C.F.G.S. DESARROLLO DE APLICACIONES WEB

Módulo Profesional
ENTORNOS DE DESARROLLO

UD 3. HERRAMIENTAS DE DESARROLLO.

Curso : 2018/19
Profesor : Jorge Martín Cabello

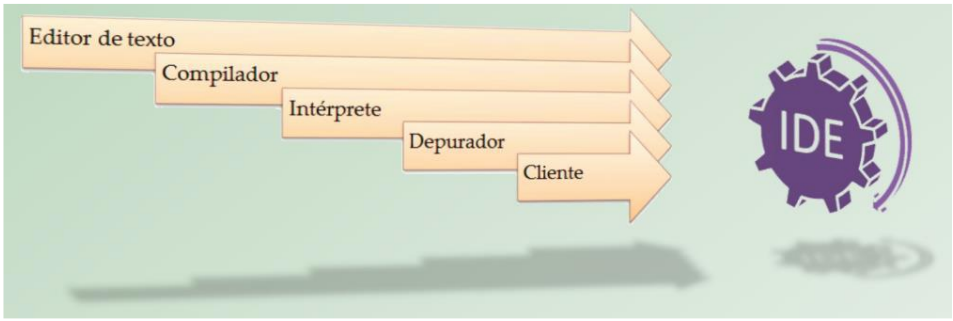
Contenido

1. DEFINICIÓN Y CARACTERÍSTICAS BÁSICAS DE UN IDE

Un IDE es un programa informático que tiene el objetivo de asistir al programador en la tarea de diseñar y codificar un software mediante la inclusión de múltiples herramientas destinadas para dicha tarea.

Por lo general los entornos de desarrollo actuales ofrecen:

- **Editor de texto**: facilita la escritura organizando las instrucciones atendiendo a un formato, colores, permite la escritura automática, autocompletar, además de las clásicas funciones de buscar, reemplazar cortar, pegar, etc.
- **Compilador/Interprete**: normalemente los IDEs incluyen herramientas de compilación y ejecución propias, sin tener necesidad de un jdk.
- **Depurador**: Herramienta que permite la ejecución de una aplicacion instrucción a instrucción, y nos ayuda a examinar las distintas situaciones y cambios que se produzcan en las variables del programa.
- **Cliente:** Módulo de ejecución del programa como usuario.



También incorporan actualmente:

- **Asistente para GUI**: (GUI - Interfaz gráfica de usuario). Normalmente es una paleta de componentes que me permite arrastrar y soltar (drag and drop) componentes gráficos a un panel que corresponde a la ventana que vamos a crear.
- **Control de versiones**: permite almacenar cambios en nuestro código y compartirlo con otros desarrolladores.
- **Exportar programas**: permitirá exportar nuestra aplicación completa en algún formato de fichero.

Un IDE aporta una serie de **herramientas adicionales** para cumplir con mayor eficacia el objetivo de facilitar el trabajo a los desarrolladores.
La **configuración del IDE** permite entre otras cosas añadir y modificar las barras de herramientas, pudiendo crear comandos personalizados y atajos de teclado para cada una de ellas.
Estableciendo el posicionamiento de las ventanas y barras conjuntamente con los atajos de teclado podremos **mejorar sumamente nuestro rendimiento** y aprovechar con mayor comodidad todas las funciones del IDE.
La mayoría de los IDE´s también permiten **configurar interfaces diferentes** dependiendo de la operación que se esté realizando, teniendo una configuración para la etapa de desarrollo y otra diferente para la etapa de depuración.

2. EXTENSIONES Y HERRAMIENTAS

Es posible programar sin un IDE utilizando dos herramientas:
EDITOR DE TEXTOS

COMPILADOR
https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html

El problema es que se pierden muchas facilidades, comodidades y otras herramientas como las siguientes.

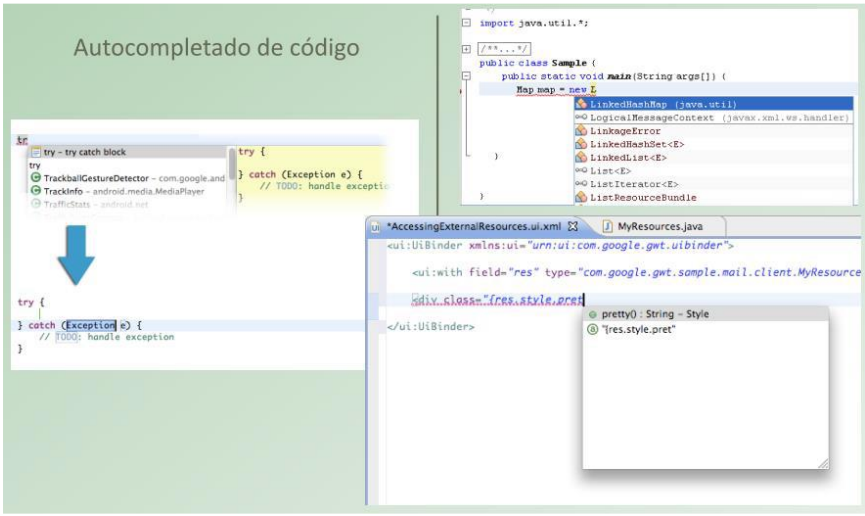
Coloreado de sintaxis/No coloreado:
Es importante destacar las palabras “reservadas” a la hora de programar en el lenguaje especificado

```
public class Prueba{
    public static void main(String [] args){

        int a = 3;
        int b = 5;

        int c = a + b;

        System.out.println("a + b = " + c);
    }
}
```

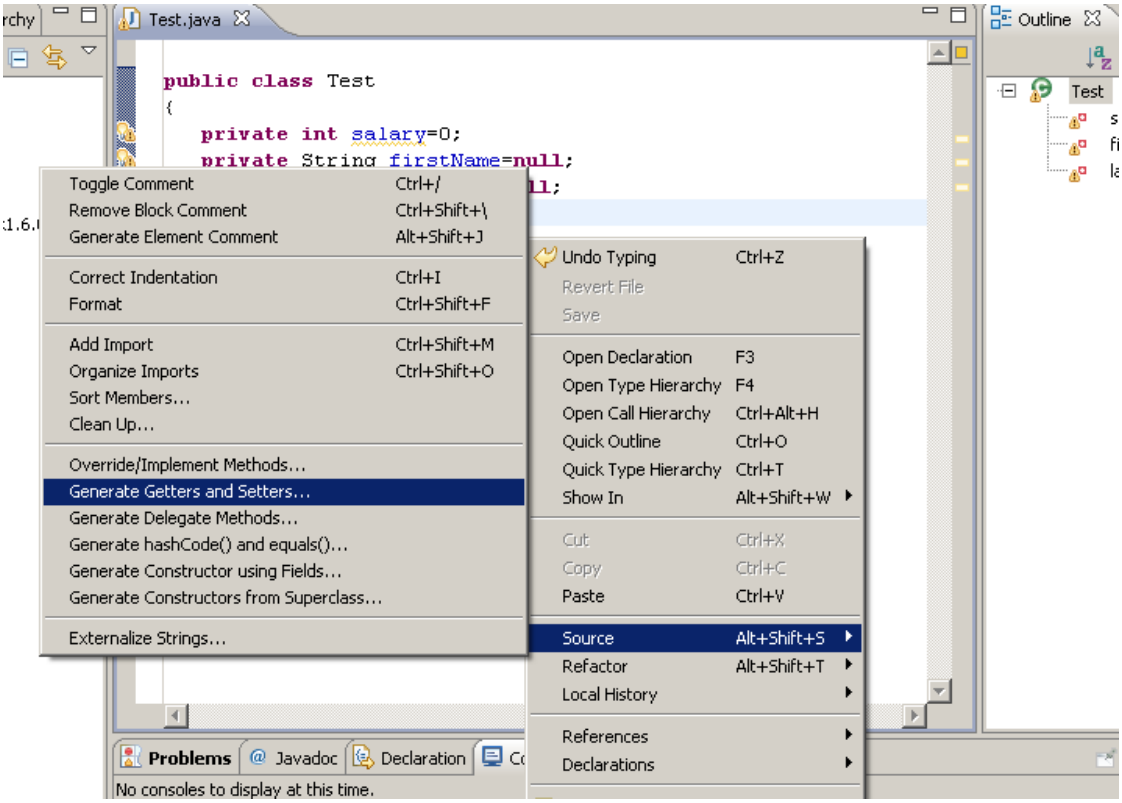
```
public class Prueba{
    public static void main(String [] args){

        int a = 3;
        int b = 5;

        int c = a + b;

        System.out.println("a + b = " + c);
    }
}
```

Es importante destacar las palabras “reservadas” a la hora de programar en el lenguaje especificado

Ayudas en la programación:



Creación automática de estructuras:



3. CRITERIOS DE ELECCIÓN DE UN IDE.

SISTEMA OPERATIVO:
Importante: en qué sistema operativo vamos a trabajar y, más importante aún, para qué sistema operativo vamos a desarrollar nuestro software.

LENGUAJE DE PROGRAMACIÓN:
Un IDE puede soportar uno o varios lenguajes de programación, por lo que saber en qué lenguaje de programación vamos a codificar nuestro software y qué lenguajes nos ofrecen los distintos IDE es una información valiosa que hay que tener en cuenta.

FRAMEWORK:
Los frameworks son un conjunto de utilidades para desarrollar aplicaciones, normalmente un conjunto de clases. Un IDE ofrece facilidades para trabajar con los frameworks, por lo que éstos también tienen relación con los sistemas operativos. Por ejemplo, si quisiéramos desarrollar en Visual Basic bajo un sistema operativo Linux no sería Visual Studio nuestra elección del IDE, sino que tendríamos que utilizar Gambas.

HERRAMIENTAS:
Las diferentes herramientas de las que disponen los IDE son el último criterio de selección. Seguramente nos encontremos con varios IDE que cumplen los requisitos de lenguaje y sistema operativo, pero no todos tienen las mismas funciones, por lo que saber cuáles son esas herramientas es un dato sumamente importante en nuestra decisión.

DISPONIBILIDAD:
Una vez comprobados todos los criterios de selección mencionados tendríamos que comprobar si el IDE que cumple los requisitos está a nuestro alcance, ya sea por una cuestión de presupuesto o localización.

COMPARATIVA SEGÚN LAS CARACTERISTICAS:

IDE	SISTEMA OP.	FRAMEWORK	LENGUAJES
Eclipse	Multiplataforma	GEF (Interfaces), EMF (Modelado), ...	JAVA, ANSI C, C++, JSP, Sh, Perl, Php, Sed
VisualStudio	Windows	.NET	C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP
NetBeans	Multiplataforma	Struts, ICEFaces...	JAVA, JSP, Haskel, CPP,Yacc, Sh, Lex, Perl, Objc, Ansic
BlueJ	Multiplataforma	JUnit, Robot world...	JAVA
IntelliJ IDEA	Multiplataforma	Struts, JSF, Hibernate...	Java, Groovy,Haxe, Perl, Scala, XML/XSL, Kotlin, ActionScript/MXML, HTML/XHTML/CSS, JavaScript, Lua, PHP, Python, Ruby/JRuby, SQL...
Android Studio	Multiplataforma	Lintelligence, JUnit...	JAVA, C
XCode	Mac OS	Cocoa, UIKit...	C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez), y Swift.
Gambas	GNU/Linux	Similar a .NET	Visual Basic, JAVA

4. ECLIPSE.


Es un entorno de desarrollo integrado de código abierto desarrollado en el lenguaje Java. Ofrece soporte para múltiples lenguajes.

4.1. Workspace / Proyecto

- Un **workspace** es un tipo de directorio que usa Eclise para almacenar proyectos. *Un workspace no es un proyecto*. Podemos organizar los distintos proyectos en diferentes workspaces. Para cambiar de workspace ire a la pestaña **File → Switch workspace**. Por defecto, Eclipse abre el último workspace utilizado.

Es importante entender que todos los proyectos que tengo en un workspace no tienen porqué aparecer en el *Explorador de Paquetes* de Eclipse. Pueden estar en el workspace y no haberlos abierto o importado en Eclipse.

- Un **proyecto** es un directorio que contiene subdirectorios para organizar todos los elementos de un programa. Principalmente tenemos la carpeta **src** donde guardamos los ficheros fuente .java, la carpeta **bin** donde se guardan los ficheros .class generados de la compilación de los fuentes. Para abrir un proyecto puedo hacerlo desde la pestaña **File → Open Projects…** o tambien desde **File → import → General → Existing Projects into workspace**.

Es importante entender que para abrir un proyecto no es necesario tenerlo en ningún workspace. Puede estar en cualquier lugar de nuestro sistema, y ahí seguirá. Sin embargo es útil almacenarlo en algún workspace.

4.2. Plantillas (Templates)

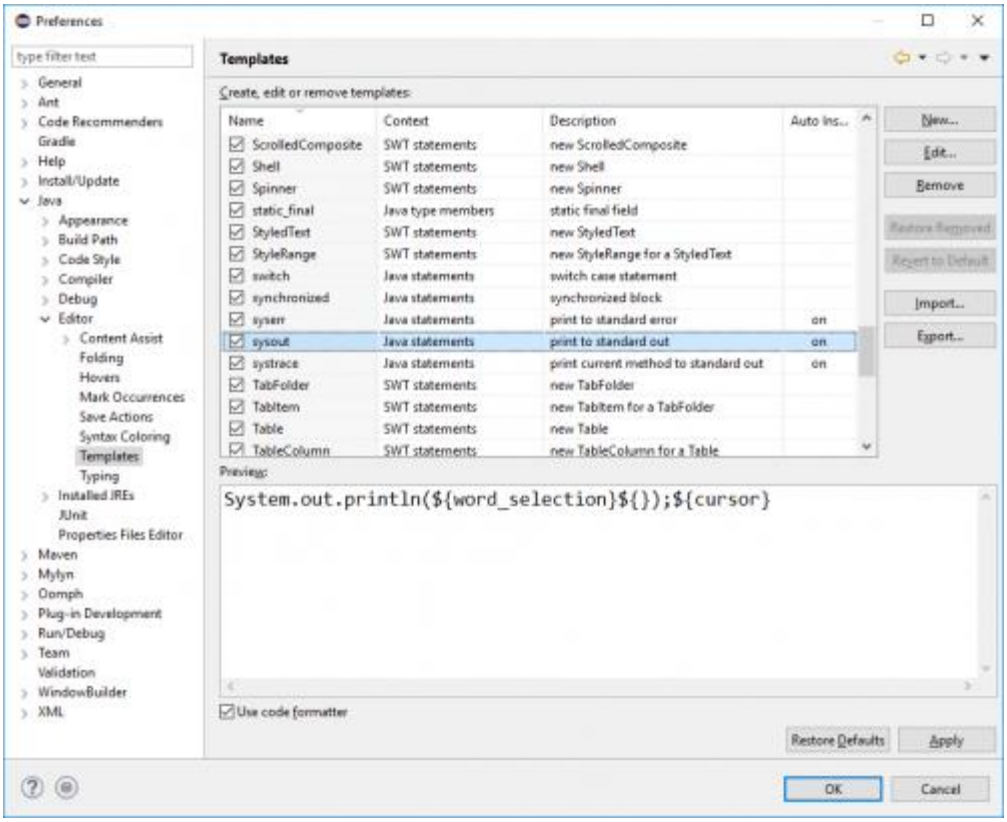
Las plantillas son un tipo de herramientas mediante las cuales se puede autogenerar (autocompletar) código a partir de una palabra. Se despliegan en un menu al pulsar la combinación de teclas *ctrl + barra espaciadora*.

```
syso -> System.out.println()
```

Para ver las plantillas que tenemos predefinidas para un lenguaje iremos a Window → Preferences.

Ahí accederemos al lenguaje del cual queramos mostrar las plantillas o gestionarlas.

Para Java: Java → Editor → Templates



Desde la opción ***New*** podremos crear nuestra propia plantilla. Esta se compone de:

- Nombre: nombre que aparecerá en mi editor cuando la invoque
- Descripción: Para qué se usa
- Contexto: Podemos indicar el lenguaje para el que se usa
- Patrón: código de la plantilla. Puedo además usar variables de Eclipse

Podemos consultar algunos aspectos de la creación de plantillas desde la documentacion de eclipse, en concreto sobre las variables que podemos usar al crear plantillas.

Ejemplo de plantilla para Scanner:

4.3. Depurador

Para establecer un punto de ruptura en mi código (breakpoint) tan solo debo hacer doble clic en mi editor de código de Eclipse, en la pequeña columna vertical que tenemos a la izquierda del número de linea. Cuando creamos un breakpoint queda marcado un pequeño punto azul. Se eliminan de la misma

forma. Para acceder al modo depuración: Menu *Run* → *Debug*, o mediante la tecla *F11*. Mi programa se ejecutará hasta que encuentre un *breakpoint*, momento en el que parará.

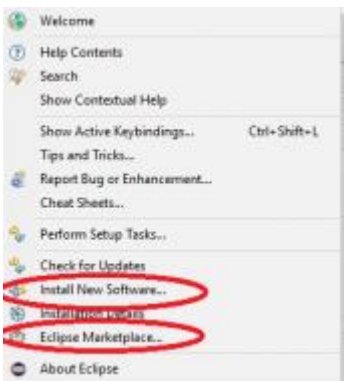
- El primer botón (F8), botón "play", hace que continúe la ejecución de nuestro programa hasta el siguiente Breakpoint.
- El siguiente botón, parecido al botón de pausa, suspende la ejecución del programa que está corriendo. Es una forma de parar "al vuelo" el programa, esté donde esté en ese momento, haya o no breakpoint.
- El cuadrado rojo es el botón de stop, aborta la ejecución de nuestro programa.

Botones de avance paso a paso:

- El siguiente botón (F5), una flecha amarilla que se "mete" entre dos puntos, hace que el debugger vaya a la siguiente instrucción y si encuentra un método, se meterá dentro del método y se parará en la primera línea del mismo, de forma que podemos depurar dentro del método.
- El siguiente botón (F6), una flecha amarilla que salta sobre un punto negro, avanza un paso la ejecución del programa, y si se encuentra un método no entra dentro de él, sino que salta a la siguiente instrucción.
- El último botón (F7) es una flecha saliendo de entre dos puntos negros, cuando nos encontramos dentro de un método, salta hasta la instrucción en la que ha sido llamado el método.

Además podemos ver el valor que devuelve una instrucción, seleccionado la expresión (método) y ejecutando el atajo del teclado **Ctrl + Shift + i**. Por ejemplo, si no se qué puede devolver la instrucción *cadena.charAt(i)*, selecciono dicha instrucción y pulso ctrl+shift+i.

4.4. Añadir componentes



Desde la pestaña *Help* en Eclipse podemos incorporar elementos a nuestro IDE. Podemos descargarlo desde un servidor repositorio externo desde la opción *Install New Software* o a través del *Eclipse MarketPlace*. Desde este último simplemente tenemos que buscar las etensiones que queremos incluir y pulsar sobre el botón instalar.

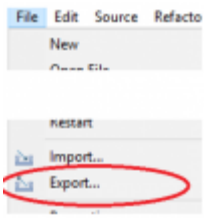
Interfaz gráfica de usuario

El plugin WindowBuilder nos permite crear ventanas a través de editores de ventanas *drag and drop*. Desde el menu *Nuevo* puedo seleccionar otros tipos de aplicaciones, → WindowBuilder → SwingDesigner → Jframe, por ejemplo. Esto crea una nueva perspectiva en la que puedo diseñar una ventana con un editor, y ver al mismo tiempo como se crea el código Java referente a mi ventana.

4.5. Librerías de clases

En Java el fichero estandar para empaquetar librerías de clases es el fichero ***Jar***. Podemos crear nuestras propias clases con métodos para usarlas en otro programa, empaquetándolas en un fichero jar.

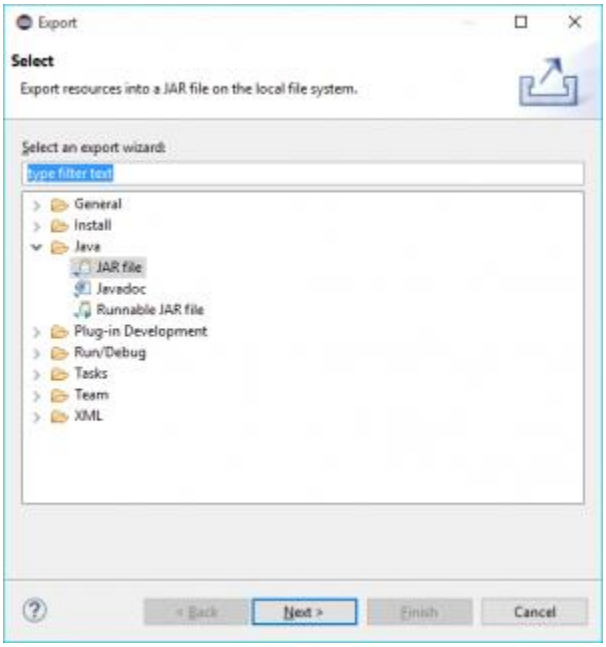
Todas las clases y los métodos definidos que guardaremos en una librería deben ser públicos (public)

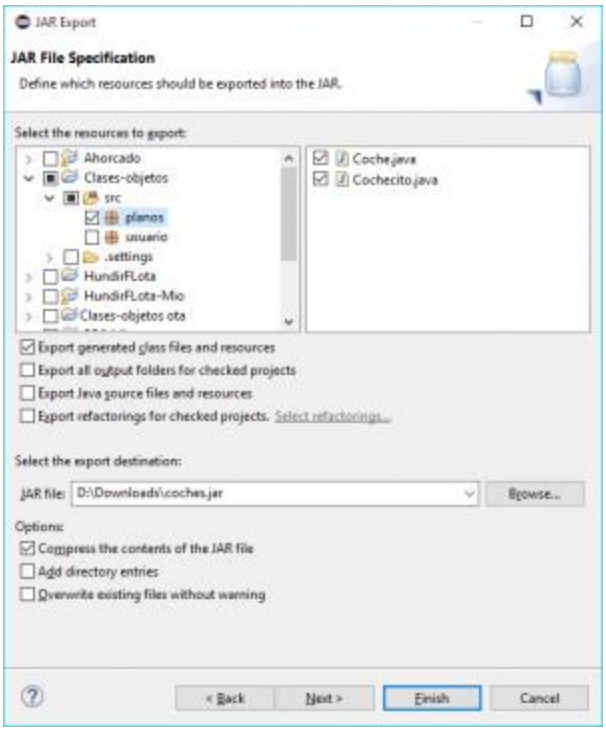


Para guardar en un fichero jar las clases que necesitamos usar en otros proyectos, debemos seleccionarlas para empaquetarlas. En Eclipse desde la pestaña *File*, seleccionaremos la opción *Export*.

Una vez en esta nueva ventana abriremos la sección *Java*, y pulsarmos sobre *JAR File*.

Después de pulsar sobre *JAR file*, se abrirá otra ventana en la que podemos seleccionar las clases que queremos guardar en nuestro JAR. Podemos seleccionar clases de distintos proyectos que queramos usar en otro.





Finalmente seleccionaremos un destino para almacenar nuestro archivo JAR. Podemos usar una carpeta nueva (libs, por ejemplo) en el workspace para almacenarlos.

4.6. Generación de Ejecutables

Ficheros Exe

Desde **Visual Studio** a través de la *Infraestructura de Lenguaje Común* los ejecutables que obtenemos después de construir el proyecto tiene extensión *.exe* y permiten ser ejecutados en cualquier máquina que tenga el framework .NET instalado.

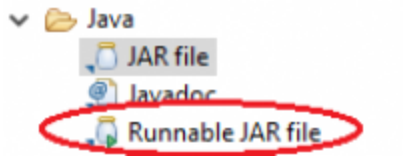
Los podemos encontrar en la carpeta `bin` dentro del directorio del proyecto de Visual Studio.

Ficheros Jar

Jar son las siglas de **J**ava **AR**chive, y es un tipo de fichero en el que podemos empaquetar las aplicaciones Java o empaquetar distintas clases. Son archivos comprimidos en formato **.zip** y cambiada su extensión a .jar. Así no tenemos que llevar nuestras carpetas y archivos sueltos de un lugar a otro. Dentro de un fichero JAR podemos empaquetar todos los recursos que tiene nuestro proyecto: ficheros .class, otras librerías, imágenes, y todo lo que esté en los directorios de nuestro proyecto.

En el directorio *bin* del JDK de Java tenemos el programa *jar* de java para crear estos ficheros, aunque también podemos crearlos desde el IDE para Java con el que trabajemos.

En Eclipse: **File → Export → Java → JAR File o Runnable JAR File**



Fichero Jar Ejecutable con programa consola: Si lo que quiero es obtener mi programa en un fichero JAR, debo crear un **Runnable JAR file**, indicando el proyecto y su clase principal (clase con método Main). Guardaremos el fichero Jar en nuestro PC.

Posteriormente para ejecutar ese programa JAR arrancaré el terminal de Windows (cmd). Para ejecutarlo iré al directorio donde lo tengo almacenado y ejecutaré: *java -jar archivoJar.jar* (donde *archivoJar.jar* es mi archivo). También puedo ejecutarlo desde cualquier lugar, dando la ruta de mi fichero jar.

Fichero Jar Ejecutable con Interfaz gráfica : Para guardar mi aplicación con interfaz de usuario en un JAR ejecutable, después de pulsar sobre exportar para crear un fichero JAR, debo indicarle que el tipo de JAR será **Runnable JAR file**.

Fichero Jar con Clases (Librería de clases): Desde File → Export → **JAR File**. En este caso no queremos un fichero Jar ejecutable, sino un contenedor de clases. Indico todas las clases de cualquier proyecto que quiero que estén en el Jar, y lo creo.

JSmooth

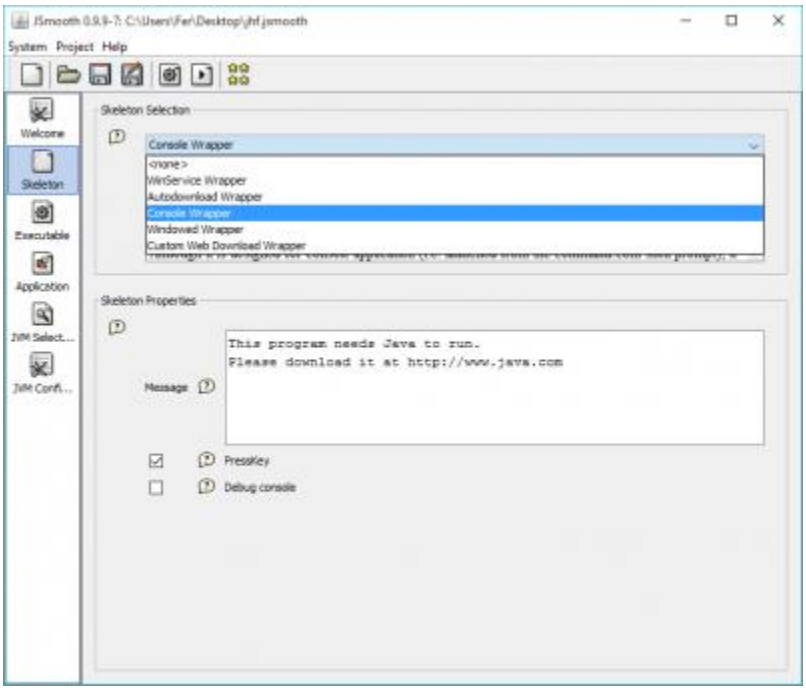
Cuando tenemos un programa en Java, podemos crear un ejecutable (Exe) en consola a partir de él. Tan solo necesitaremos haber exportado nuestro programa como un archivo Jar.

Una vez que tengamos instalado el programa JSmooth, gratuito, y descargable desde su página web, lo ejecutaremos.

Problema al iniciar JSmooth - JSmooth no reconoce Java: Al instalar Java hay algunas variables de entorno que no se crean de forma adecuada. Para usar JSmooth necesitamos crear una nueva *variable de sistema* llamada *JAVA_HOME* (Equipo → Propiedades → Configuración Avanzada → Opciones Avanzadas → Variables de entorno) y darle el valor de la ruta del JDK que estamos utilizando (p.e. → C:\Program Files\Java\jdk1.X.X.X).

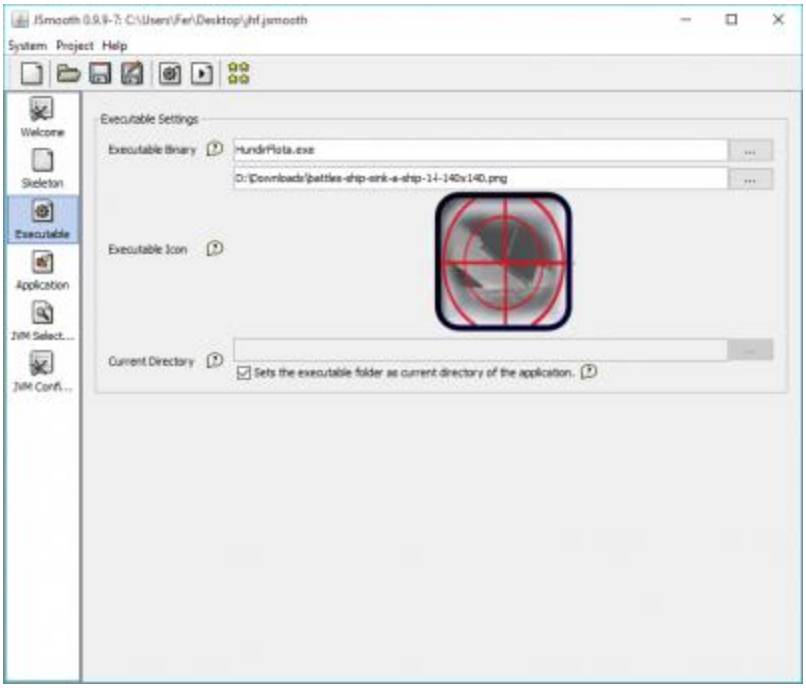
Sección Skeleton: Aquí indicamos simplemente de qué tipo de aplicación se trata, si de consola (Console Wrapper) o de Ventana (Windowed Wrapper). En esta sección configuramos qué mensaje aparecerá en caso de que el usuario no tenga instalada la máquina virtual de Java.

La opción **presskey** es útil en aplicaciones de consola, para que no se cierre la ventana del terminal automáticamente.

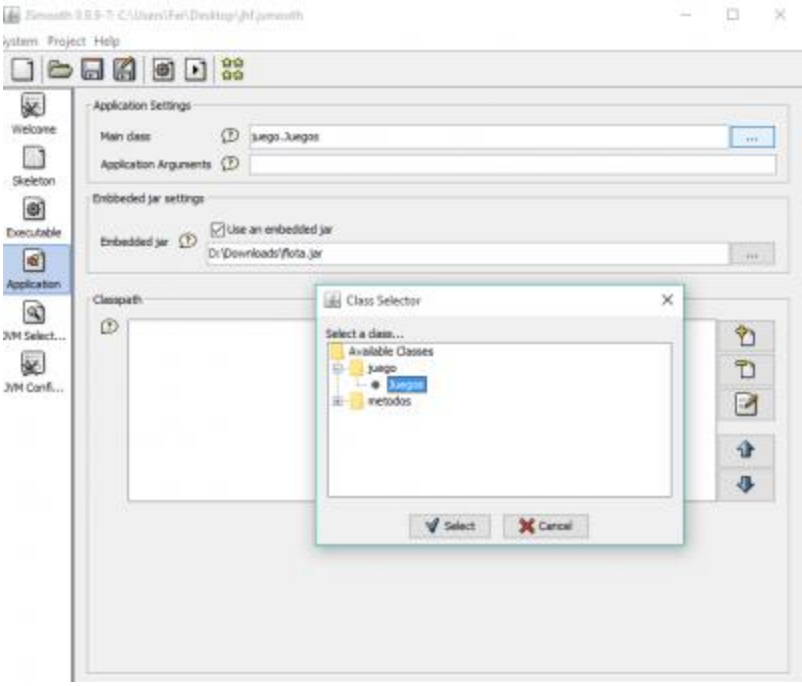


Sección Executable: Aquí indicaremos simplemente el nombre y el lugar donde creamos el fichero ejecutable (Executable Binary). También podemos seleccionar una imagen, para el icono de nuestro

ejecutable. Debe ser una imagen (p.e. PNG) pequeña, 120×120 o similar. En “Current Directory” indicamos simpemente el directorio por defecto que usará la aplicación cuando la ejecutemos. Podemos marcar el Checkbox ya que en principio no nos importa demasiado.



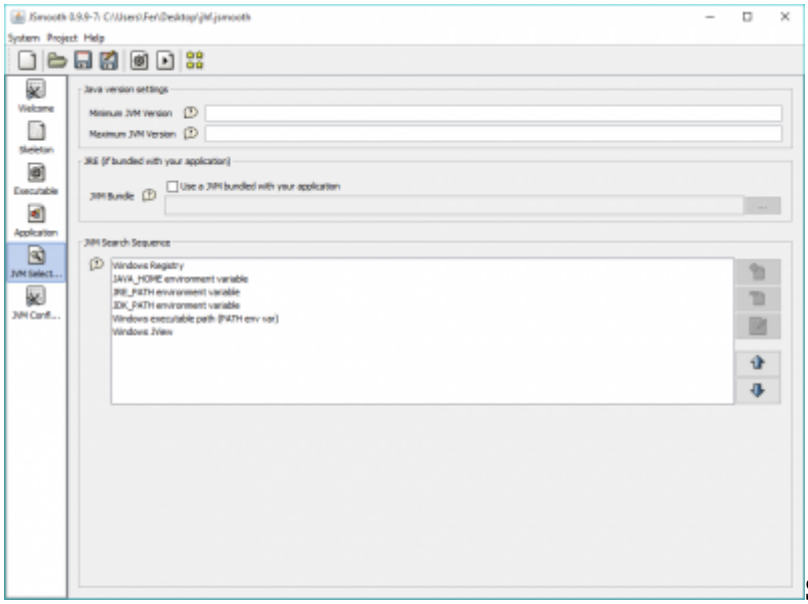
Sección Application: En esta sección debemos indicar dónde está nuestro archivo Jar en nuestro ordenador (Embbeded Jar Settings) marcando el checkbox. Después en “Application Settings” pulsaremos sobre Main Class para indicarle cuál es la clase de nuestro JAR que contiene el método ***main***, desde la que se iniciará el programa. Y está!!



Crear nuestro ejecutable: Pulsaremos sobre la pestaña “Project” y después sobre “Compile”. Nos pedirá de nuevo una ruta y un nombre para guardar un archivo de proyecto de JSmooth (no es el exe). También nos creará un fichero .exe en la ruta que le indicamos anteriormente. Ya podemos ejecutar nuestro programa.

Además también podemos configurar nuestra aplicación a partir de estas secciones:

Sección JVM Selection: Podemos indicarle la versión de Java mínima que necesitamos para ejecutar la aplicación. Esto se debe a que hay instrucciones de Java que han salido en una versión concreta. En principio no necesitamos indicar nada, a no ser que sepamos que nuestro programa no funciona en todas las versiones de Java. (Por ejemplo, Scanner está en java desde 1.5v, o “Java 5”). Actualmente

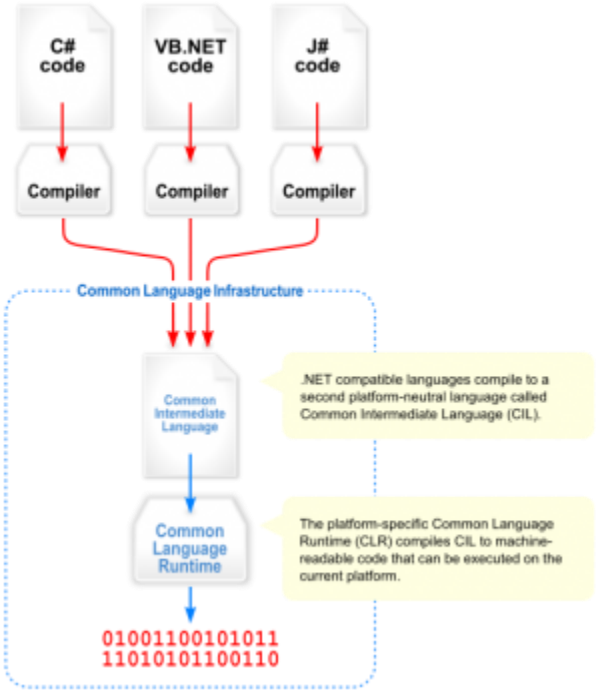
usamos Java 8 (1.8v) 

Sección JVM Configuration: Aquí podemos indicar la cantidad de memoria que usará nuestra aplicación de Java. Es algo que para aplicaciones pequeñas no es demasiado relevante.

5. VISUAL STUDIO.

Es un IDE creado por Microsoft y usado en plataformas Windows. Tiene licencia propietaria y soporta distintos lenguajes como C#, C++, Visual Basic .NET, Python, etc. Este IDE está preparado para trabajar con todas las tecnologías y lenguajes de la plataforma .NET de Microsoft.

5.1. Plataforma .NET



.NET es un framework creado por Microsoft principalmente para sistemas Windows. De forma parecida a la *plataforma Java*, .NET ofrece librerias de clases y un entorno de ejecución para diferentes lenguajes. A esta plataforma se le conoce como *CLI* (Common Language Infraestructure). Esta formada de 2 partes:

- Todos los programas escritos en el *framework* .NET se compilan a un lenguaje intermedio conocido como *CIL* (Common Intermediate Language)
- Posteriormente se ejecutan en un entorno de ejecución llamado *CLR* (Common Language Runtime), parecido a la máquina virtual de Java.

Esto hace que las aplicaciones .NET puedan ser independientes de la máquina, a través de un enfoque similar al que usa Java.

5.2. Lenguaje C#

Es un lenguaje de programación de propósito general, orientado a objetos y multiparadigma. Aunque ha sido muy desarrollado por Microsoft para su plataforma .NET, empleado como un Lenguaje de Infraestructura Común (CLI), C# es un lenguaje independiente que trabaja para otras plataformas.

La sintaxis del lenguaje *C# está basada en la sintaxis de C/C++*, de la misma forma que Java, por lo que la mayor parte de sus instrucciones se escriben y organizan de la misma forma. .NET tambien actua como API para el lenguaje C#. Podemos acceder a la documentación de la biblioteca de clases de .NET

Programa `Hola Mundo` en lenguaje C#:

```
class Saludo {
  static void Main() {
    System.Console.WriteLine("Hola Mundo!");

    System.Console.WriteLine("Pulsa una tecla para terminar");
    System.Console.ReadKey();
  }
}
```

Las instrucciones principales que necesitaremos para crear programas de consola son:

- Mostrar mensajes por consola

```
System.Console.Write("Mensaje"); //Escribe por consola, sin salto de linea
System.Console.WriteLine("Mensaje"); //Escribe por consola, con salto de linea
```

- Leer texto por consola

En C# se leen los datos siempre como String y posteriormente se convierten al tipo necesario. **C# no permite leer int directamente.**

```
System.Console.ReadLine(); //Lee la siguiente linea (String)
System.Console.ReadKey(); //Lee una tecla pulsada en código unicode.
```

- Convertir de String a otro tipo y viceversa

```
int numero = int.Parse("-466");  //Convierto de String a int
double numeroReal = double.Parse("32.345"); //Convierto de String a double

String cadena1 = numero.ToString(); //Convierto de int a String
String cadena2 = numeroReal.ToString(); //Convierto de int a String
```

- Espacio de nombres y sentencia *using*

Las clases en C# se organizan en espacios de nombres (namespaces). Un conjunto de clases que tengan relacion entre si, se agruparán bajo un mismo espacio de nombres. LA idea es similar a los paquetes (packages) en Java. Ejemplos de espacios de nombres:

- System → contiene clases como Console, String, y otros tipos de datos muy comunes.
- System.IO → Contiene clases para entreda y salida, manejo de ficheros, etc.
- System.Windows.Forms → Contiene clases para crear interfaces gráficas de usuarios.

```
namespace Aplicacion  //espacio de nombres 'Aplicacion'
{
    public class MiClase{   //Puedo acceder a esta clase indicando
'Aplicacion.miClase'

    }
}
```

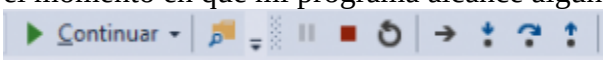
Por otra parte podemos usar la sentencia *using* al comienzo de una clase para indicar a nuestro compilador los espacios de nombres de las clases que vamos a usar (p.e. clase Console).

De esta forma podemos acceder a los métodos de la clase *Console* directamente sin necesidad de indicar el espacio de nombres (System):

```
using System;    //espacios de nombres que estoy usando
using System.IO;
namespace Aplicacion.parte1 //espacio de nombres de 'miClase'
{
    public class miClase{
       static void main(string[] args)
       {
          Console.WriteLine("No necesito escribir System para usar la clase
Console");
          Console.WriteLine("Pulse una tecla para terminar...");
          Console.ReadKey();
       }
    }
}
```

5.3. Depurador

Para utilizar la herramienta de depuración debo crear antes algún *breakpoint* pulsando doble clic sobre la barra vertical a la izquierda del número de linea. La perspectiva del modo depuración se iniciará en el momento en que mi programa alcance algún breakpoint.



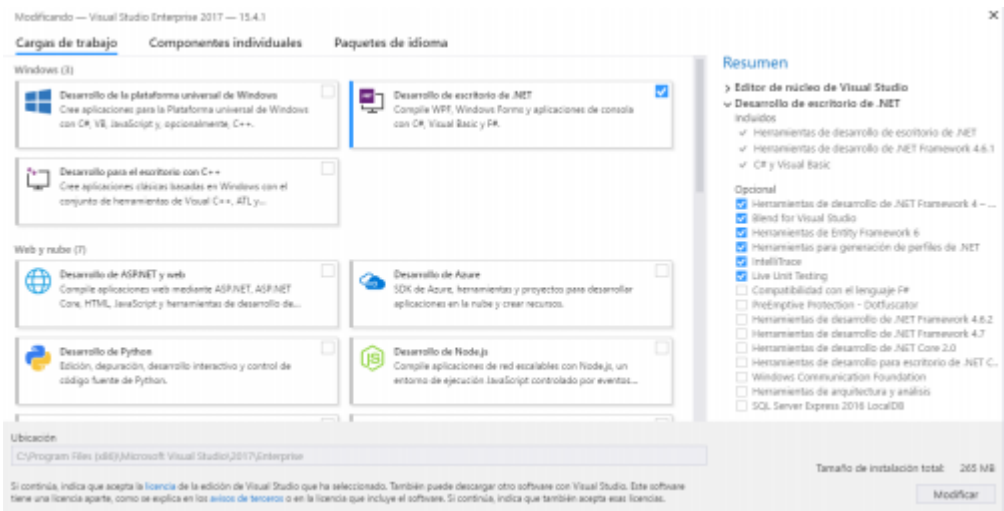
Para iniciar el modo depuración debo ir al menú *Depurar* y pulsar sobre *Iniciar Depuración* o también pusando la tecla F5.

Comando	Atajo de Teclado	Descripcion
Paso a paso por instrucciones	**F11**	Si la línea contiene una llamada a un método, entra en el método y se detiene en la primera línea de código incluida en él.
Paso a paso por procedimientos	**F10**	Si la línea contiene la llamada a un método, no entra en el método y continua en la siguiente isntrucción.
Paso a paso pasa salir	**Shift+F11**	Si estamos depurando un método, sale del método y continua en la siguiente instrucción.

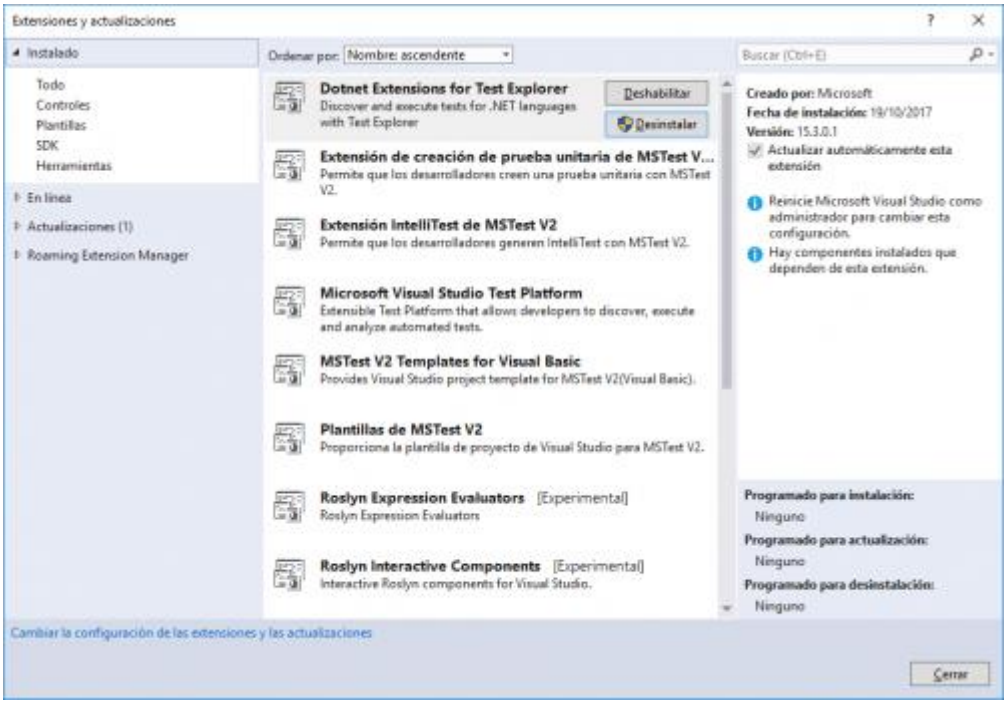
5.4. Añadir Componentes.

Puedo acceder a la herramienta para incorporar extensiones o módulos, desde el menú *Herramientas* con las opciones *Obtener herramientas y características* o con *Extensiones y actualizaciones*.

- Obtener herramientas y características

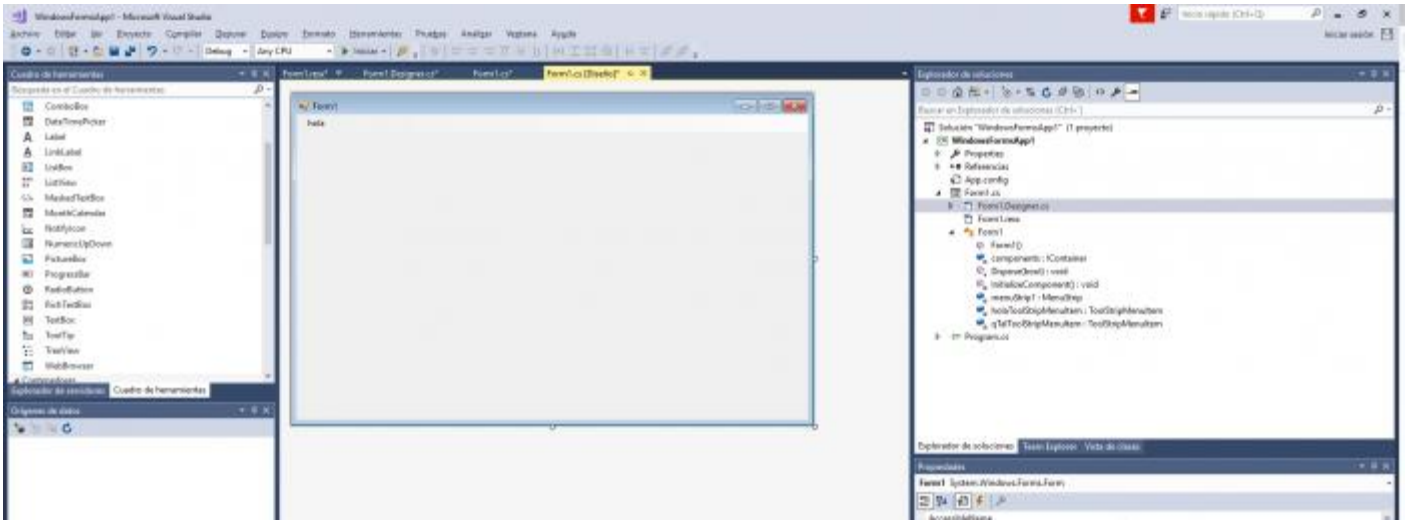


- Extensiones y actualizaciones



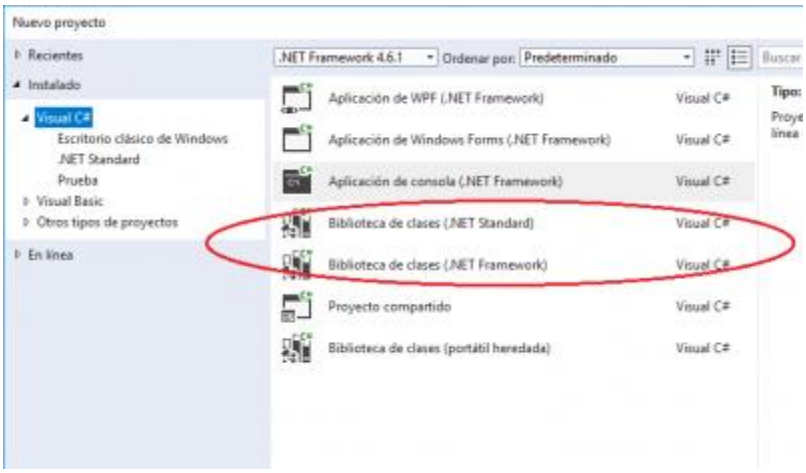
5.5. Interfaz gráfica de usuario.

Para crear una aplicación que ofrezca una interfaz gráfica debemos crear un proyecto de Windows Forms. Esto nos creará una clase en C# vinculada a un formulario que nos permite diseñar una ventana. Nos ofrece una paleta con componentes gráficos que podemos arrastrar a un formulario para diseñar nuestra interfaz de usuario (*GUI*).



5.5. Librerías de clases.

En la plataforma .NET uno de los tipos de ficheros que contienen librerías de clases se conocen como **ficheros .dll** (librerías de enlace dinámico). Para poder crear un proyecto de librerías de clases debo seleccionar un proyecto de Librerías de Clases.



Después en él crearemos las clases que necesitemos con los métodos que queramos. Y finalmente compilaremos la solución. De este modo en el diretorio ***bin/Debug*** de la carpeta del proyecto tendré el fichero *.dll* con las librerías que he creado.

Para incluirlas en otro proyecto, en el *Explorador de Soluciones* debo pulsar con el botón derecho sobre el proyecto en el que las quiero incluir. Despues **Agregar → Referencia** y ahí buscaré el *.dll* dentro de mi equipo y lo incorporaré al proyecto.

Todas las clases y los métodos definidos en una librería deben ser públicos (public) si quiero tener acceso a ellos.