



Guion de prácticas

NetBeans

Febrero 2020



Metodología de la Programación

DGIM

Curso 2019/2020

Índice

1. Introducción	5
2. Empezando	5
3. Un primer proyecto	7
3.1. Creación del proyecto NetBeans	7
3.2. Creación de un fichero fuente	8
3.3. Generando y ejecutando el binario	8
4. Un proyecto de compilación separada	9
4.1. El programa en un único fichero	9
4.2. Ejecutando el programa	9
4.3. Utilizando ficheros de entrada	9
4.4. Modularización: separando el programa en múltiples ficheros	10
4.4.1. Separar declaración e implementación	10
4.5. Crear la estructura del proyecto en la vista física	12
4.6. Configurar la vista lógica del proyecto	12
4.7. Configurar los parámetros del proyecto	12
5. Bibliotecas	15
5.1. Crear la estructura de ficheros	15
5.2. Usar la nueva biblioteca en el proyecto	16
6. Depurador	20
6.1. Conceptos básicos	20
6.2. Ejecución de un programa paso a paso	20
6.3. Inspección y modificación de datos	21
6.4. Inspección de la pila	21
6.5. Reparación del código	22
7. Otras características de NetBeans	22
8. Personalización de NetBeans	26
9. El programa original	27
9.1. Fichero de validación <code>v_0inside.test</code>	28
10. Apendice 2. Modularización	29
10.1. <code>Point2D.h</code>	29
10.2. <code>Rectangle.h</code>	29
10.3. <code>Point2D.cpp</code>	29
10.4. <code>Rectangle.cpp</code>	29
10.5. <code>main.cpp</code>	31

1. Introducción

Esta sesión de prácticas está dedicada a aprender a utilizar el entorno de desarrollo de software multi-language **NetBeans** como alternativa a la gestión de proyectos desde la línea de comandos. NetBeans es un entorno de desarrollo integrado libre y multiplataforma, creado principalmente para el lenguaje de programación Java, pero que ofrece soporte para otros muchos lenguajes de programación. Existe además un número importante de módulos para extenderlo. NetBeans es un producto libre y gratuito sin restricciones de uso. En este guión se explicará cómo utilizarlo en un **Linux** que ya tenga instalado **g++** y **make**.

2. Empezando

Es necesario instalar NetBeans con JDK ¹ y el plugin para **C++** ² instalados en ese orden. También se puede instalar el plugin de **C++** desde el menu “Tools-Plugins” del menu de NetBeans.

Una vez instalado se ejecuta desde la línea de comandos (para la versión 8.2)

```
/usr/local/netbeans-8.2/bin/netbeans
```

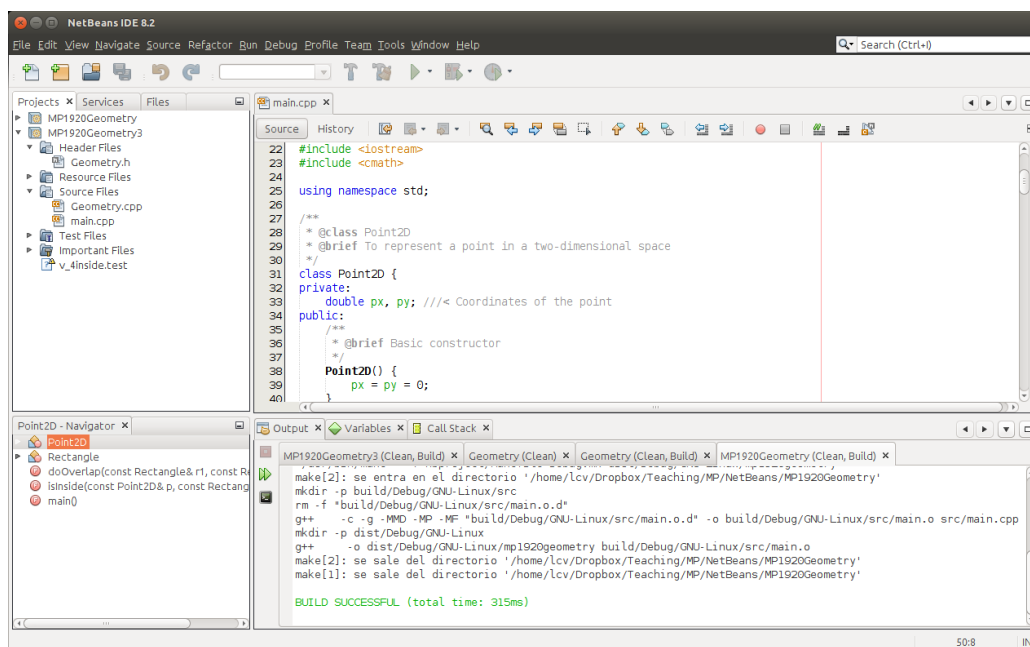


Figura 1: Entorno de trabajo en NetBeans con C++

En la Figura 1 se puede ver la distribución de las áreas de trabajo en NetBeans:

¹<http://www.oracle.com/technetwork/es/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html>

²<https://netbeans.org/features/cpp/>

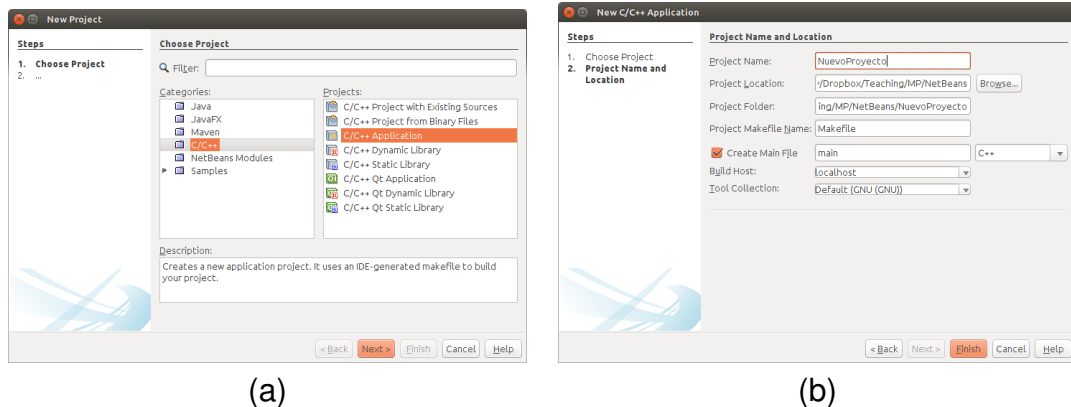
- La parte superior contiene un menú de opciones típico de un entorno de desarrollo de software del que se irá detallando lo más importante a lo largo de este guión. En cualquier caso, para más información se puede consultar la guía oficial de NetBeans³.
- El cuadrante superior izquierdo muestra el navegador de proyectos.
- El cuadrante superior derecho muestra las pestañas para editar ficheros fuente.
- El cuadrante inferior izquierdo, que muestra el contexto del código (funciones, pila, variables locales) durante las sesiones de depuración.
- El cuadrante inferior derecho que muestra múltiples pestañas asociadas con la ejecución del proyecto:
 - Output. Muestra las salidas estándar y de error y recoge la entrada estándar.
 - Terminal. Línea de comandos en la carpeta principal del proyecto.
 - Variables. Inspección de variables durante la depuración.
 - Call Stack. Estado de la pila de llamadas (depuración).
 - Breakpoints. Lista de puntos de ruptura activos (depuración).

³<https://netbeans.org/kb/docs/java/quickstart.html>

3. Un primer proyecto

3.1. Creación del proyecto NetBeans

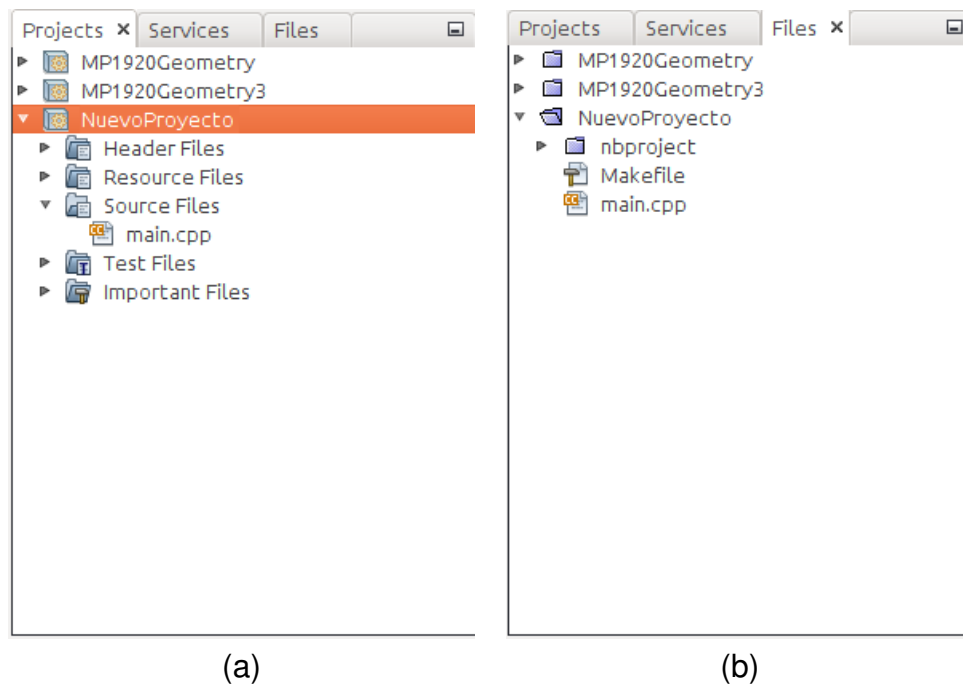
En el navegador de proyecto (cuadrante superior izquierda) con la pestaña "Projects" activa, pulsar con el botón derecho y seleccionar "New Project" (alternativamente "File" - "New Project"). Seleccionar el tipo de aplicación que se quiere construir (Figura 2.a), entre los múltiples lenguajes soportados por NetBeans, en este caso "C/C++ Application". Seleccionar el nombre del proyecto ("NuevoProyecto") y la ubicación que se le quiere dar en disco "Project location / Browse" (Figura 2.b).



(a) (b)

Figura 2: Creando un proyecto nuevo

En el navegador de proyectos (cuadrante superior izquierdo) aparecerán los árboles lógicos y físicos del proyecto recién creado (Figura 3).



(a) (b)

Figura 3: Árbol lógico (izquierda: pestaña "Projects") y físico (derecha: pestaña "Files") de un proyecto nuevo

3.2. Creación de un fichero fuente

Vamos a crear el primer fichero fuente del proyecto. Para ello, en el recién creado fichero **main.cpp** se introduce un primer programa en C++, como el conocido “Hola Mundo”⁴ (Figura 4).

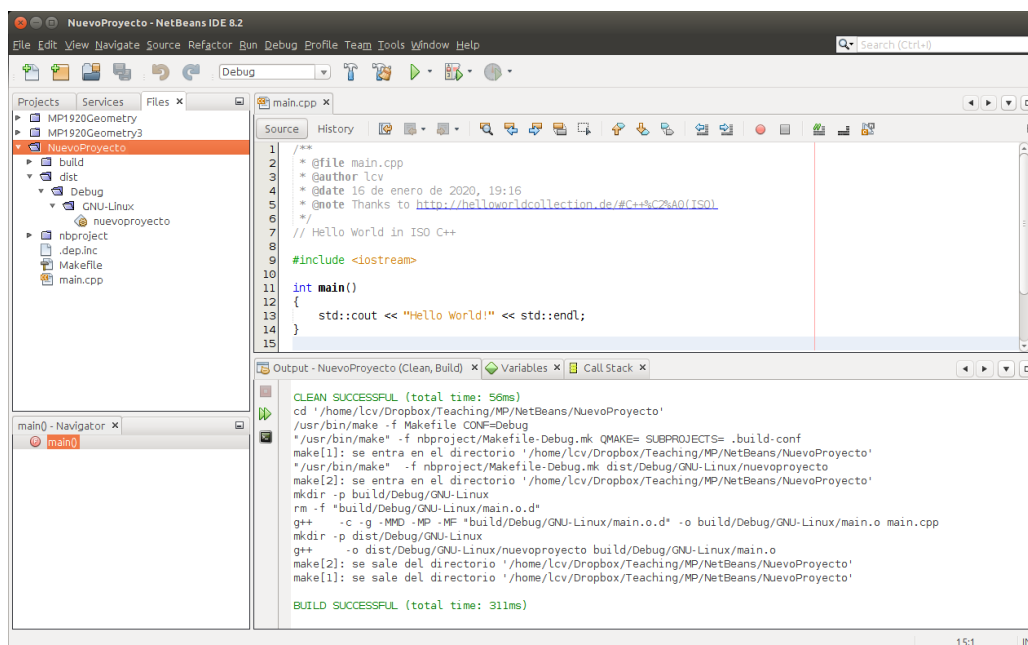


Figura 4: Creación de un nuevo fichero fuente en el proyecto

3.3. Generando y ejecutando el binario

Desde NetBeans se puede compilar el proyecto y ejecutarlo en un mismo paso:

Run - Clean and Build Project

Cuya salida aparece en el cuadrante inferior derecho, pestaña **Output** (Figura 4).

El binario recién generado se encuentra en la carpeta **dist** tal y como muestra el cuadrante superior derecho de la Figura 4.

La ejecución del binario que se acaba de generar se ordena como

Run - Run Project

Y se puede seguir su ejecución, desde la pestaña **Output** (Figura 5) como si fuese una consola de órdenes del sistema operativo.

⁴El programa “Hola mundo” en más de 200 lenguajes de programación <http://helloworldcollection.de>

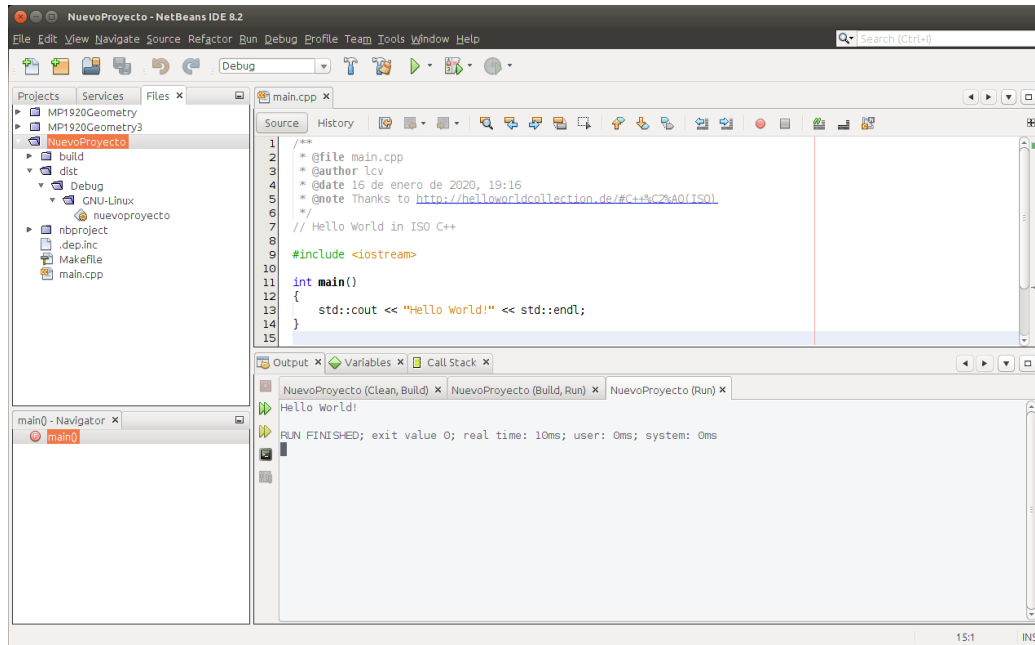


Figura 5: Ejecutando el binario

4. Un proyecto de compilación separada

Esta sección describe cómo crear un proyecto más complejo y dividirlo en múltiples ficheros.

4.1. El programa en un único fichero

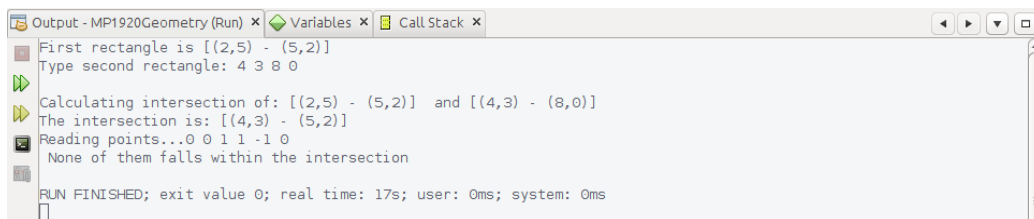
El objetivo del programa es calcular la intersección de dos rectángulos, leer una serie de puntos y calcular cuántos de esos puntos caen dentro de la intersección. Para ello se utilizan las clases **Point2D** y **Rectangle** que aparecen en el código escrito en un único fichero mostrado en la sección 9. Crear un nuevo proyecto llamado **MP1920Geometry** y copiar este código en el fichero **main.cpp**. Desde la vista de ficheros, crear una carpeta llamada **src** y mover el fichero dentro de esta carpeta.

4.2. Ejecutando el programa

Ejecutar el programa como se ha visto hasta ahora y seguir la secuencia que se muestra en la Figura 6.a) introduciendo los mismos datos. Otra forma de ejecutar el programa es desde la línea de comandos tal y como muestra la Figura 6.b).

4.3. Utilizando ficheros de entrada

Se pueden utilizar ficheros de datos de validación como el mostrado en la Sección 9. Para ello es conveniente crear una carpeta específica para almacenar estos ficheros de validación, la cual se podría llamar **tests**.



```

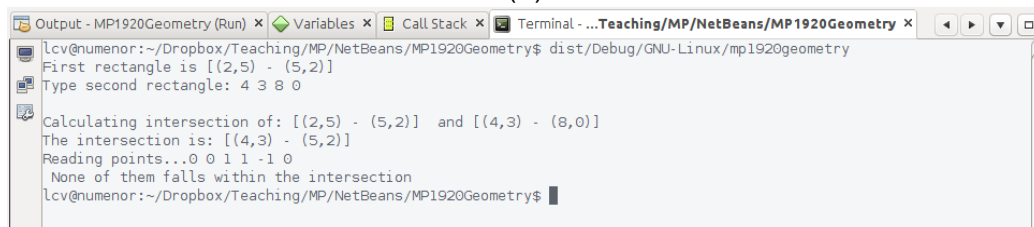
Output - MP1920Geometry (Run) x Variables x Call Stack x
First rectangle is [(2,5) - (5,2)]
Type second rectangle: 4 3 8 0

Calculating intersection of: [(2,5) - (5,2)] and [(4,3) - (8,0)]
The intersection is: [(4,3) - (5,2)]
Reading points...0 0 1 1 -1 0
None of them falls within the intersection

RUN FINISHED; exit value 0; real time: 17s; user: 0ms; system: 0ms

```

(a)



```

Terminal - ...Teaching/MP/NetBeans/MP1920Geometry x
lcv@numenor:~/Dropbox/Teaching/MP/NetBeans/MP1920Geometry$ dist/Debug/GNU-Linux/mp1920geometry
First rectangle is [(2,5) - (5,2)]
Type second rectangle: 4 3 8 0

Calculating intersection of: [(2,5) - (5,2)] and [(4,3) - (8,0)]
The intersection is: [(4,3) - (5,2)]
Reading points...0 0 1 1 -1 0
None of them falls within the intersection
lcv@numenor:~/Dropbox/Teaching/MP/NetBeans/MP1920Geometry$

```

(b)

Figura 6: Ejecutando el binario. a) Desde el menú “Run Project”. b) Desde la shell en la carpeta del proyecto

Desde la línea de comandos tan solo tenemos que redirigir la entrada desde este fichero

```
dist/Debug/GNU-Linux/mp1920geometry < tests/v_0inside.test
```

Si se quiere ejecutar esta redirección de datos de entrada para validar el programa desde el menú de Netbeans, es necesario acceder a las propiedades del proyecto

Project - Project Properties - Run - Run Command

e introducir las modificaciones según se muestra en la Figura 7 para indicar que el binario se va a ejecutar redireccionando la entrada estándar desde **v_0inside.test**.

4.4. Modularización: separando el programa en múltiples ficheros

En proyectos complejos se hace necesario dividir el programa en múltiples ficheros, de forma que cada uno de ellos soporte un parte de la implementación total. De esta forma, separaremos el programa original en los siguientes ficheros, tal y como se detalla en los listados de la Sección 10. Se separará la implementación de las clases y funciones en cada fichero **.cpp** mientras que las declaraciones de las clases y funciones se incluirán en unos ficheros **.h** llamados ficheros de cabeceras.

4.4.1. Separar declaración e implementación

El primer paso para hacer esta separación es desvincular la declaración de las clases de su implementación concreta. Para ello, dada una implementación primaria como la de la Figura .

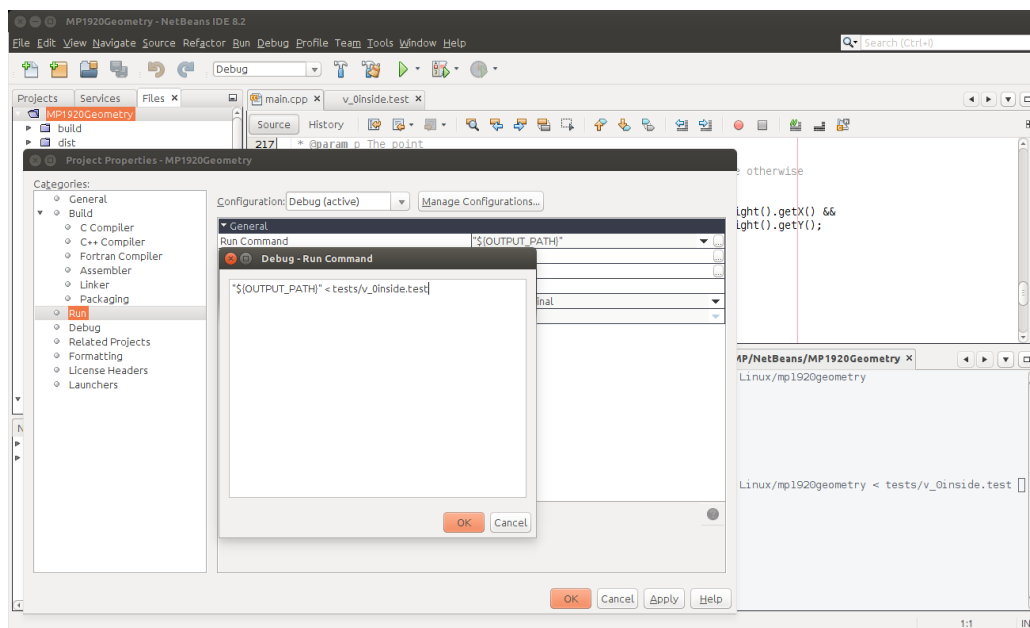


Figura 7: Ejecutando el binario con argumentos o redireccionamiento de entrada

El siguiente paso es separar aún más estos ítems: las declaraciones irán a un fichero con extensión **.h** y la implementación a un fichero con extensión **.cpp**

1. Módulo **Point2D**: declarado en **Point2D.h** e implementado en **Point2D.cpp**. Contiene el código para manejar el tipo de dato **Point2D**.
2. Módulo **Rectangle**: declarado en **Rectangle.h** e implementado en **Rectangle.cpp**. Contiene el código para manejar el tipo de datos **Rectangle**. Hace uso del módulo **Point2D**.
3. Módulo **main**: implementado en **main.cpp**. Contiene el código que implementa el programa de cálculo de la intersección de los rectángulos y de los puntos que caen dentro de ella. Hace uso de los módulos **Point2D** y **Rectangle**.

Tenga en cuenta que para evitar dobles inclusiones, los ficheros **.h** deben contener directivas del preprocesador de la forma siguiente (incluidas automáticamente por NetBeans al crear un nuevo fichero de cabeceras):

```
#ifndef _FICHERO_H_
#define _FICHERO_H_
...
#endif
```

Para poder compilar bien los ficheros **cpp** deberán incluirse donde sea necesario, los ficheros **.h** con la directiva:

```
#include "Point2D.h"
#include "Rectangle.h"
```

<pre>#include <iostream> using namespace std; class HelloWorld { private: string message; public: HelloWorld() { message = "Hello_world!"; } void print() { cout << endl << message << endl; } void set(string s) { message = s; } }; int main() { HelloWorld hw; hw.set("Hola_mundo!"); hw.print(); return 0; }</pre>	<pre>#include <iostream> using namespace std; class HelloWorld { private: string message; public: HelloWorld(); void print() const; void set(const string &s); }; int main() { HelloWorld hw; hw.set("Hola_mundo!"); hw.print(); return 0; } HelloWorld::HelloWorld() { message = "Hello_world!"; } void HelloWorld::print() const { cout << endl << message << endl; } void HelloWorld::set(const string &s) { message = s; }</pre>
---	--

Figura 8: Separar declaración/definición de implementación. Obsérvese a la derecha el uso del cualificador `HelloWorld::` en las implementaciones de los métodos para indicar que se trata de métodos de esa clase, no de funciones genéricas

4.5. Crear la estructura del proyecto en la vista física

De esta forma, será necesario organizar la carpeta interna del proyecto para que todo esté en su sitio, tal y como muestra la Figura 9.

Para crear esta estructura en NetBeans hay que hacerlo desde la pestaña “Files” del navegador de proyectos, crear la estructura que muestra la Figura 9 quedando como muestra la Figura 10.a)

4.6. Configurar la vista lógica del proyecto

En la vista lógica del proyecto, pestaña “Projects” es necesario indicar en qué carpeta se encuentran los ficheros `.h` y `.cpp`. Para ello se procede como sigue:

1. Header files. Pulsar con el botón derecho, “Add existing item...” y navegar hasta donde están los ficheros `.h`, seleccionarlos y añadirlos.
2. Source files. Pulsar con el botón derecho, “Add existing item...” y navegar hasta donde están los ficheros `.cpp`, seleccionarlos y añadirlos.

Esta operación no añade nuevos ficheros al proyecto, tan sólo le indica a NetBeans dónde están estos. Al concluir esta etapa, el proyecto aparece como muestra la Figura 10.b).

4.7. Configurar los parámetros del proyecto

Ya casi está terminado. Las opciones del proyecto se definen a cabo desde la pestaña de propiedades del proyecto (Figura 11), la cual aparece

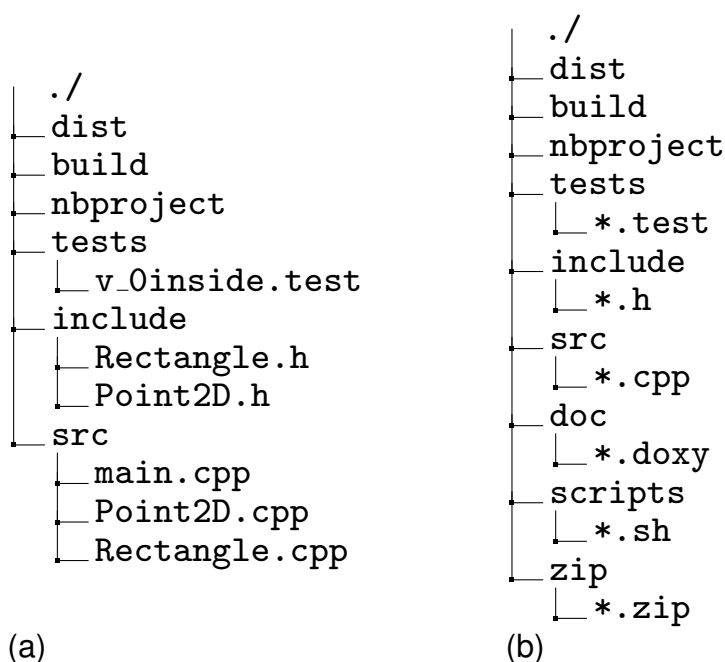


Figura 9: a) Estructura básica de las carpetas de un proyecto con múltiples módulos. Las tres primeras subcarpetas (**dist**, **build** y **nbproject**) son creadas y mantenidas automáticamente por NetBeans. El resto (**tests**, **src**, **include**) deben crearse a mano en cada proyecto. b) Estructura extendida de carpetas para la gestión de un proyecto que incluyen las carpetas **doc** (para almacenar ficheros Doxygen y documentación del proyecto), **scripts** para almacenar scripts de gestión del mismo como las que se puede descargar de Prado, y la carpeta **zip** para guardar copias de seguridad del proyecto.



Figura 10: Estructura de carpetas del proyecto, en su visión física (a) y lógica (b)

con botón derecho, “Project properties”, “C++ compiler”, “Include directories” y se selecciona la carpeta “include” perteneciente al proyecto.

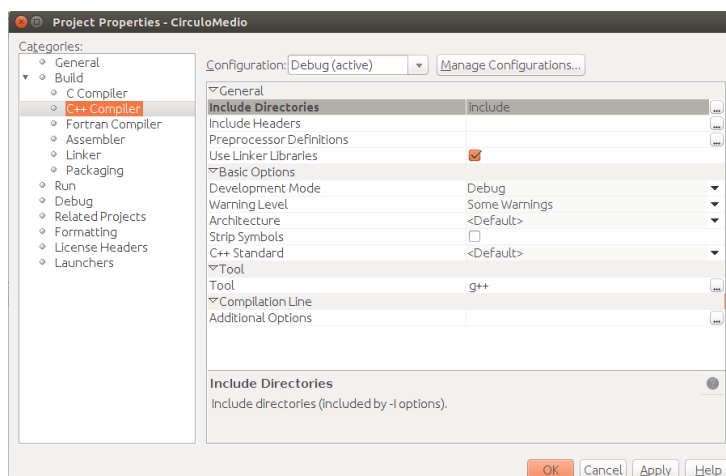


Figura 11: Propiedades del proyecto

1. Modo de depuración o de producción. Para recompilar todo el proyecto se elige en el campo **Configuration** y se elige **Debug** o **Release** respectivamente.
2. “C++ Standard”. Selecciona bajo qué estándar de C++ se va a compilar el proyecto. Seleccionaremos siempre “C++14”.

Para introducir parámetros en la llamada al programa o redireccionamientos de la entrada o salida del programa (por ejemplo con ficheros de validación de datos) es necesario modificar las propiedades del proyecto.

5. Bibliotecas

En NetBeans cada proyecto que se crea sirve para generar un único binario, de forma que para generar más de un binario habrá que crear más de un proyecto. Igualmente pasa con las bibliotecas para las que hace falta crear un proyecto individual (Figura 12). Desde el menú **File** o desde el navegador de proyectos será necesario crear un nuevo proyecto e indicar **C/C++ Static Library**, en vez de **C/C++ Application**. Le daremos un nombre al proyecto, que será el nombre de la biblioteca **Geometry** y se gestionará como un proyecto independiente de NetBeans.

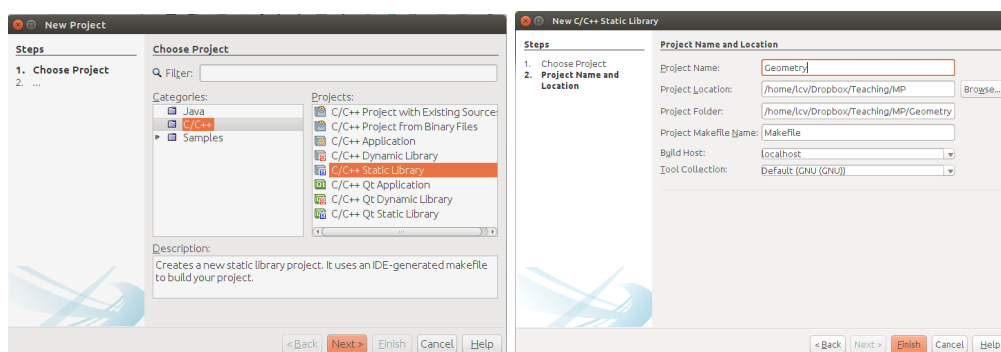


Figura 12: Creando un nuevo proyecto para gestionar una biblioteca

5.1. Crear la estructura de ficheros

1. En el árbol físico del proyecto de la biblioteca crear la estructura necesaria (carpetas **src** e **include** nada más) e incluir los ficheros oportunos (Figura 13.a).
2. Configurar el proyecto para definir la carpeta de ficheros de cabeceras igual que se hizo en el proyecto anterior (ver Figura 11) en el parámetro **Include Directories**.
3. En el árbol lógico del proyecto, añadir los **Header Files** y los **Source Files** que acabamos de duplicar usando el menú contextual (botón derecho del ratón) **Add Existing Item** y seleccionando los ficheros que acabamos de introducir (Figura 13.b).
4. Construir el proyecto de biblioteca (**Clean and Build Project**) igual que se hizo en el proyecto anterior, con la diferencia de que, en este caso, no se ha generado un binario, sino una biblioteca. En la Figura 14 se puede ver cómo NetBeans llama adecuadamente al compilador para compilar las fuentes y, posteriormente, al programa **ar** para generar la biblioteca. La biblioteca recién generada se encuentra en la carpeta **dist** tal y como muestra la Figura 15.a).

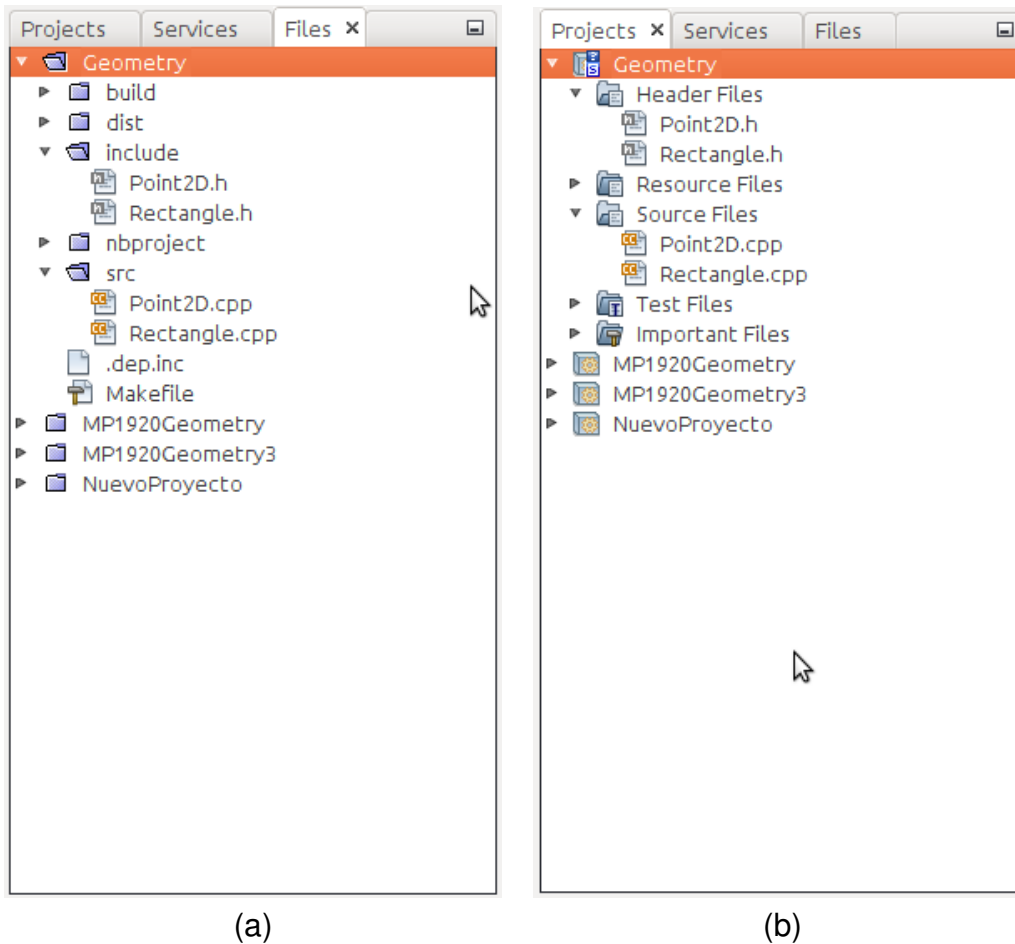
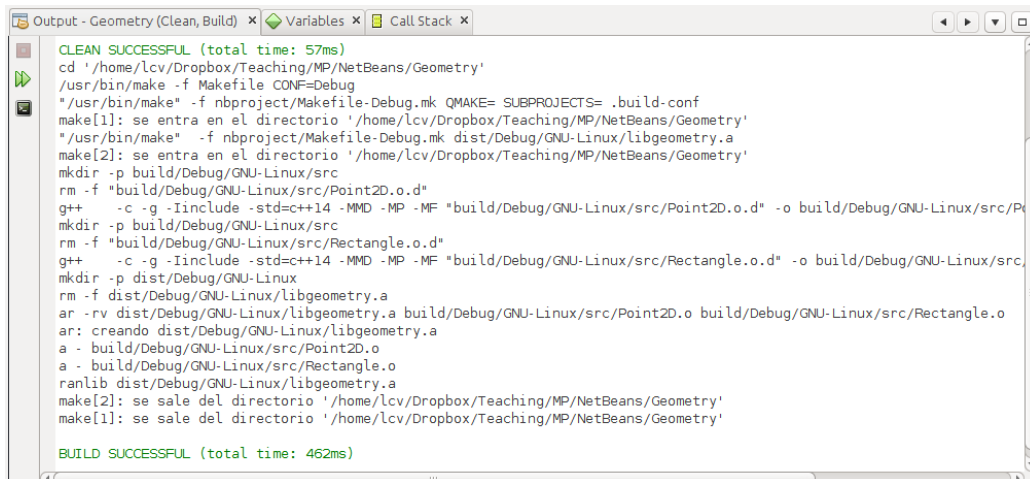


Figura 13: Vistas física (a) y lógica (b) del nuevo proyecto de biblioteca

5.2. Usar la nueva biblioteca en el proyecto

Siguiendo el ejemplo de la intersección de rectángulos, el nuevo proyecto, que constaría de dos proyectos vinculados entre sí, uno para la biblioteca recién creada y otro para el programa principal, que debería constar sólo del fichero **main.cpp**, el resto de ficheros desaparecen del proyecto.

1. En el árbol lógico del proyecto los ficheros .h y .cpp ya no aparecen porque han pasado a formar parte de la biblioteca y ya no se van a usar más desde aquí, sino desde el proyecto de la biblioteca. El nuevo árbol lógico queda como muestra la Figura 15.b).
2. Integración de la biblioteca en el proyecto. Esto se puede hacer de dos formas distintas.
 - a) Colocar los ficheros .h y .a de la nueva biblioteca en las carpetas que Linux tiene preparadas para esto y que son **/usr/local/include** y **/usr/local/lib** para que se puedan reutilizar de ahora en adelante por todos los proyectos futuros.
 - b) Vincular los proyectos de NetBeans del programa principal y la biblioteca. Para ello se selecciona el menú



```

Output - Geometry (Clean, Build) x Variables x Call Stack x
CLEAN SUCCESSFUL (total time: 57ms)
cd '/home/lcv/Dropbox/Teaching/MP/NetBeans/Geometry'
/usr/bin/make -f Makefile CONF=Debug
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: se entra en el directorio '/home/lcv/Dropbox/Teaching/MP/NetBeans/Geometry'
"/usr/bin/make" -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux/libgeometry.a
make[2]: se entra en el directorio '/home/lcv/Dropbox/Teaching/MP/NetBeans/Geometry'
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/Point2D.o.d"
g++ -c -g -Iinclude -std=c++14 -MMD -MP -MF "build/Debug/GNU-Linux/src/Point2D.o.d" -o build/Debug/GNU-Linux/src/Point2D.o
mkdir -p build/Debug/GNU-Linux/src
rm -f "build/Debug/GNU-Linux/src/Rectangle.o.d"
g++ -c -g -Iinclude -std=c++14 -MMD -MP -MF "build/Debug/GNU-Linux/src/Rectangle.o.d" -o build/Debug/GNU-Linux/src/Rectangle.o
mkdir -p dist/Debug/GNU-Linux
rm -f dist/Debug/GNU-Linux/libgeometry.a
ar -rv dist/Debug/GNU-Linux/libgeometry.a build/Debug/GNU-Linux/src/Point2D.o build/Debug/GNU-Linux/src/Rectangle.o
ar: creando dist/Debug/GNU-Linux/libgeometry.a
a - build/Debug/GNU-Linux/src/Point2D.o
a - build/Debug/GNU-Linux/src/Rectangle.o
ranlib dist/Debug/GNU-Linux/libgeometry.a
make[2]: se sale del directorio '/home/lcv/Dropbox/Teaching/MP/NetBeans/Geometry'
make[1]: se sale del directorio '/home/lcv/Dropbox/Teaching/MP/NetBeans/Geometry'

BUILD SUCCESSFUL (total time: 462ms)

```

Figura 14: Creación de la biblioteca **libgeometry.a** de forma automática desde NetBeans

Projects - Project properties - Related Projects

y se selecciona la carpeta donde está el proyecto de la biblioteca, asegurandose de marcar la casilla “Build”. De esta forma, cualquier cambio en la biblioteca haría que se recompilasen de nuevo, no solo la biblioteca, sino también el nuevo proyecto.

3. Sea cual sea la opción elegida, en las propiedades del proyecto se debe añadir la carpeta donde se encuentren los ficheros .h en la opción **Include Directories**, es decir, bien la carpeta incluye del sistema, bien la carpeta incluye del proyecto de la biblioteca. Los includes del código se pueden dejar con comillas dobles o como ángulos indiferentemente a partir de ahora.
4. En las propiedades del proyecto pero en las opciones del enlazador (**Linker** Figura 17), añadir la biblioteca recién creada en la opción **Libraries**. Elegir la ubicación de la misma igual que en el apartado anterior.
5. Crear el nuevo binario (**Clean and Build Project**).

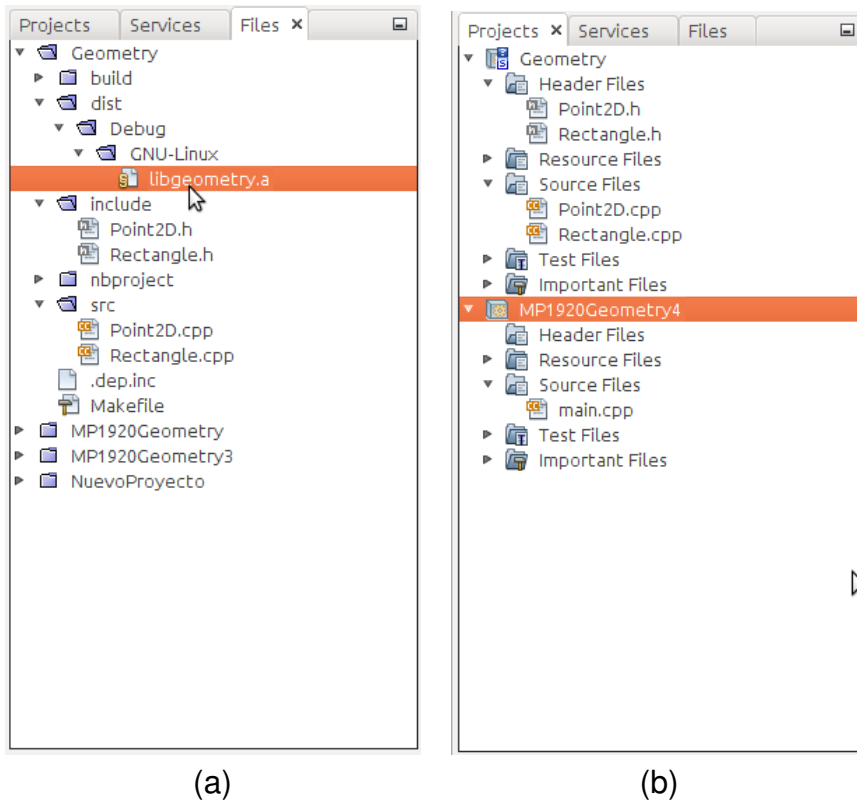


Figura 15: (a) Situación de la biblioteca **libgeometry.a** en el árbol físico del proyecto dentro de la carpeta **dist** (b) Nuevo árbol lógico del proyecto, que excluye a todos los **.h** y **.cpp** que se ya no pertenecen al proyecto principal sino a la biblioteca recién creada

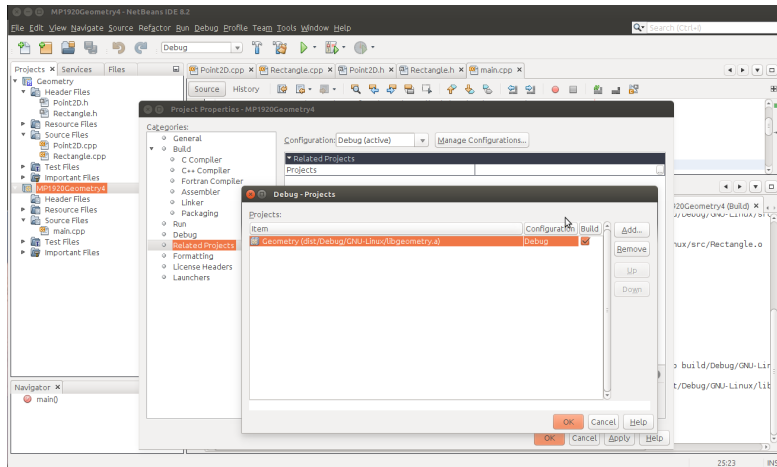


Figura 16: Configuración del proyecto para vincular el nuevo proyecto al proyecto de la biblioteca

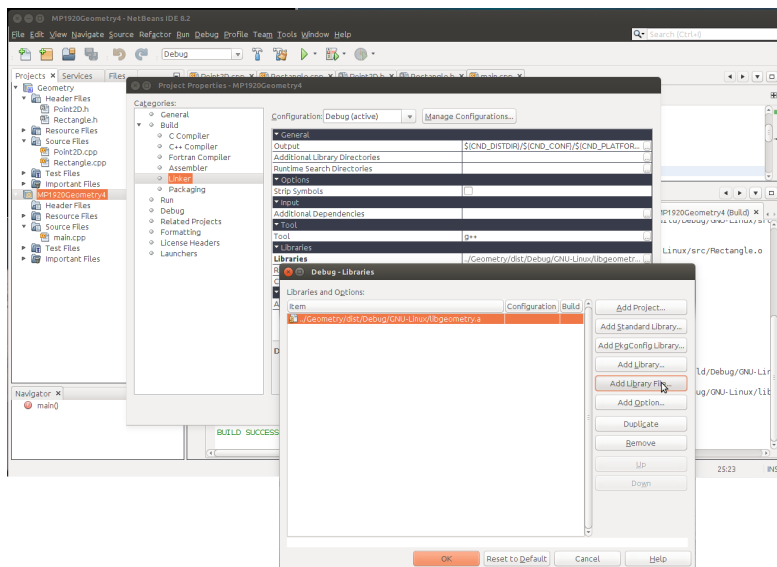


Figura 17: Incluyendo la biblioteca **libgeometry.a** para enlazarla

6. Depurador

6.1. Conceptos básicos

NetBeans actúa, además, como una interfaz separada que se puede utilizar con un depurador en línea de órdenes. En este caso será la interfaz de alto nivel del depurador **gdb**. Para poder utilizar el depurador es necesario compilar los ficheros fuente con la opción **-g** o, lo que es lo mismo, con la configuración **Debug** en la ventana de propiedades del proyecto NetBeans.

6.2. Ejecución de un programa paso a paso

Para comenzar a ejecutar un programa bajo control del depurador es conveniente colocar un punto de ruptura⁵. NetBeans permite crear un PR simplemente haciendo click sobre el número de línea correspondiente en la ventana de visualización de código y visualiza esta marca como una línea en rojo (Figura 18).

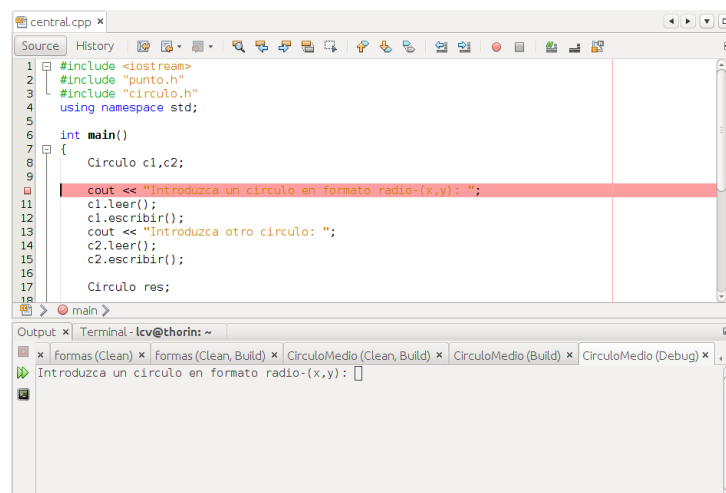


Figura 18: Creando un punto de ruptura para depurar un programa. El PR aparece como una línea de color rojo

Una vez colocado este punto de ruptura se puede comenzar la ejecución del programa con el menú⁶

Debug - Debug Project

NetBeans señala la línea de código activa con una pequeña flecha verde a la izquierda de la línea y remarca toda la línea en color verde (Figura 19). A la misma vez, se abren una serie de pestañas adicionales en el cuadrante inferior derecho, tal y como se comentó en la Sección 2.

⁵Un punto de ruptura (abreviadamente PR) es una marca en una línea de código ejecutable de forma que su ejecución siempre se interrumpe antes de ejecutar esta línea, pasando el control al depurador

⁶Consultar <https://netbeans.org/kb/docs/cnd/debugging.html> para una descripción más detallada de las funciones de depuración de NetBeans, tanto para C++ como para otros lenguajes.

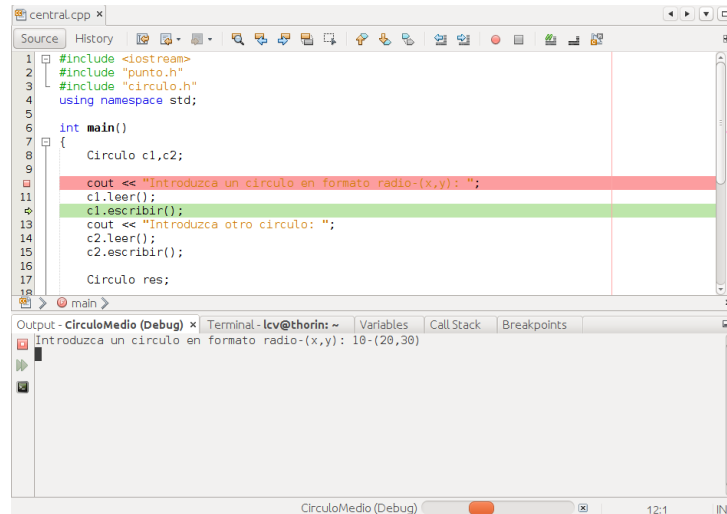


Figura 19: Ejecutando un programa paso a paso. La línea que se va a ejecutar a continuación aparece marcada en color verde

Las siguientes son algunas de las funciones más habituales de ejecución paso a paso:

- **Debug - Step Into**
Ejecuta el programa paso a paso y entra dentro de las llamadas a funciones o métodos.
- **Debug - Step Over**
Ejecuta el programa paso a paso sin entrar dentro de las llamadas a funciones o métodos, las cuales las resuelve en un único paso.
- **Debug - Continue**
Ejecuta el programa hasta el siguiente punto de ruptura o el final del programa.

6.3. Inspección y modificación de datos

NetBeans, como cualquier depurador, permite inspeccionar los valores asociados a cualquier variable y modificar sus valores sin más que acceder a la pestaña **Variables** y desplegar las variables deseadas y modificarlas, en caso necesario (Figura 20).

6.4. Inspección de la pila

Durante el proceso de ejecución de un programa se suceden llamadas a módulos que se van almacenando en la pila. NetBeans ofrece la posibilidad de inspeccionar el estado de esta pila y analizar qué llamadas se están resolviendo en un momento dado de la ejecución de un programa (Figura 21).

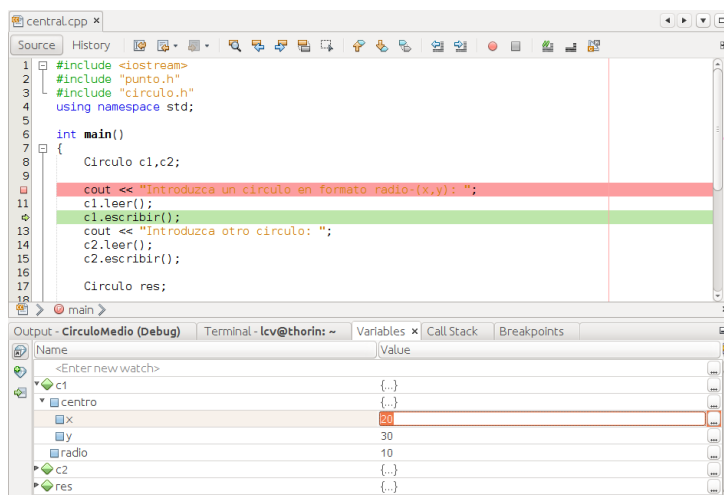


Figura 20: Inspeccionado y modificando, si fuese necesario, las variables del programa

6.5. Reparación del código

Durante una sesión de depuración es normal que sea necesario modificar el código para reparar algún error detectado. En este caso es necesario mantener bien actualizada la versión del programa que se encuentra cargada. Para ello lo mejor es interrumpir la ejecución del programa

Debug - Finish Debugger Session
y re-escribir los cambios y recompilarlos (**Clean and Rebuild Project**).

7. Otras características de NetBeans

Además de estas características, el editor de código de NetBeans dispone de algunas ayudas a la escritura de código que incrementan enormemente la eficiencia y la detección temprana de errores de programación. Estas son algunas de estas características.

1. Ayuda a la escritura de llamadas a métodos y funciones (Figura 22). Mientras se escribe la llamada a un método se muestran las distintas posibilidades, sus parámetros y la información de ayuda.
2. Ayuda a la escritura de llamadas a métodos y funciones (Figura 23). Si se escribe una llamada a un método inexistente o con una llamada incorrecta, NetBeans lo señala inmediatamente con un icono rojo en la línea de la llamada.
3. Ayuda a la escritura de variables (Figura 24). Si se escribe una variable no declarada aún, NetBeans lo señala inmediatamente con un icono rojo en la línea de la llamada.
4. Ayuda a la indentación de código (Figura 25). Cada vez que se escribe código en una ventana, al pie de la misma aparece una indicación del nivel de anidamiento de código en el que se encuentra.

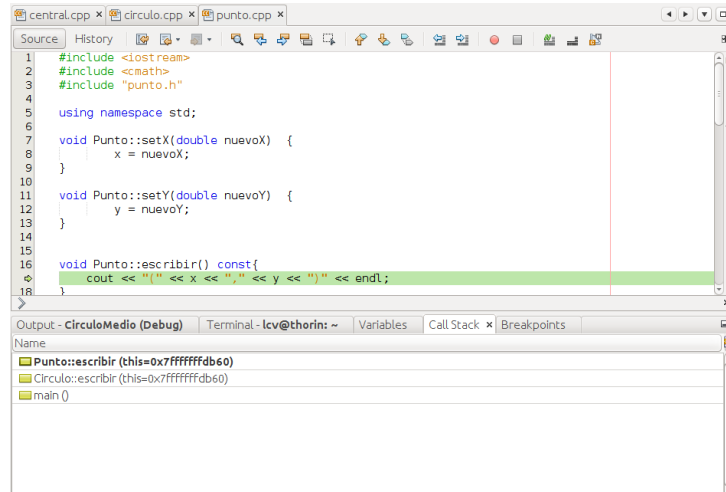


Figura 21: Inspeccionado y modificando, si fuese necesario, las pila de llamadas del programa

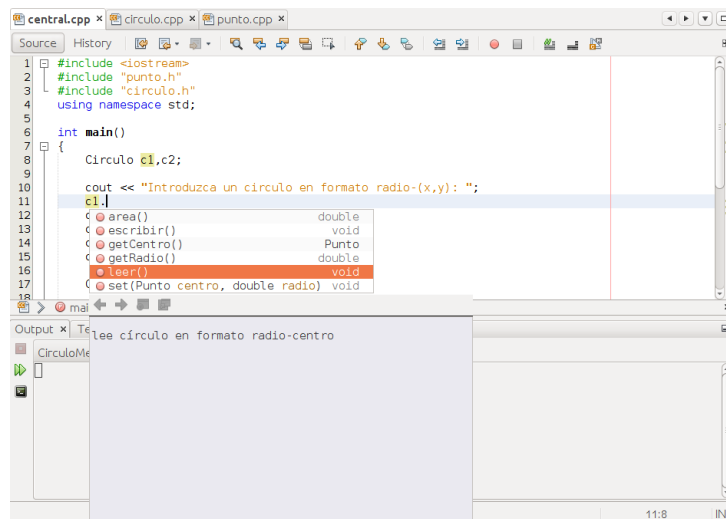


Figura 22: Auto-relleno de código

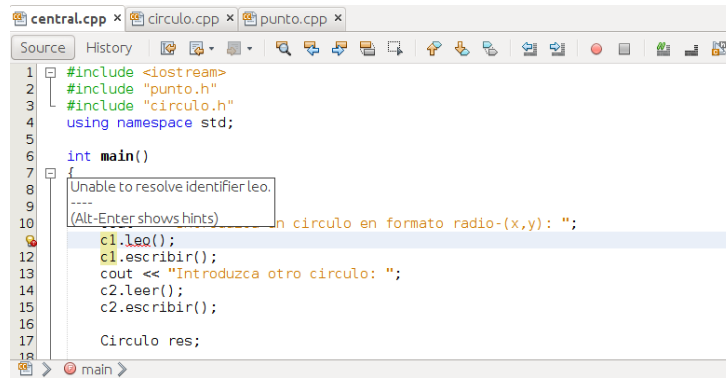


Figura 23: Detección inmediata de llamadas erróneas

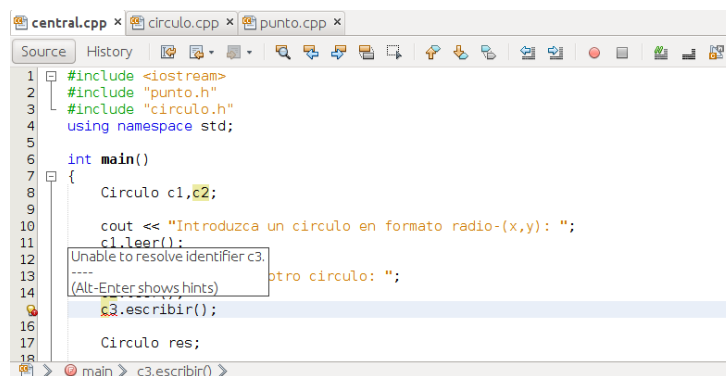
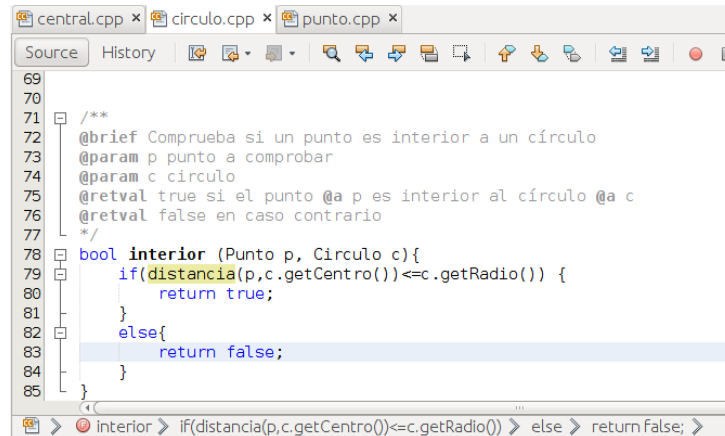


Figura 24: Detección inmediata del uso de identificadores erróneos



```

69
70
71 /**
72  * @brief Comprueba si un punto es interior a un círculo
73  * @param p punto a comprobar
74  * @param c círculo
75  * @retval true si el punto @a p es interior al círculo @a c
76  * @retval false en caso contrario
77  */
78 bool interior (Punto p, Circulo c){
79     if(distancia(p,c.getCentro())<=c.getRadio()) {
80         return true;
81     }
82     else{
83         return false;
84     }
85 }

```

Figura 25: Detección inmediata del nivel de anidación del código. En la parte inferior del editor se puede observar la localización precisa de la línea que se está editando (línea número 83): Función **interior**, dentro de un **if** y en la parte **else** del mismo

8. Personalización de NetBeans

NetBeans puede ejecutar scripts del sistema operativo (Linux) para realizar tareas externas al propio entorno, normalmente para operaciones de mantenimiento del proyecto o para llamar a programas externos. En esta asignatura se propone el uso de tres de estas scripts, las cuales se encuentran disponibles en Prado, y que es aconsejable tenerlas reunidas en una misma carpeta, por ejemplo **scripts**. Son los que aparecen a continuación (los cuales pueden ser modificados o ampliados a criterio del alumno).

doConfig.sh Configura parámetros básicos del proyecto. No es necesario usarla directamente sino que está diseñada para ser usada por las demás scripts para autoconfigurarse dentro de los márgenes del proyecto NetBeans.

doDoxygen.sh Según el fichero **.doxy** que se haya guardado en la carpeta **doc/** del proyecto, genera la documentación de Doxygen de forma automática y en formato **html** y **latex**. Si la carpeta no existe, se crea automáticamente.

doZipProject.sh Genera un zip comprimido de las carpetas más importantes del proyecto para llevarlo a otro ordenador o para entregar las prácticas en Prado. Coloca el fichero zip en la carpeta **zip/** y, si no existe, la crea.

doTests.sh Accede a la carpeta **tests** del proyecto y, por cada fichero **.test** que haya en ella, ejecuta el programa con redirección de entrada desde ese fichero. Atención, todos y cada uno de estos ficheros deben estar preparados con este propósito.

9. El programa original

Todos estos ficheros están disponibles en Prado para su descarga en la página de la asignatura.

```
#include <iostream>
#include <cmath>
using namespace std;
class Point2D {
private:
    double px, py;
public:
    Point2D() {
        px = py = 0;
    }
    Point2D(int x, int y) {
        px = x;
        py = y;
    }
    void setX(int px) {
        this->px = px;
    }
    void setY(int py) {
        this->py = py;
    }
    int getX() const {
        return px;
    }
    int getY() const {
        return py;
    }
    void read() {
        cin >> px >> py;
    }
    void print() const {
        cout << "(" << px << ", " << py << ")";
    }
};

class Rectangle {
private:
    Point2D topleft, bottomright; ///< Points that define the rectangle
public:
    Rectangle() {}
    Rectangle(int x, int y, int w, int h) {
        topleft.setX(x);
        topleft.setY(y);
        bottomright.setX(x+w-1);
        bottomright.setY(y+h-1);
        // setGeometry(x,y,w,h);
    }
    void setGeometry(int x, int y, int w, int h) {
        topleft.setX(x);
        topleft.setY(y);
        bottomright.setX(x+w);
        bottomright.setY(y-h);
    }
    void setGeometry(const Point2D &tl, const Point2D &br) {
        topleft = tl;
        bottomright = br;
    }
    Point2D getTopLeft() const {
        return topleft;
    }
    Point2D getBottomRight() const {
        return bottomright;
    }
    bool isEmpty() const {
        return topleft.getX() > bottomright.getX() || topleft.getY() < bottomright.getY();
    }
    void read() {
        topleft.read();
        bottomright.read();
    }
    void print() const {
        cout << "[ ";
        topleft.print();
        cout << " - ";
        bottomright.print();
        cout << " ] ";
    }
    friend Rectangle doOverlap(const Rectangle &r1, const Rectangle &r2);
};

Rectangle doOverlap(const Rectangle &r1, const Rectangle &r2) {
    Rectangle result;
    Point2D rTL, rBR;
    rTL.setX(max(r1.topleft.getX(), r2.topleft.getX()));
    rTL.setY(min(r1.topleft.getY(), r2.topleft.getY()));
    rBR.setX(min(r1.bottomright.getX(), r2.bottomright.getX()));
    rBR.setY(max(r1.bottomright.getY(), r2.bottomright.getY()));
    result.setGeometry(rTL, rBR);
    return result; // Read more
}

bool isInside(const Point2D &p, const Rectangle &r) {
```

```

    return r.getTopLeft().getX() <= p.getX() && p.getX() <= r.getBottomRight().getX() &&
           r.getTopLeft().getY() >= p.getY() && p.getY() >= r.getBottomRight().getY();
}

int main() {
    Rectangle A, B, Intersection;
    Point2D p;
    int count;

    A.setGeometry(2,5,3,3);
    cout << "First rectangle is:";
    A.print();
    cout << endl << "Type second rectangle:";
    B.read();
    cout << endl << "Calculating intersection of:";
    A.print();
    cout << "and:";
    B.print();
    cout << endl;
    Intersection = doOverlap(A,B);
    if (Intersection.isEmpty()) {
        cerr << "Empty intersection" << endl;
    } else {
        cout << "The intersection is:";
        Intersection.print();
        count = 0;
        cout << endl << "Reading points...";
        p.read();
        while (p.getX() >= 0 && p.getY() >= 0) {
            if (isInside(p, Intersection)) {
                p.print();
                count++;
            }
            p.read();
        }
        if (count > 0)
            cout << "fall within the intersection (" << count << " total)" << endl;
        else
            cout << "None of them falls within the intersection" << endl;
    }
    return 0;
}

```

9.1. Fichero de validación v_0inside.test

El cuadro siguiente muestra los datos de un fichero de validación hasta el valor “-1 0” que concluye los datos de entrada. El resto del fichero se proporciona para automatizar los procesos de validación tal y como se indica en la script **doTests.sh** (Sección 8).

```

4 3 8 0
0 0
1 1
-1 0

%%CALL < tests/v_0inside.test
%%OUTPUT
First rectangle is [(2,5) - (5,2)]
Type second rectangle:
Calculating intersection of: [(2,5) - (5,2)] and [(4,3) - (8,0)]
The intersection is: [(4,3) - (5,2)]
Reading points... None of them falls within the intersection

```

10. Apendice 2. Modularización

10.1. Point2D.h

```
class Point2D {
private:
    double px, py;
public:
    Point2D();
    Point2D(int x, int y);
    void setX(int px);
    void setY(int py);
    int getX() const;
    int getY() const;
    void read();
    void print() const;
};
#endif
```

10.2. Rectangle.h

```
#ifndef RECTANGLE.H
#define RECTANGLE.H

#include "Point2D.h"

class Rectangle {
private:
    Point2D topleft, bottomright;
public:
    Rectangle();
    Rectangle(int x, int y, int w, int h);
    void setGeometry(int x, int y, int w, int h);
    void setGeometry(const Point2D &tl, const Point2D &br);
    Point2D getTopLeft() const;
    Point2D getBottomRight() const;
    bool isEmpty() const;
    void read();
    void print() const;
    friend Rectangle doOverlap(const Rectangle &r1, const Rectangle &r2);
};
bool isInside(const Point2D &p, const Rectangle &r);
#endif
```

10.3. Point2D.cpp

```
#include<iostream>
#include "Point2D.h"

using namespace std;

Point2D::Point2D() {
    px = py = 0;
}
Point2D::Point2D(int x, int y) {
    px = x;
    py = y;
}
void Point2D::setX(int px) {
    this->px = px;
}
void Point2D::setY(int py) {
    this->py = py;
}
int Point2D::getX() const {
    return px;
}
int Point2D::getY() const {
    return py;
}
void Point2D::read() {
    cin >> px >> py;
}
void Point2D::print() const {
    cout << "<<px << ", << py<<";
}
```

10.4. Rectangle.cpp

```
#include<iostream>
#include<cmath>
#include "Point2D.h"
#include "Rectangle.h"
using namespace std;

Rectangle::Rectangle() { }

Rectangle::Rectangle(int x, int y, int w, int h) {
    topleft.setX(x);
    topleft.setY(y);
    bottomright.setX(x+w-1);
    bottomright.setY(y+h-1);
    // setGeometry(x,y,w,h);
}

void Rectangle::setGeometry(int x, int y, int w, int h) {
    topleft.setX(x);
    topleft.setY(y);
    bottomright.setX(x+w);
    bottomright.setY(y-h);
}

void Rectangle::setGeometry(const Point2D &tl, const Point2D &br) {
    topleft = tl;
    bottomright = br;
}

Point2D Rectangle::getTopLeft() const {
    return topleft;
}

Point2D Rectangle::getBottomRight() const {
    return bottomright;
}

bool Rectangle::isEmpty() const {
    return topleft.getX()>bottomright.getX() || topleft.getY() < bottomright.getY();
}

void Rectangle::read() {
    topleft.read();
    bottomright.read();
}

void Rectangle::print() const {
    cout << "[";
    topleft.print();
    cout << " - ";
    bottomright.print();
    cout << "]" << endl;
}

Rectangle doOverlap(const Rectangle &r1, const Rectangle &r2) {
    Rectangle result;
    Point2D rTL, rBR;
    rTL.setX(max(r1.topleft.getX(), r2.topleft.getX()));
    rTL.setY(min(r1.topleft.getY(), r2.topleft.getY()));
    rBR.setX(min(r1.bottomright.getX(), r2.bottomright.getX()));
    rBR.setY(max(r1.bottomright.getY(), r2.bottomright.getY()));
    result.setGeometry(rTL, rBR);
    return result; // Read more
}

bool isInside(const Point2D &p, const Rectangle &r) {
    return r.getTopLeft().getX() <= p.getX() && p.getX() <= r.getBottomRight().getX() &&
        r.getTopLeft().getY() <= p.getY() && p.getY() <= r.getBottomRight().getY();
}
```

10.5. main.cpp

```
#include <iostream>
#include <cmath>
#include "Point2D.h"
#include "Rectangle.h"

using namespace std;

int main() {
    Rectangle A, B, Intersection;
    Point2D p;
    int count;

    A.setGeometry(2,5,3,3);
    cout << "First rectangle is ";
    A.print();
    cout << endl << "Type second rectangle: ";
    B.read();
    cout << endl << "Calculating intersection of: ";
    A.print();
    cout << " and ";
    B.print();
    cout << endl;
    Intersection = doOverlap(A,B);
    if (Intersection.isEmpty()) {
        cerr << "Empty intersection" << endl;
    } else {
        cout << "The intersection is: ";
        Intersection.print();
        count = 0;
        cout << endl << "Reading points ... ";
        p.read();
        while (p.getX() >= 0 && p.getY() >= 0) {
            if (isInside(p, Intersection)) {
                p.print();
                count ++;
            }
            p.read();
        }
        if (count > 0)
            cout << " fall within the intersection (" << count << " total)" << endl;
        else
            cout << " None of them falls within the intersection " << endl;
    }

    return 0;
}
```