Módulo Profesional 05:
Entornos de desarrollo

UF1 Actividad B

CICLO FORMATIVO DE GRADO SUPERIOR EN
DESARROLLO DE APLICACIONES MULTIPLATAFORMA
MODALIDAD ONLINE

ALUMNA:

MARÍA LAURA RONDINA IOBBI

Descripción de la actividad:

En la siguiente actividad práctica se valorarán los conceptos más importantes de la actividad.

Desarrollo de la actividad:

Dar respuesta a los siguientes ejercicios teórico/práctico en el mismo documento para entregar por el campus. Poner nombre al documento y entregar en formato Word.

1. ***Instalar los siguientes dos IDE's(Entorno de Desarrollo integrado) para JAVA***
   - **Instalar eclipse(con la ayuda de la guía de instalación*, youtube e internet)**
   - **Instalar NeatBeans [con la ayuda del libro (información UF1, aparado 2. Instalación y uso de entornos de desarrollo), youtube e internet]**

Dar respuesta a las siguientes preguntas de ambos IDE's:

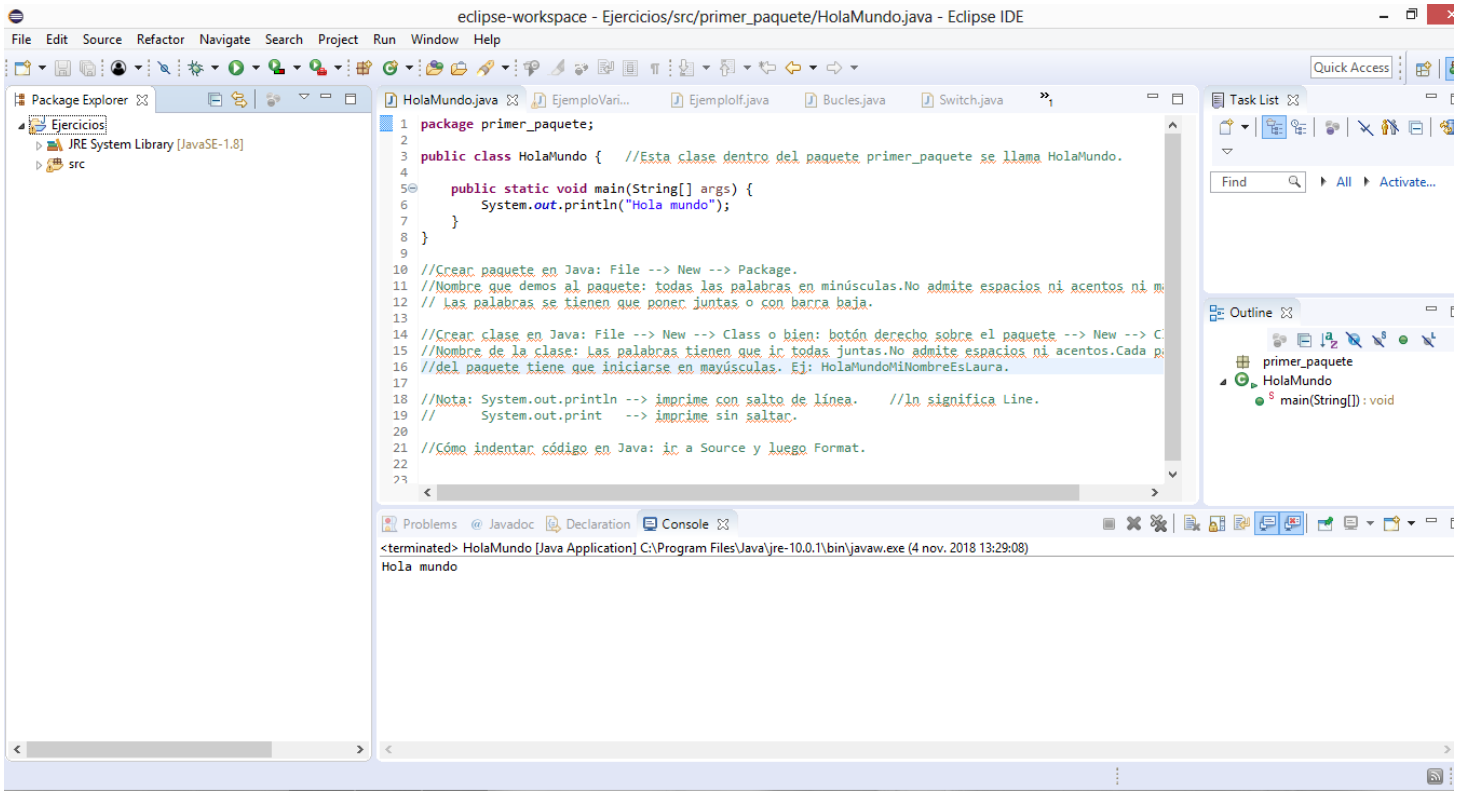
Ejercicio1. Realizar una tabla en la que se comparen estos dos IDE's. Especificar para cada uno de ellos: los lenguajes que soportan, las principales características. [2.0 puntos]

En general, tanto NetBeans como Eclipse son IDE Java de código abierto y multiplataforma. Pero Eclipse es compatible con IBM, mientras que NetBeans es compatible con Oracle. Cada IDE proporciona varias características y herramientas innovadoras para simplificar y acelerar el desarrollo de aplicaciones Java. Pero los desarrolladores deben evaluar las ventajas y desventajas de cada IDE de Java de acuerdo con las necesidades específicas de cada proyecto.

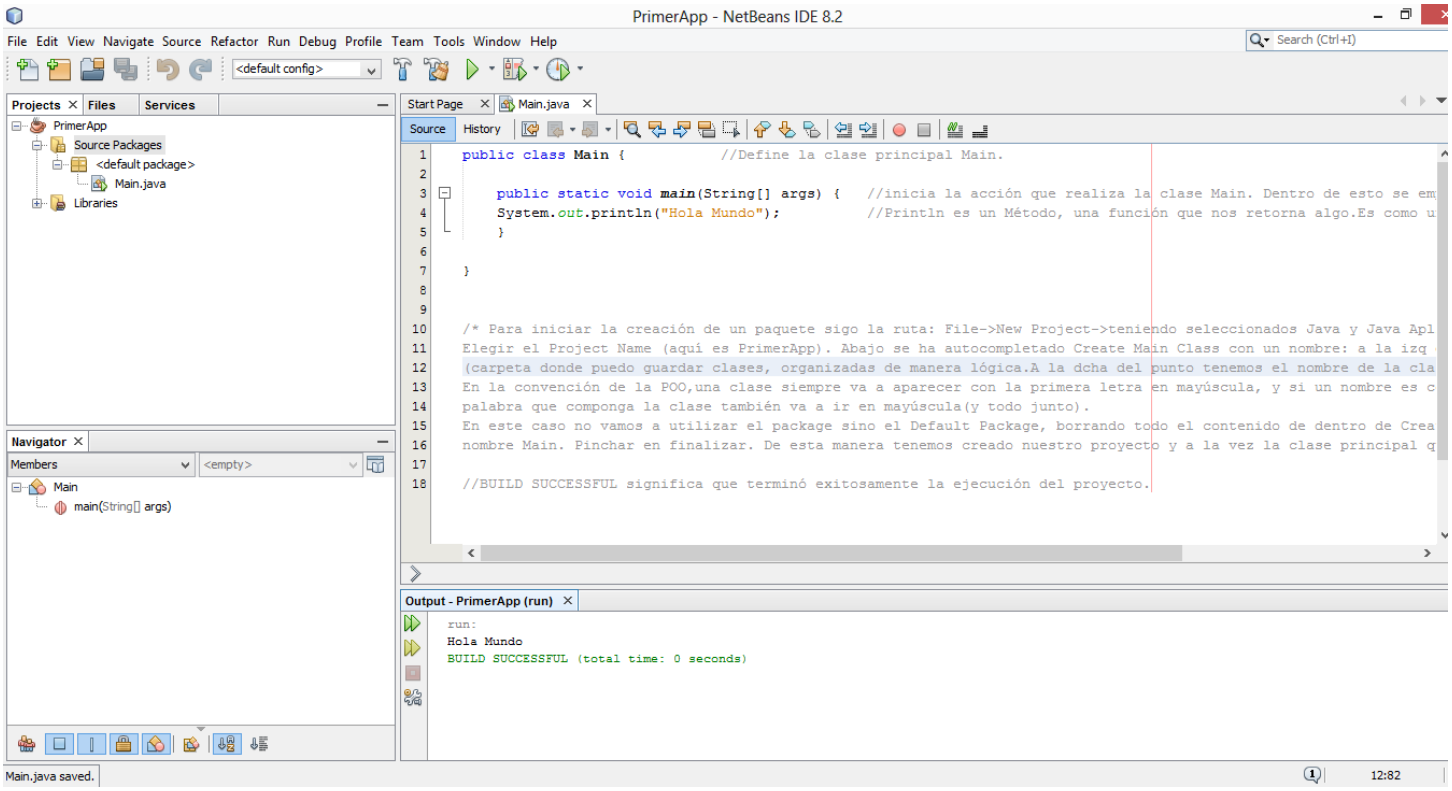
Eclipse	Netbeans
Ambos cuentan con soporte multiplataforma.	
Ambos soportan varios idiomas.	
Ambos fueron desarrollados con la tecnología Java,por ende, necesitan la máquina virtual de Java.	
Ambos son de código abierto	
Ambos permiten integrar otros lenguajes de programación.	
Ambos dan soporte necesario para los frameworks de desarrollo de aplicaciones Java EE.	
Ambos cuentan con una considerable cantidad de comunidades que aportan su conocimiento, basado en sus experiencias de desarrollo (documentación).	
Ambos poseen herramientas de diseño UML.	
Consume MENOS recursos de máquina.	Consume MÁS recursos de máquina.
La curva de aprendizaje es MÁS compleja.	La curva de aprendizaje es MENOS compleja.
Destaca por su vista de perspectivas.	No usa vista de perspectivas.
Tiiene integrado soporte para desarrollo de Android.	Tiene soporte integrado de conexión a base de datos MySql y Sql.
Requiere el plugin WB.	Tiene integrada la GUI.
Tiene un tamaño de 150 MB.	Tiene un tamaño de 79,3 MB.
Tiempo de descarga: Internet de 2MB(23 minutos).	Tiempo de descarga: Internet de 2MB(14 minutos).
Tiempo de ejecución: Pc 1.8 Ghz y 6 Gram (13,25 segundos)	Tiempo de ejecución: Pc 1.8 Ghz y 6 Gram (11,20 segundos)
Plataforma ligera para componentes de software.	Menor consumo de memoria.
Es fácil de usar, ligero y estable.	Es más inestable y lento.
Más rápido, más flexible y más plugins.	Más pesado.
	Permite importar proyectos de Eclipse y otros IDE´s.
Sistema Operativo: Windows/Mac/Linux.	Sistema Operativo: Windows/Mac/Linux/Solaris.
IDE de propósito general.	Variedad de aplicaciones web.
Integra Java con otros lenguajes de programación populares, incluidos Ruby, Perl, PHP, Scala y Groovy	Admite HTML, CSS, JavaScript, PHP y C / C ++ junto con Java.
No tiene muchas funcionalidades listas para usar: se debe instalar y configurar complementos.	Viene con múltiples funcionalidades integradas listas para usar.

Ejercicio 2. Identificar las principales áreas de Trabajo. Adjuntar capturas de pantallas indicándolo. [2.0 puntos]

Pantallazo de **Eclipse** con ejecución del mensaje "Hola Mundo"



Pantallazo de **NetBeans** con ejecución del mensaje "Hola Mundo"

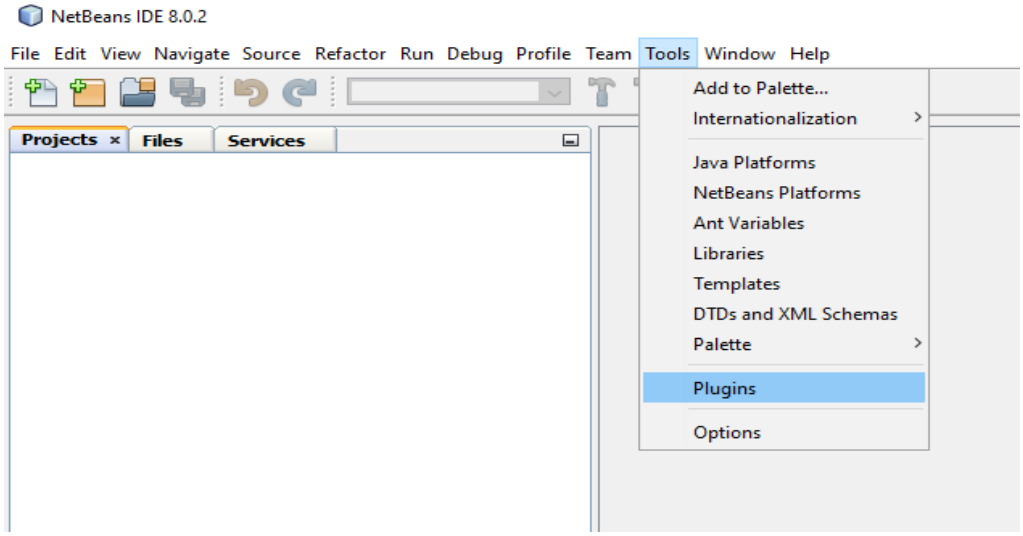


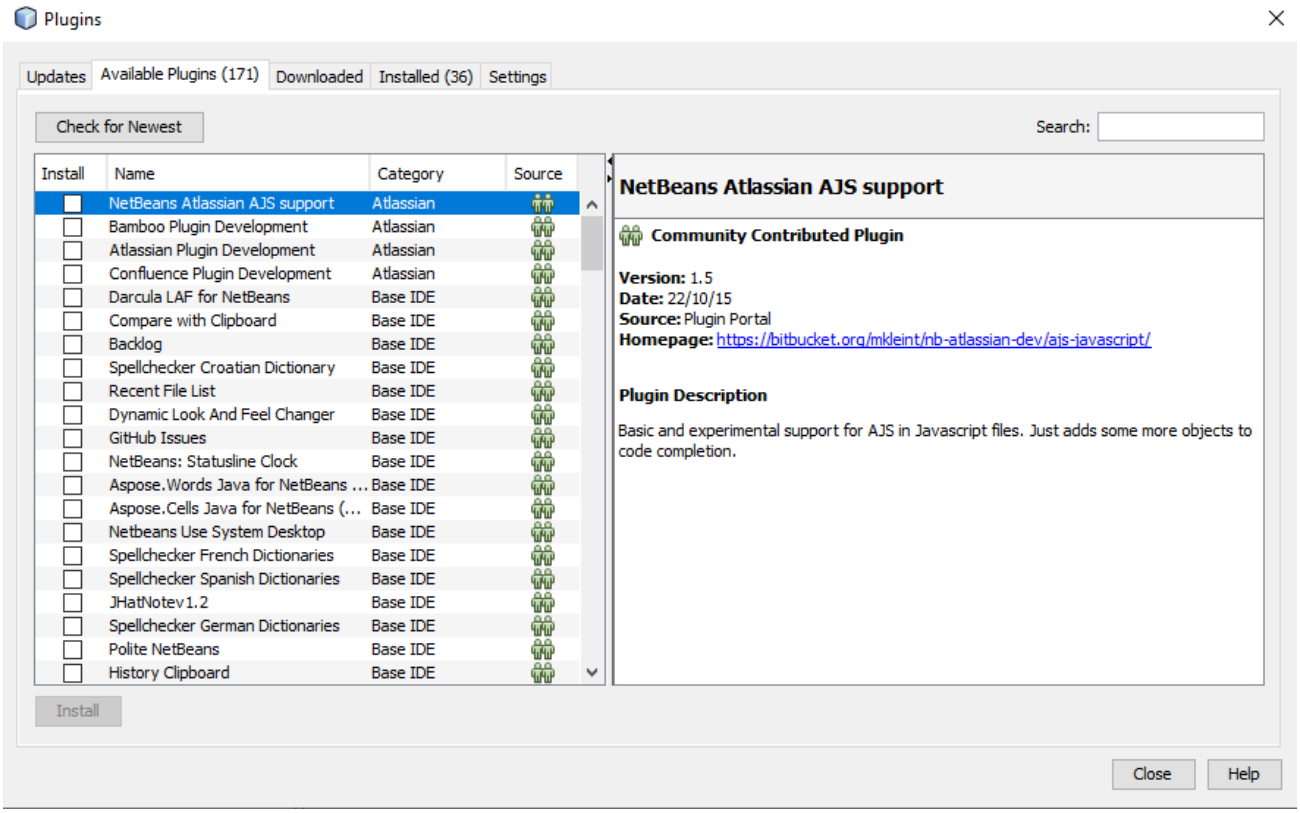
Localización de elementos en las IDEs.

Menú principal, en la parte superior la pantalla, desde donde se realizan las principales acciones del entorno (ficheros, depuración y control del proyecto). Aquí se localizan FILE,EDIT,SOURCE,etc. en Eclipse.. O FILE,EDIT,VIEW..en NetBeans.
Ventana de proyecto. Visión de conjunto del proyecto (ficheros, librerías necesarias para el código entrado, etcétera). Se localiza a la izquierda en Eclipse (Package Explorer), y arriba a la Izquierda en NetBeans (Projects..)
Ventana de navegación. Permite acceder de forma rápida a los elementos de las clases que tengamos seleccionadas. Se localiza a la derecha en Eclipse (Outline) y abajo a la izquierda en NetBeans(Navigator).
Ventana de edición. Editor de texto a través del cual entramos en el código del desarrollo de nuestro proyecto. En ambos casos es la ventana central donde se desarrolla el código para ejecutar la salida del texto "Hola Mundo".
Ventana de salida / Tareas. Muestra los errores de compilación e información seleccionada por el programador mediante palabras clave del código. En la parte inferior de la pantalla: en Eclipse es el campo Console y en NetBeans el campo Output.

Ejercicio3. Identificar cómo se instalan y desinstalan los módulos adicionales (complementos). Indicar procedimiento. Para explicarlo puedes ayudarte de capturas de pantalla. [2.0 puntos]

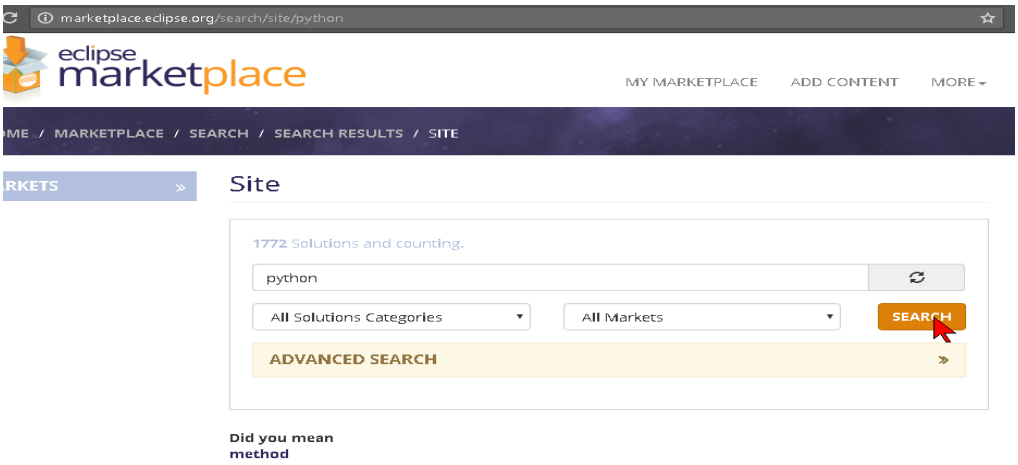
NetBeans es un software libre, por lo que cualquier desarrollador puede crear complementos adicionales o existentes que pueden ser utilizados por otros programadores. Para añadir un complemento, abriremos el menú "Herramientas"(Tools), haremos clic en "Complementos"(plugins) y seleccionaremos la pestaña "Complementos disponibles"(Available Plugins).



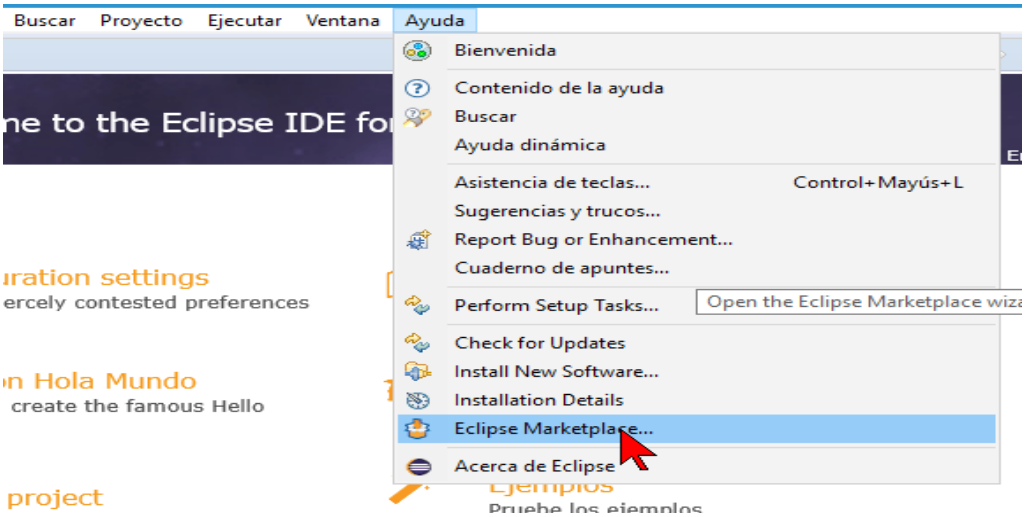


Luego seleccionaremos el complemento que deseemos instalar y haremos clic en “Instalar”. Para desinstalar un complemento la acción es similar pero, en ese caso, cambiaremos la pestaña; es decir, haremos clic en “Complementos Instalados” y desmarcaremos el complemento que deseemos desinstalar. En el botón donde antes podía leerse “Instalar” veremos que indica “Desinstalar”.

Elicpse. Los plugins proporcionan funcionalidades adicionales al núcleo del entorno de desarrollo Eclipse. Al momento de la instalación Eclipse viene con una cantidad de plugins pre-instalados y configurados por defecto. Para añadir uno nuevo hay que recurrir al siguiente enlace **http://marketplace.eclipse.org/** y localizar algún plugin que ofrezca cierta funcionalidad mediante la búsqueda.



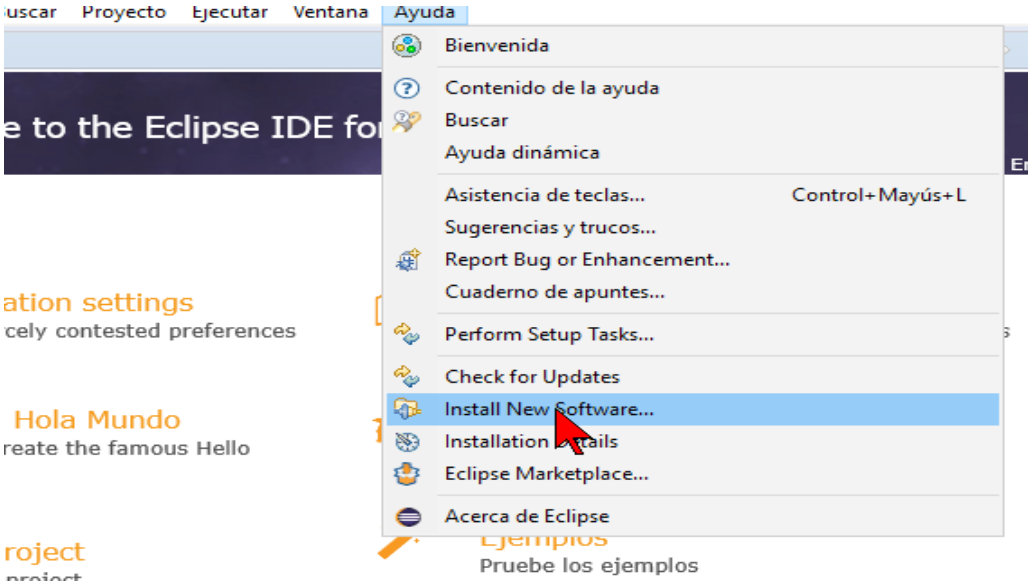
Una vez identificado el plugin, debes dirigirte a Eclipse y en el menú de la parte superior seleccionar la opción Ayuda y luego seleccionar *Eclipse Marketplace…*.



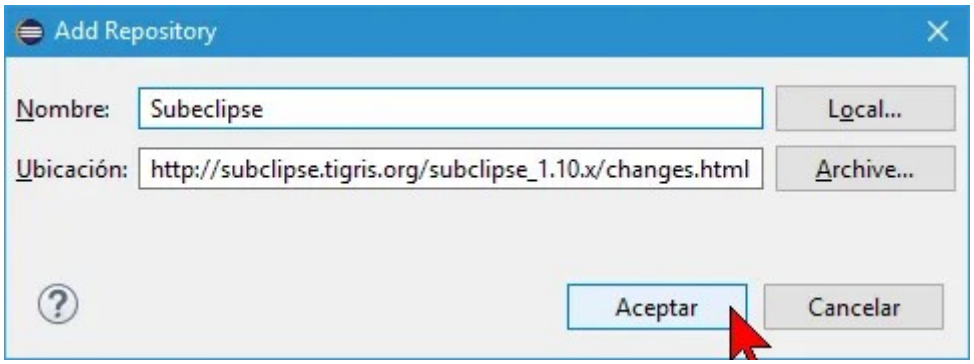
Eso te llevará a la ventana del *Eclipse Marketplace* donde podrás buscar el plugin. Cuando lo halles haz clic en el botón de **Instalar** al lado del mismo. Eso te llevará a otra ventana donde tendrás que dar clic en el botón de **Confirm** para continuar con la instalación. Una vez hecho esto podrás ver el progreso de la instalación en la ventana *Instalando Software*. Espera a que el proceso de instalación termine.

Cómo instalar plug-ins y extensiones en Eclipse directo de una URL:

Algunos plugin no se hallan en el *Marketplace*, pero sí se te provee un URL para acceder el mismo. Para instalar estos plugins debes usar otro método, mediante el uso de la opción de *Ayuda*, seguido de la opción *Install New Software…*.

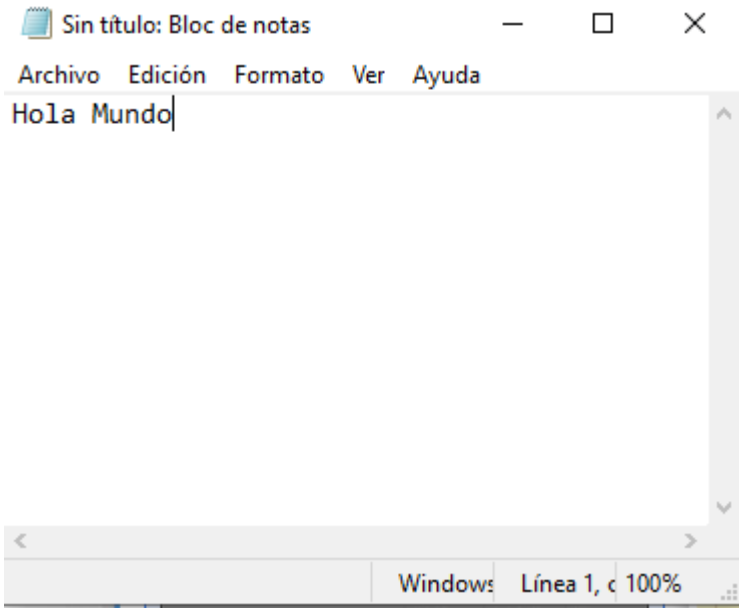


Al hacerlo pasarás a la ventana de instalar. Allí hacer clic en el botón **Añadir**. En este próximo paso es donde tienes que saber la dirección URL desde donde el plugin se puede descargar. Entonces debes dar un nombre al plugin en la caja de texto que dice **Nombre**: y proporcionar la URL en el cuadro de texto que dice **Ubicación**:. Entrado estos datos, haz clic en el botón **Aceptar**.



Una vez hagas esto podrás comenzar la instalación.

Ejercicio 4. En la edición de programas prueba a realizar un ejemplo del programa clásico "hola mundo", desde el editor del entorno de desarrollo y compáralo con respecto a un editor de texto convencional (notepad). Indica las ventajas y desventajas. Para explicarlo puedes ayudarte de capturas de pantalla. [2.0 puntos]



A simplemente vista, tanto el IDE como el editor de texto pueden resultar similares en lo relativo a que en ambos es posible escribir código y ejecutarlo. Sin embargo, la diferencia está en los detalles.

Los **IDEs** no trabajan con archivos y carpetas; en su lugar, emplean el concepto de Proyectos. Un proyecto es una carpeta en el disco duro, pero tiene la diferencia en que el IDE crea archivos adicionales al código para optimizar la experiencia del usuario. En estos archivos puede

tener configuraciones de ejecución, deploy, tipo de proyecto, etc. Todos los archivos que forman parte del proyecto están relacionados entre sí.

Los IDEs concentran todo lo necesario para ejecutar un programa que el usuario escriba en su interfaz, tiene muchas más herramientas que se integran en el mismo programa. Por ejemplo, nos permiten gestionar conexiones a bases de datos, iniciar o detener servidores de aplicaciones, conectarnos a las terminales (locales o remotas), ejecutar nuestro código en modo debug (step by step), ver métricas del performance de la aplicación, etcétera.

También cuentan con resaltado de sintaxis por defecto. Para quienes se inician en el mundo de la programación suelen ser la mejor manera de empezar a escribir y ejecutar código. Ya no sólo por el resaltado de sintaxis, sino porque cuentan también con correctores de sentencias que detectan palabras mal escritas y con depuradores que señalan los errores en el código (que se presentan aunque la sintaxis de la instrucción sea correcta). De hecho la depuración o debuger en tiempo real resulta esencial al tratar con programas largos y complejos. Además, el manejo de la memoria se hace automáticamente utilizando el garbage collector.

Gracias a estas características se puede ver el código en acción mientras se ejecuta, lo que permite tomar medidas mucho más precisas sobre los errores en lugar de estar buscándolos entre un montón de líneas en una pantalla. Con esta ayuda en tiempo real, a medida que vamos escribiendo, los IDEs nos van arrojando las clases o métodos disponibles, qué parámetros reciben y qué responden, o la documentación de la clase o método a medida que vamos escribiendo.

Entre sus desventajas podríamos mencionar el hecho de que requiere un intérprete; que algunas implementaciones y librerías pueden tener código rebuscado; que una mala implementación de un programa en java puede resultar en algo muy lento; que algunas herramientas tienen un coste adicional..

Por su parte, los **editores de texto** son un tipo de software diseñado para manipular archivos de texto. Por lo general se trabaja con los archivos de manera independiente (no están relacionados entre sí). Permiten editar el texto de manera simple y trabajar con texto plano (no tiene color, no señala errores, no tiene autocompletado inteligente, etc). A menudo carece de cualquier tipo de opciones de formato avanzadas y la capacidad de mostrar gráficos. La mayoría de los sistemas operativos modernos incluyen un editor de textos. También hay muchas opciones de terceros disponibles. Existe editores de pago y libres, por lo que hay opciones para elegir, aunque un editor de texto siempre es mucho más económico que un IDE también de pago.

Trabajar con este tipo de editores es más complicado en el sentido de que no tiene todas las herramientas que nos brinda un IDE: los editores de texto son herramientas más minimalistas, mucho más simples y compactas, que proporcionan un entorno de desarrollo simple. El nivel de extensibilidad (Plugin) es muy limitado. Por lo general, tienen una ayuda muy básica y una inteligencia limitada a la hora de predecir el texto que vas a escribir.

A pesar del estrecho enfoque de este tipo de software, existe una serie de ventajas en el uso de un editor de textos:

-Integridad de los datos: Los editores de textos generalmente no añaden formato al escribir de la forma en la que lo hacen los procesadores de palabras. Esto convierte a los editores de textos en la herramienta preferida para trabajar con archivos de texto plano, en los que la integridad de los datos es primordial. Cuando se codifica a mano una página HTML o se edita un código fuente de programación es imperativo que los datos permanezcan en formato de texto plano. Cualquier formato adicional, como el tipo de formato que los procesadores de texto insertan usualmente, puede alterar de forma significativa la integridad del archivo.

-Archivos grandes: Muchos archivos de texto plano, como los de códigos fuente, pueden ser muy grandes. Muchos procesadores de palabras no están optimizados para trabajar con archivos grandes, especialmente al realizar búsquedas. En contraste, los archivos de texto usualmente se destacan al manipular archivos grandes.

-Expresiones regulares: Uno de los beneficios fundamentales de muchos editores de textos es su soporte para la búsqueda de expresiones regulares (regex, o regular expression en inglés). Las regex permiten que un usuario busque patrones en vez de texto específico. Por ejemplo, en vez de buscar una sola dirección de correo electrónico en un documento, las regex permiten que un usuario busque y recupere todas las direcciones que se encuentran en el archivo.

Multi-plataforma: Tanto Mac OS X, Linux/Unix como Windows tienen su propia forma de registrar el final de una línea de texto. Los procesadores de palabras no siempre reconocen las diferentes terminaciones de una línea, lo que ocasiona que el texto se vea perfecto en una plataforma y unido en otra. Los editores de texto tienen un soporte mucho mejor para las diferentes terminaciones de las líneas, y algunos incluso las manipulan de forma transparente.

En conclusión, los editores de texto en un inicio eran solo programas que permitían ver el código de una forma agradable y realizar algunas acciones muy simples, como abrir terminales para ejecutar el programa y refactors muy simples. Sin embargo, los editores de texto modernos, como es el caso de Atom o Sublime texto, han estado evolucionando muy rápido, agregando capacidades que solo los IDE's tenían. Ahora bien, el hecho de que los editores de texto estén evolucionando muy rápido, la realidad es que están muy por detrás de los IDEs, pues un IDE tiene herramientas realmente espectaculares para los programadores, evitando tener que salir del IDE para realizar tareas adicionales, como ejecutar comandos, iniciar servidores, ejecutar consultas de bases de datos, etc, etc.

2. Da respuesta a las siguientes preguntas:

a) Clasifica los siguientes lenguajes de programación, como lenguajes de alto/bajo nivel, interpretados/compilados, declarativos/imperativos, estructurados/orientados a objetos, base de datos.

Python, c, c++, c#, java, php, Mysql, pascal, basic, prolog, Objective-c [0,5 puntos]

Python: Lenguaje de Alto Nivel, Interpretado o de Script (aunque tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi- interpretado), Imperativo, Orientado a Objetos.

c: Lenguaje de Medio Nivel (ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel), Compilado, Imperativo, Estructurado.

c++: Lenguaje de Medio nivel, Compilado, Imperativo, Multiparadigma (un híbrido de programación estructurada y programación orientada a objetos, ya que extiende el lenguaje c con mecanismos que permiten la manipulación de objetos).

c#: Lenguaje de Alto nivel, Compilado, Imperativo, Multiparadigma (Su sintaxis básica deriva de C/C++ , es decir, estructurado y orientado a objetos; además incorpora características de otros lenguajes como Java y Visual Basic)

java: Lenguaje de Alto nivel, Compilado (es compilado a un lenguaje intermedio llamado bytecode, que después es interpretado), Imperativo, Orientado a Objetos.

php: Lenguaje de Alto nivel, Interpretado, Imperativo, Multiparadigma (funcional+orientado a objetos+procedural).

MySQL: Gestor de base de datos. ***SQL:*** Lenguaje de Consulta de Base de Datos. ***PL/SQL:*** Lenguaje de Alto nivel, Imperativo, Compilado, Procedural.

pascal: Lenguaje de Alto nivel, Compilado, Imperativo, Estructurado.

basic: Lenguaje de Alto nivel, Interpretado, Imperativo, Procedural.

Prolog: Lenguaje de Alto nivel , Semi-interpretado (el código fuente se compila a un código de byte el cuál se interpreta en una máquina virtual denominada Warren Abstract Machine-WAM). , Declarativo y Lógico, Procedural.

Objective-c: Lenguaje de Alto nivel, Compilado, Declarativo, Orientado a Objetos.

b) Busca y explica los siguientes conceptos relacionados con el lenguaje java: [0,5 puntos]

JSE: Java SE o Java Standar Edition. Es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java. Constituye la base del lenguaje de programación Java sobre el que se desarrollan Java EE y Java ME. Por ejemplo, Java Enterprise Edition incluye todas las clases en Java SE, más otras específicas.

Java JSE está orientado a desarrollar aplicaciones bajo la arquitectura cliente / servidor, sin que tenga soporte a tecnologías para internet. Con Java SE es posible programar y ejecutar aplicaciones de escritorio y applets. Algunos de los de los principales paquetes Java SE son:

- swing.text.html.parser
- security
- beans
- applet
- awt
- sql
- omg.CORBA
- swing
- rmi

JEE: Java EE o Java Enterprise Edition. Traducido informalmente como **Java Empresarial,** por estar orientado a empresas y a la integración entre sistemas. Es una plataforma de programación parte de la Plataforma Java, para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya

ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. Es considerado informalmente como un estándar debido a que los proveedores deben de cumplir ciertos requisitos de conformidad para declarar que sus productos son *conformes a Java EE*.

Java EE tiene varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc, y define cómo coordinarlos. También configura algunas especificaciones únicas para Java EE para componentes. Su principal ventaja es que permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores.

JME: Java ME o Java Micro Edition. Es una especificación de un subconjunto de la plataforma Java orientada a proveer una colección certificada de APIs de desarrollo de software para dispositivos con recursos restringidos. Está orientado a productos de consumo como PDAs, teléfonos móviles, tablets o electrodomésticos.

Fue una buena opción para crear juegos en teléfonos móviles debido a que se puede emular en un PC durante la fase de desarrollo y luego subirlos fácilmente al teléfono. Al utilizar tecnologías Java, el desarrollo de aplicaciones o videojuegos con estas APIs resulta bastante económico de portar a otros dispositivos. Sin embargo, pocos dispositivos actualmente utilizan la tecnología por la que poco a poco ésta se ha ido al olvido.

JDK: Java Development Kit (**JDK**). Es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red. En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

En los sistemas operativos Microsoft Windows sus variables de entorno son:

- JAVAPATH: es una ruta completa del directorio donde está instalado JDK.
- CLASSPATH: son las bibliotecas o clases de usuario.
- PATH: variable donde se agrega la ubicación de JDK.

Los programas más importantes que se incluyen son:

- **appletviewer.exe**: es un visor de *applets* para generar sus vistas previas, ya que un *applet* carece de método *main* y no se puede ejecutar con el programa java.
- **javac.exe**: es el compilador de Java.
- **java.exe**: es el masterescuela (intérprete) de Java.
- **javadoc.exe**: genera la documentación de las clases Java de un programa.

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

El código binario de Java no es un lenguaje de alto nivel, sino un verdadero código máquina de bajo nivel, viable incluso como lenguaje de entrada para un microprocesador físico. Básicamente se sitúa en un nivel superior al hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un *puente* que entiende tanto el bytecode como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una

máquina virtual Java en concreto, siendo esta la que en última instancia convierte de código bytecode a código nativo del dispositivo final.

La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje, de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas, y así, un programa .class escrito en Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para esos entornos. De ahí el famoso axioma que sigue a Java: "escríbelo una vez, ejecútalo en cualquier parte", o "Write once, run anywhere". La máquina virtual de Java puede estar implementada en software, hardware, una herramienta de desarrollo o un navegador web, y lee y ejecuta código precompilado bytecode que es independiente de la plataforma multiplataforma.

Applets: Un *applet* es un componente de una *aplicación* que se ejecuta en el contexto de otro programa, por ejemplo, en un navegador web. Cuando un navegador carga una página web que contiene un *applet*, este se descarga en el navegador web y comienza a ejecutarse, ofreciendo información gráfica. El navegador que carga y ejecuta el *applet* se conoce como "contenedor". A diferencia de un *programa*, un *applet* no puede ejecutarse de manera independiente.

Un applet se puede cargar y ejecutar desde cualquier explorador que soporte JAVA (Internet Explorer, Netscape, Mozilla Firefox...). Ejemplos comunes de *applets* son las Java applets, las animaciones Flash, el Windows Media Player utilizado para desplegar archivos de video incrustados en los navegadores como el Internet Explorer.

Un **applet Java** es un código JAVA que carece de un método main, por eso se utiliza principalmente para el trabajo de páginas web, ya que es un pequeño programa que es utilizado en una página HTML y representado por una pequeña pantalla gráfica dentro de ésta. Los applets de Java pueden ejecutarse en un navegador web utilizando la Java Virtual Machine (JVM), o en el AppletViewer de Sun. Son multiplataforma (funcionan en Linux, Windows, OS X, y en cualquier sistema operativo para el cual exista una Java Virtual Machine) y son compatibles con la mayoría de los navegadores web.

API: A*pplication Programming Interface o* interfaz de programación de aplicaciones. Es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del *software*. Uno de sus principales propósitos consiste en proporcionar un conjunto de funciones de uso general, de forma que los programadores se benefician de su funcionalidad, evitándose el trabajo de programar todo desde el principio.

La **API Java** es la interfaz de programación de aplicaciones provista por los creadores del lenguaje de programación Java, que da a los programadores los medios para desarrollar aplicaciones Java. Como el lenguaje Java es un lenguaje orientado a objetos, la API de Java provee de un conjunto de clases utilitarias para efectuar toda clase de tareas necesarias dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.

Javac: Pronunciado "java-see", es el compilador principal de Java incluido en el Kit de desarrollo de Java (JDK) de Oracle Corporation. Sintaxis:

javac [opciones] archivo.java
javac_g [opciones] archivo.java

javac compila los archivos con código fuente Java (*.java*) en archivos de código de bytes Java (*.class*). Los archivos de bytecodes son independientes de la plataforma; esto significa que puede compilar su código en un tipo de hardware y sistema operativo, y luego ejecutar el código en cualquier otra plataforma que admita Java.

El compilador de Java y el resto de la cadena de herramientas estándar de Java coloca las siguientes restricciones en el código:

- El código fuente se guarda en archivos con el sufijo ".java"
- Los códigos de bytes se guardan en archivos con el sufijo ".class"
- Para los archivos de código fuente y bytecode en el sistema de archivos, las rutas de los archivos deben reflejar el nombre del paquete y la clase.

Por defecto javac genera los archivos *.class* en el mismo directorio del archivo fuente *.java*, excepto en el caso de utilizar la opción –**d**, que permite especificar otro directorio. Cuando un archivo fuente Java hace referencia a una clase que no esta definida en alguno de los archivos fuente de la línea de comandos, entonces **javac** hace uso de la variable de entorno **CLASSPATH**, que por defecto, apunta a las clases del sistema.

Nota: javac compilador no debe confundirse con el compilador Just in Time (JIT) que compila los códigos de bytes en código nativo.

Javadoc: Es una utilidad de Oracle para la generación de documentación de APIs para el programador en formato HTML a partir de código fuente Java. Básicamente, es un programa, que toma lo comentarios que se colocan en el código con marcas especiales y construye un archivo HTML con clases, métodos y la documentación que corresponde. Es el estándar de la industria para documentar clases de Java. La mayoría de los IDEs los generan automáticamente.

Se intenta evitar que la documentación que se genera mediante un editor de texto se quede rápidamente obsoleta cuando el programa continúa su desarrollo y no se tiene la disciplina/tiempo para mantener la documentación al día. Para ello, se pide a los programadores de Java que escriban la documentación básica (clases, métodos, etc.) en el propio código fuente (en comentarios en el propio código), con la esperanza de que esos comentarios sí se mantengan actualizados cuando se cambia el código. La herramienta Javadoc extrae dichos comentarios y genera con ellos un juego de documentación en formato html. Lo interesante de la herramienta es que la transformación fuente->html la realiza utilizando un mecanismo extensible llamado doclet, que le permite analizar la estructura de una aplicación Java. Así es como el doclet javadoc “JDiff” puede generar informes de lo que ha cambiado entre dos versiones de una API.

Para generar APIs con Javadoc han de usarse etiquetas (tags) de HTML o ciertas palabras reservadas precedidas por el carácter "@". Estas etiquetas se escriben al principio de cada clase, miembro o método, dependiendo de qué objeto se desee describir, mediante un comentario iniciado con "/**" y acabado con "*/".

3. En el desarrollo de proyectos (desarrollo de aplicaciones) se emplean metodologías para gestionar todo el ciclo de vida del proyecto. Estás metodologías están divididas en dos ramas diferentes que son:

a. Metodología tradicional como es la de cascada (waterfall) [0,5 puntos]

En Ingeniería de *software* el **desarrollo en cascada**, también llamado **secuencial** o **ciclo de vida de un programa** (denominado así por la posición de las fases de su desarrollo, que parecen caer en cascada "*por gravedad*" hacia las siguientes fases), es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de *software*, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. Al final de cada etapa, el modelo está diseñado para llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo para avanzar a la siguiente fase. Este modelo fue el primero en originarse y es la base de todos los demás modelos de ciclo de vida.

Un ejemplo de una metodología de desarrollo en cascada es:

1. Análisis de requisitos.
2. Diseño del sistema.
3. Diseño del programa.
4. Codificación.
5. Pruebas.
6. Implementación o verificación del programa.
7. Mantenimiento.

De esta forma, cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo. La palabra *cascada* sugiere el esfuerzo necesario para introducir un cambio en las fases más avanzadas de un proyecto.

Si bien ha sido ampliamente criticado desde el ámbito académico y la industria, sigue siendo el paradigma más seguido al día de hoy.

Ventajas:

- Debido a que el método de cascada requiere un amplio esfuerzo de preparación previa, permite comenzar con el software con bastante rapidez.
- Estimar calendarios y presupuestos con mayor precisión.
- Lograr un nivel de satisfacción del cliente más elevado que otros enfoques, ya desde el principio.
- Realiza un buen funcionamiento en equipos débiles y productos maduros, por lo que se requiere de menos capital y herramientas para hacerlo funcionar de manera óptima.
- Es un modelo fácil de implementar y entender.
- Está orientado a documentos.
- Es un modelo conocido y utilizado con frecuencia.
- Promueve una metodología de trabajo efectiva: Definir antes que diseñar, diseñar antes que codificar.

Desventajas:

- En la vida real, un proyecto rara vez sigue una secuencia lineal, esto crea una mala implementación del modelo, lo cual hace que lo lleve al fracaso.

- El proceso de creación del *software* tarda mucho tiempo ya que debe pasar por el proceso de prueba y hasta que el *software* no esté completo no se opera. Esto es la base para que funcione bien.
- Cualquier error de diseño detectado en la etapa de prueba conduce necesariamente al rediseño y nueva programación del código afectado, aumentando los costos del desarrollo.
- Una etapa determinada del proyecto no se puede llevar a cabo a menos de que se haya culminado la etapa anterior.

b. O Metodologías ágiles como es SCRUM** [0,5 puntos]

Scrum es un marco de trabajo para desarrollo ágil de software, que solapa las diferentes fases de desarrollo y otorga prioridad a lo que tiene más valor para el cliente, fijando tiempos máximos para lograr objetivos. Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo y obtener el mejor resultado posible de proyectos. Permite la creación de equipos auto organizados impulsando la co-localización de todos los miembros del equipo, y la comunicación verbal entre todos los miembros y disciplinas involucrados en el proyecto. Los roles principales en Scrum son el *Scrum Master*, que procura facilitar la aplicación de Scrum y gestionar cambios, el *Product Owner*, que representa a los *stakeholders* (interesados externos o internos), y el *Team* (equipo) que ejecuta el desarrollo y demás elementos relacionados con él.

Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas **amarillas "post-it" y pizarras hasta paquetes de software.** Así, si se utiliza una pizarra con notas autoadhesivas cualquier miembro del equipo podrá ver tres columnas: trabajo pendiente ("backlog"), tareas en proceso ("in progress") y hecho ("done"). De un solo vistazo, una persona puede ver en qué están trabajando los demás en un momento determinado.

La implementación de la metodología Scrum puede completarse en 7 pasos:

1. Identificar al propietario de un producto que pueda asumir el rol de creación y facilitar la creación de la acumulación de productos.

2. Crear el Backlog, una lista de tareas y requisitos que el producto final necesita. Se considera prioritario.

3. Calcular el tiempo aproximado para la creación del backlog del producto.

4. Planificar los Sprint cuidadosamente (son un marco de tiempo predeterminado dentro del cual el equipo completa los conjuntos de tareas del *Backlog). I*dealmente deberían ser de 30 días.

5. Decidir el presupuesto del Sprint, para tener una idea aproximada de la cantidad de horas disponibles que el equipo tiene para trabajar.

Paso 6. Habilitar un espacio de colaboración para el equipo de Scrum.

Paso 7. Preparar un gráfico diario de *burndown* que permita rastrear el progreso diario. Se trata de una representación gráfica del trabajo por hacer en un proyecto en el tiempo.

Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales. También se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

Beneficios de Scrum:

1. Flexibilidad a cambios. Gran capacidad de reacción ante los cambiantes requerimientos generados por las necesidades del cliente o la evolución del mercado.
2. Reducción del Time to Market. El cliente puede empezar a utilizar las características más importantes del proyecto antes de que esté completamente terminado.
3. Mayor calidad del software. El trabajo metódico y la necesidad de obtener una versión de trabajo funcional después de cada iteración, ayuda a la obtención de un software de alta calidad.
4. Mayor productividad. Se logra, entre otras razones, debido a la eliminación de la burocracia y la motivación de un equipo autónomo.
5. Maximiza el retorno de la inversión (ROI). Creación de software solamente con las prestaciones que contribuyen a un mayor valor de negocio gracias a la priorización por retorno de inversión.
6. Predicciones de tiempos. Se conoce la velocidad media del equipo por sprint, con lo que es posible estimar de manera fácil cuando se podrá hacer uso de una determinada funcionalidad que todavía está en el Backlog.
7. Reducción de riesgos.El hecho de desarrollar, en primer lugar, las funcionalidades de mayor valor y de saber la velocidad a la que el equipo avanza en el proyecto, permite despejar riesgos efectivamente de manera anticipada.

Desventajas:

1. Dificultad de aplicación en grandes proyectos.
2. Plantea un problema si el desarrollo está restringido por una fecha de entrega y un precio de entrega cerrados por contrato.
3.Supone que el equipo está muy formado y motivado.
4. Funciona bien sólo en equipos pequeños.
5. Necesita sólo los miembros del equipo con experiencia. Si el equipo está formado por personas que son novatos, el proyecto no se puede completar en el tiempo.
6. Si alguno de los miembros del equipo dejan el desarrollo del proyecto, puede tener un enorme efecto inverso en el proyecto.
7. Es necesario que el equipo de trabajo sea auto organizado.
8. Depende en gran medida de la interacción del cliente, por lo que si el cliente no está claro, el equipo se puede conducir en la dirección equivocada.

Ejercicio 4. Indicar las principales diferencias entre estas dos metodologías de gestión de proyectos.

Las metodologías ágiles son un conjunto de mejores prácticas que se basan principalmente en su manifiesto ágil, es decir, una serie de cuatro principios: los individuos y su interacción, el software utilizado, la colaboración con el cliente y la respuesta al cambio. Las metodologías tradicionales son aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos modelado. Una vez planificado, controlado y detallado, comienza el ciclo de desarrollo del producto. La principal diferencia entre ambas es que Scrum es una metodología ágil y Waterfall es tradicional.

Scrum adopta una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto. Basa la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados. Solapa las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada.

A continuación, se explicará las principales diferencias entre ambas metodologías.

LIDERAZGO

En la metodología Scrum, encontramos el Scrum Master que es una persona encargada de hacer que toda la metodología de Scrum funcione, de eliminar todas las barreras que puedan presentarse y de proyectar las iteraciones, entregables e informes de cada Sprint. Mientras que en Waterfall, encontramos al Project Manager, que es un gerente que se encarga de dirigir el proyecto y cuya función principal es la de lograr que se realice el objetivo. Es una persona organizada, con conocimientos financieros, habilidades de negociación, planeación y diagnósticos.

La principal diferencia es que mientras el Project Manager depende de él para designar responsabilidades a su equipo, el Scrum Master trabaja en conjunto con su equipo para desarrollar esas responsabilidades, es decir, en Scrum, el equipo tiene más libertad para trabajar mientras que en Waterfall todo depende el Project Manager.

EQUIPOS

Mientras el equipo Waterfall sigue las indicaciones del Project Manager, el equipo Scrum es auto-organizable. Respecto a los tiempos, Waterfall basa sus acciones en cuanto los tiempos indicados por el Project Manager, mientras que Scrum se basa en las prioridades y resultados de valor para ser entregados al cliente. Respecto a las modificaciones, el equipo Waterfall sigue un plan en cuanto al desarrollo del proyecto y Scrum constantemente puede modificar sus actividades.

ENTREGAS

En la metodología Waterfall las entregas se presentan al final del proyecto, siendo un desarrollo aproximadamente realista ya que no han sido aprobados por el cliente. En Scrum, las entregas se realizan cada 2 o 4 semanas, dependiendo de los tiempos de iteración marcados por el equipo, para comprobar los resultados obtenidos en la iteración.

En conclusión, ambas metodologías son aptas para el desarrollo de un proyecto, pero será mejor elegir una u otra metodología dependiendo de la clase de proyecto a realizar. Decidir la metodología más adecuada para cada proyecto es una decisión complicada. Para aplicar la metodología Waterfall es conveniente que los requisitos y características del modelo estén bien definidos, la definición del producto sea estable, la tecnología se entienda bien, que no haya requisitos ambiguos, que el proyecto sea corto y tener un Project Manager bien definido y con conocimientos amplios. Mientras que para aplicar la metodología Scrum se tiene que adoptar una estrategia de desarrollo incremental en lugar de la planificación y ejecución completa del producto, y basar la calidad del resultado más en el conocimiento tácito de las personas y en equipos auto-organizados.

	SCRUM	WATERFALL
ENFÁSIS	Persona	Proceso
DOCUMENTACIÓN	Mínima	Completa
PLANIFICACIÓN	Baja	Alta
ORGANIZACIÓN	Auto-organizado	Gestionado
GESTIÓN	Descentralizado	Centralizado
CAMBIOS	Constante	Mínimo
LIDERAZGO	Equipo	Project Manager
PARTICIPACIÓN DEL CLIENTE	Alta	Baja