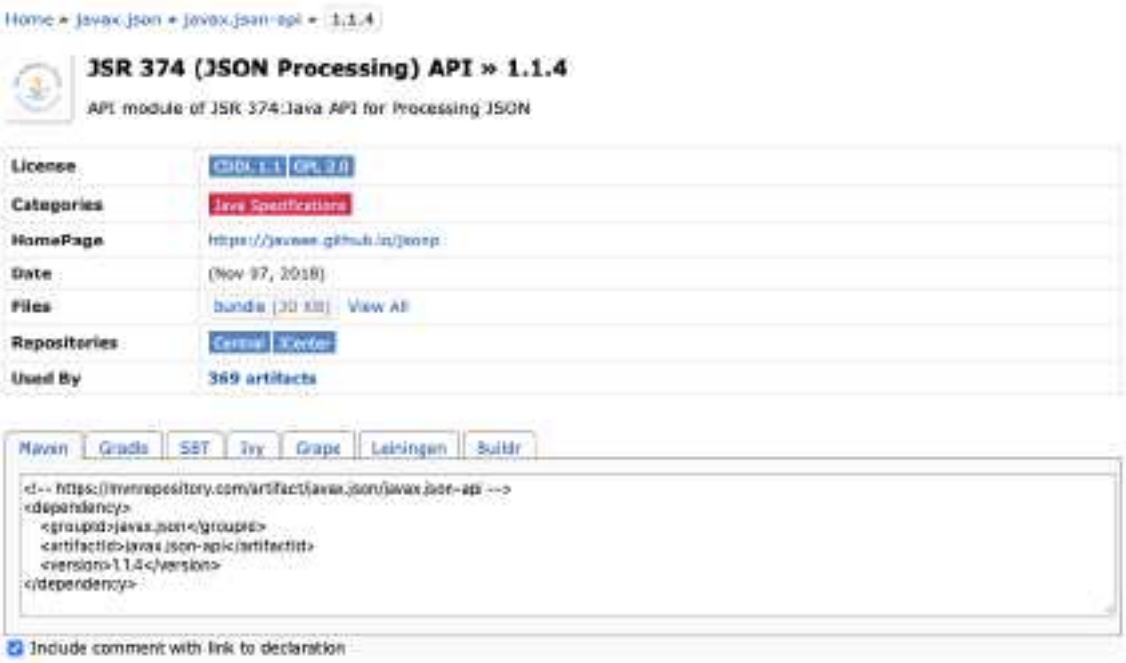
b) En segundo lugar, es necesario añadir las dependencias a las librerías externas que se utilizan para el código funcional.

En este caso, se utilizan dos librerías externas javax.json (librería utilizada para parsear los ficheros JSON de entrada y google gson (librería utilizada para almacenar los datos internos de licencias emitidas en un fichero JSON).

Para encontrar los valores que se deben proporcionar para cada una de las librerías consideradas en Maven, es necesario consultar su repositorio - http://mvnrepository.com.

Los datos de las librerías a incorporar son las siguientes:

- Javax.json



- org.glassfish:


Search for groups, artifacts, categories
Search
Home » org.glassfish » javax.json » 1.1.4
JSR 374 (JSON Processing) Default Provider » 1.1.4
Default provider for JSR 374:Java API for Processing JSON
License
CDDL 1.1 GPL 2.0
HomePage
https://javaee.github.io/jsonp
Date
(Nov 07, 2018)
Files
bundle (125 KB) View All
Repositories
Central jCenter
Used By
696 artifacts
Maven Gradle SBT Ivy Grape Leiningen Buildr
<!-- https://mvnrepository.com/artifact/org.glassfish/javax.json -->
<dependency>
<groupId>org.glassfish</groupId>
<artifactId>javax.json</artifactId>
<version>1.1.4</version>
</dependency>

- Google gson:


Search for groups, artifacts, categories
Search
Home » com.google.code.gson » gson » 2.8.5
Gson » 2.8.5
Gson
License
Apache 2.0
Categories
JSON Libraries
Date
(May 22, 2018)
Files
jar (235 KB) View All
Repositories
Central Pivotal
Used By
9,655 artifacts
Maven Gradle SBT Ivy Grape Leiningen Buildr
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
<groupId>com.google.code.gson</groupId>
<artifactId>gson</artifactId>
<version>2.8.5</version>
</dependency>

Paso 4. Configurar el empaquetado del componente en un JAR

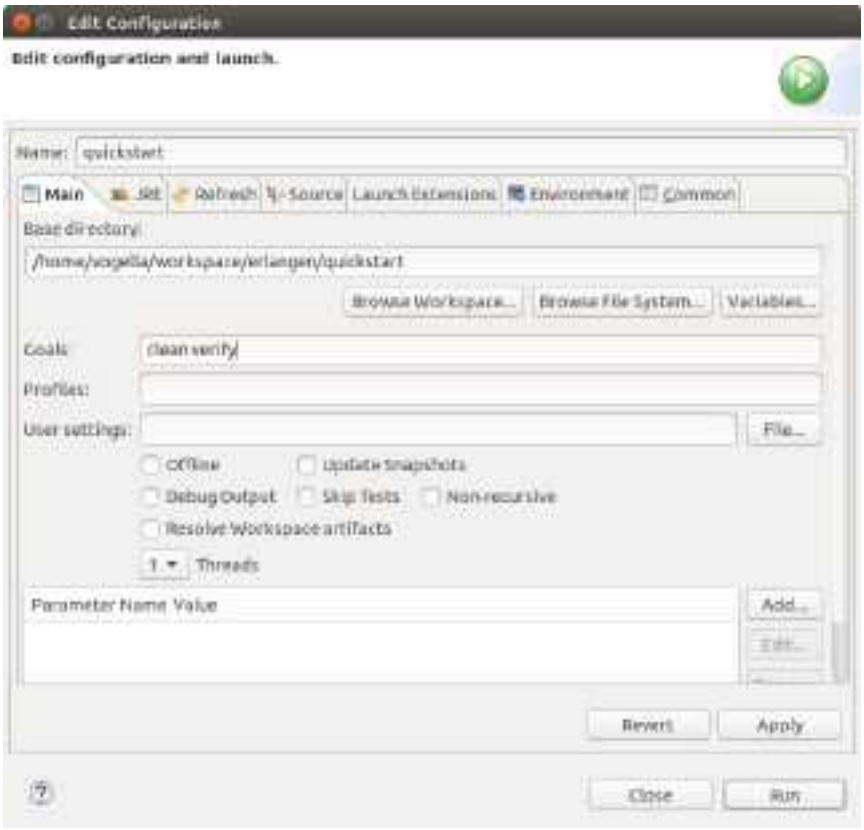
Para que una integración automatizada sea satisfactoria, es necesario que empaquete en un único fichero, el código objeto y las librerías que se utilizan para la correcta ejecución del mismo. Para ello, es necesario incluir en el fichero Pom.xml el siguiente complemento:

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-assembly-plugin</artifactId>
<executions>
<execution>
         <phase>package</phase>
                  <goals>
                  <goal>single</goal>
                  </goals>
                  <configuration>
                  <archive>
                           <manifest>
                                    <mainClass>
                                             Transport4Future.TokenManagement
                                    </mainClass>
                           </manifest>
                  </archive>
                  <descriptorRefs>
                           <descriptorRef>jar-with-dependencies</descriptorRef>
                  </descriptorRefs>
                  </configuration>
</execution>
</executions>
</plugin>
```

Paso 5. Probar la generación del código con Maven

En este momento, es necesario verificar que la configuración de generación funciona correctamente ejecutando la compilación. Para ello, haga clic con el botón derecho en el archivo pom.xml y seleccione Ejecutar como compilación Maven.

Esto abre un diálogo que permite definir los parámetros para el inicio. Escriba *package* en el campo Objetivos: y presione el botón Ejecutar (Run).



Paso 6. Registro del código funcional en Git

Al finalizar la sesión de trabajo se deberá realizar en GIT la ejecución los comandos *Push* y *Commit* para el correcto registro del trabajo realizado, de acuerdo con las instrucciones proporcionadas en la sección 3.

Sección 4

Pasos para establecer una normativa de codificación

Al comenzar un proyecto de software es necesario establecer un estándar de codificación, aceptado e implantado por todo el equipo, para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Se pretende que este estilo ayude a construir programas correctos, entendibles y fáciles de mantener. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

A la hora de establecer un estándar de codificación se debe distinguir entre normas y recomendaciones. Una "norma" es una regla que debe cumplirse obligatoriamente, mientras que una "recomendación" es una regla válida en general pero que puede admitir o requerir excepciones.

Los pasos que se indican a continuación son orientativos. El grupo es libre de añadir más elementos al estándar de codificación o de prescindir de los que considere innecesarios.

Paso 1: Estándar para la Organización de Archivos

Archivos de Código Fuente

✓ Se deberá establecer una norma para determinar el formato de leyenda de derechos de autor que deberán tener todos los archivos. Por ejemplo, se puede incluir en el código fuente una línea comentada donde figuren la fecha, el autor y la compañía a la que pertenece la persona que desarrolló dicho código.

✓ Se deberá establecer una norma que determine de qué forma se gestionarán todos los ficheros fuente, al igual que el resto de los ficheros de interés (ejecutables, bibliotecas de funciones, etc.). En las prácticas de la asignatura, el control de versiones de los ficheros se gestionará a través del servicio GitLab desplegado a tal efecto para la asignatura.

Ficheros de Clases

✓ Se deberá establecer si se emplea o no un fichero distinto para cada clase, así como el nombre que tendrán dichos ficheros.

✓ Se deberá establecer si se incluye una cabecera en cada fichero de clase que aporte información resumida de la misma, utilización y precauciones a tener en cuenta para facilitar su revisión/actualización posterior.

Paso 2: Estándar para Nombres y Variables

Clases y Miembros de Clases

✓ Se establecerán normas para el nombrado de las clases, interfaces y tipos de datos definidos por el desarrollador.

✓ Se deberán establecer normas para el nombrado de los campos, métodos, propiedades y para las constantes.

Visibilidad

✓ Se deberá establecer si las variables (de instancia o estáticas) podrán ser públicas o se usarán en su lugar propiedades.

✓ Se deberá establecer el formato del texto que compone el código fuente. Por ejemplo, se deberá plantear si cada declaración se corresponde con una línea y si las declaraciones e inicializaciones consecutivas deberán estar alineadas (tipo tabla).

✓ Se deberá establecer dónde se inicializan los campos o variables. Habrá que tener en cuenta que los bucles `for` requieren también la inicialización de una variable para ser utilizada como índice.

✓ Se deberá determinar si se añade `this` a los nombres de los campos de la clase para distinguir entre variables locales de una función de los campos de una clase.

Paso 3: Estándar para Métodos

Estructura de los Métodos

✓ Se deberá plantear si se establece un tamaño máximo de los métodos, así como posibles separaciones del cuerpo del método en bloques lógicos y comentarios descriptivos de los mismos.

Indentación y llaves

✓ Se deberá establecer una norma que determine el tipo de indentación que se aplicará al código fuente.

✓ Se deberá establecer una norma que determine la forma en que se colocarán las llaves en bucles, condicionales, etc.

Declaración de los Métodos

✓ Se deberá contemplar la forma en que se declararán los métodos en caso de que sus argumentos no quepan en la misma línea.

✓ Se establecerá una norma que indique si se deberá incluir una descripción antes de la declaración de un método a modo de resumen de este. En este punto se podrán establecer normas distintas en función de si el método es público o es privado.

Paso 4: Estándar para el Tratamiento de Excepciones

✓ Se deberán contemplar aspectos como la detección de errores en los argumentos de entrada, o si las excepciones podrán ser mostradas por los propios componentes o únicamente serán mostradas por la aplicación principal.

✓ Se establecerá una norma que indique qué medidas se tomarán para el control de métodos que ejecuten código con altas posibilidades de tener una excepción.

Paso 5: Otros estándares

Separación de Argumentos

✓ Se establecerán normas que determinen la separación entre argumentos y la colocación del paréntesis de la función en relación con el nombre de ésta. Se deberá contemplar la posibilidad de aplicar normas especiales para las sentencias `for`, `while`, `if`, etc.

Paso 6: Estándar para el diseño de la aplicación

Se deberán incluir los patrones de diseño que sean necesarios y relevantes para el desarrollo de la aplicación propuesta, indicando cómo se implementará cada uno de dichos patrones y qué instrucciones se deberán seguir para su utilización. Se recuerda que uno de los patrones que deberán ser aplicados en la implementación de la aplicación será un Modelo-Vista-Controlador.

La normativa de código se proporcionará en un documento en formato PDF denominado *"Normativa de codigo.pdf"*. Este documento se deberá publicar en el directorio raíz del proyecto de software correspondiente al módulo pedido, para cada grupo de prácticas.

También deberá incluirse la normativa de código de Eclipse ChekStyle, que puede exportarse en fichero XML. Dicho fichero se almacenará en el mismo directorio que el documento PDF anterior, y tendrá por nombre "*Normativa de cóodigo.xml".*

Paso 7: Revisar el código fuente y actualizar el repositorio

Una vez definida la normativa de código, los miembros del grupo deberán revisar que este cumple las normas y recomendaciones establecidas, introduciendo los cambios que se consideren necesarios para que el código cumpla la normativa de código previamente establecida.

Tras comprobar que se tiene en local la última versión, los estudiantes deberán aplicar los cambios en el código fuente que sean necesarios para satisfacer la normativa de código previamente definida. Para ello será necesario desproteger antes los ficheros de la solución que se pretenden modificar.

Después de realizar los cambios oportunos, se deben compilar el código fuente en local para comprobar que no se han introducido nuevos defectos en el código. Una vez que la compilación del código no origine ningún error, se pueden proteger los ficheros modificados y desbloquearlos en el repositorio. Para documentar los cambios realizados sobre el código se recomienda añadir un comentario al proteger, el cual debe ser significativo de las modificaciones realizadas.

Sección 5

Normas y entrega del ejercicio guiado

Las sesiones de ejercicios guiados se realizarán en **grupos de 2 personas**. Los estudiantes que no encuentren compañero o que no puedan asistir a las sesiones deberán ponerse en contacto con los profesores de la asignatura para informar de su situación.

La entrega se compondrá de las siguientes partes:

1) Propiedad colectiva de código.

   Para realizar correctamente la entrega, en el repositorio de GitLab asignado a cada grupo deberá figurar la solución *Transport4FutureIoTServices* con el código que implemente correctamente las restricciones de la normativa de código definida por los estudiantes.

2) Normativa de código.

   La normativa de código se proporcionará en un documento en formato PDF denominado "Normativa de codigo.pdf". Este documento se deberá publicar en el directorio raíz del proyecto de software correspondiente al módulo pedido, para cada grupo de prácticas

   Asimismo, esta normativa debe implementarse en Checkstyle para Eclipse (https://checkstyle.org/eclipse-cs/#!/project-setup) dejando en el directorio raíz el fichero XML exportado con dicha normativa, nombrado como "Normativa de código.xml".

   Igualmente, el código fuente que se encuentre en el repositorio GitLab debe cumplir la normativa de código establecida en dichos ficheros.

3) Integración Automatizada.

El código registrado en el GitLab subido por cada grupo debe incluir el proyecto debidamente configurado (incluyendo el fichero pom.xml) que permita la compilación e integración automatizada de los elementos incluidos en el proyecto.

La fecha límite para la entrega de este ejercicio guiado es el **25/02/2020 a las 23:55**.

Se recuerda que los estudiantes que no entreguen a tiempo una solución de este guion de prácticas serán considerados NO APTOS (0 puntos) en dicha entrega, tal y como se indica en las normas de la asignatura disponibles en la presentación de esta, disponibles en Aula Global.

Sugerencias

Cada estudiante debe guardar una copia de la solución entregada hasta la publicación de las calificaciones finales de la asignatura.