CALIDAD, PRUEBAS Y MANTENIMIENTO DEL SOFTWARE

Práctica 2: Parte común

Ana Cortés Cercadillo
a.cortesc@edu.uah.es
Carlos Javier Hellín Asensio
carlos.hellin@edu.uah.es
Daniel Ferreiro Rodríguez
daniel.ferreiror@edu.uah.es

Contenido

Software metrics

El código utilizado para realizar los diagramas de flujo es el siguiente:

```
#include<stdio.h>
#include<conio.h>

void main()
{
   int a, b, c, s = 0;
   clrscr();
   printf("Enter a number:\t");
   scanf("%d", &a);
   c = a;

  // the number is reversed inside the while loop.
  while(a > 0)
  {
    b = a%10;
    s = (s*10)+b;
    a = a/10;
  }

  // here the reversed number is compared with the given number.
  if(s == c)
  {
    printf("The number %d is a palindrome", c);
  }
  else
  {
    printf("The number %d is not a palindrome", c);
  }
 getch();
}
```

Para dibujar el diagrama de flujo se ha utilizado la herramienta diagrams [1], dando el siguiente resultado:

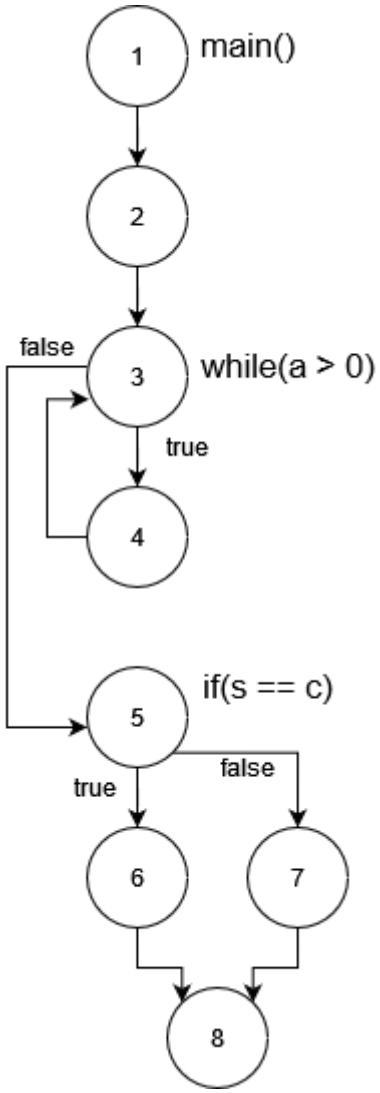


Diagrama de flujo 1 con herramienta de dibujo

Y con code2flow [2], como ya se explicó paso a paso en la anterior práctica, da el siguiente diagrama:

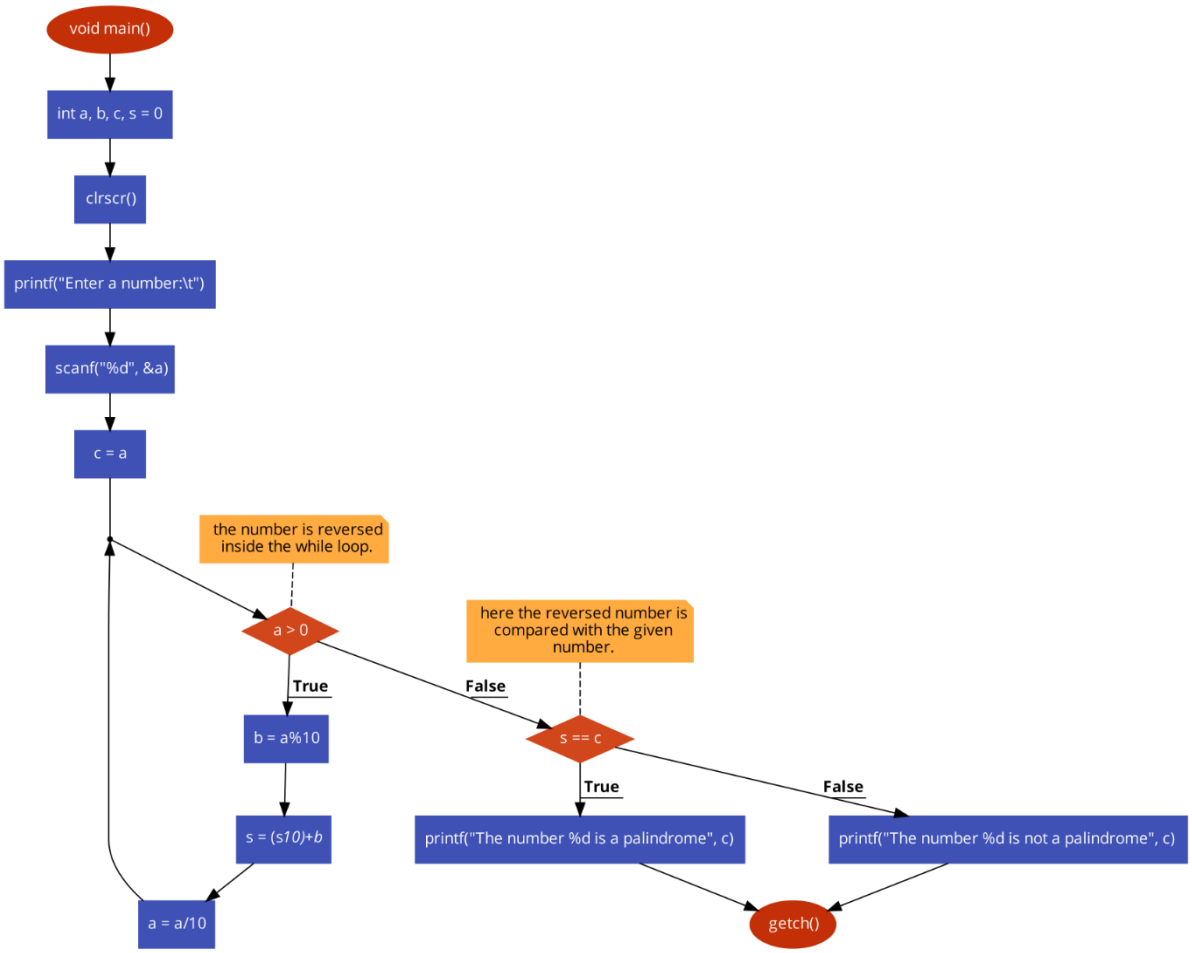


Diagrama de flujo 2 con la herramienta code2flow

Al comparar estos diagramas se puede ver que en el caso del diagrama hecho con la herramienta de dibujo tiene menos nodos y aristas que la generada por code2flow. Esto es debido a que se ha estimado que varias instrucciones seguidas estén en un mismo nodo, mientras que los while e if son separados como lo hace el code2flow. Por lo demás, es prácticamente igual siguiendo el mismo flujo con la repetición del while y el if que se divide en dos: true y false.

Resultado para McCabe

Para calcular McCabe se tiene las siguientes ecuaciones de los apuntes de la asignatura:

- V(G) = nº de regiones cerradas
- V(G) = Aristas – Nodos + 2
- (VG) = Nodos predicado + 1

Se realiza el cálculo para el diagrama de flujo 1:

- V(G) = nº de regiones cerradas = 3
- V(G) = Aristas – Nodos + 2 = 9 – 8 + 2 = 3
- (VG) = Nodos predicado + 1 = 2 + 1 = 3

Se realiza el cálculo para el diagrama de flujo 2:

- V(G) = nº de regiones cerradas = 3
- V(G) = Aristas – Nodos + 2 = 15 – 14 + 2 = 3
- (VG) = Nodos predicado + 1 = 2 + 1 = 3

Al comparar estos resultados, se ve que dan los mismos resultados a pesar de que haya más o menos nodos y aristas como se ha visto en la comparación anterior. Estos resultados reflejan que la complejidad ciclomática del programa es la misma ya que define el número de caminos independientes que hay dentro del código.

Resultado para LOC

Usando los ejercicios en Aula virtual como referencia y la Wikipedia [3], se va a calcular las métricas de tipo LOC del siguiente código:

```
#include<stdio.h>
#include<conio.h>

void main()
{
   int a, b, c, s = 0;
   clrscr();
   printf("Enter a number:\t");
   scanf("%d", &a);
   c = a;

  // the number is reversed inside the while loop.
  while(a > 0)
  {
    b = a%10;
    s = (s*10)+b;
    a = a/10;
  }

  // here the reversed number is compared with the given number.
  if(s == c)
  {
    printf("The number %d is a palindrome", c);
  }
  else
  {
    printf("The number %d is not a palindrome", c);
  }
 getch();
}
```

30 LOC físicas (incluidas líneas en blanco)

- 25 LOC sin blancas ni comentarios
- 25 sentencias excepto comentarios (sentencias que ocupan más de una línea cuentan como una)
- 21 sentencias excepto líneas blancas, comentarios, declaraciones y cabeceras
- 12 sentencias ejecutables excepto condiciones de excepción (y directivas de compilación, etc.)

Resultado de Halstead

Para calcular las métricas de Halstead del siguiente código también se han tenido en cuenta los ejercicios de Aula Virtual, la Wikipedia [4] y la web de ristancase [5] que explican las fórmulas.

```
int sort (int x[ ], int n)

{
  int i, j, save, im1;
  /*This function sorts array x in ascending order */
  If (n< 2) return 1;
  for (i=2; i< =n; i++)
  {
    im1=i-1;
    for (j=1; j< =im1; j++)
      if (x[i] < x[j])
      {
        Save = x[i];
        x[i] = x[j];
        x[j] = save;
      }
  }
  return 0;
}
```

Operandos	Ocurrencias	Operadores	Ocurrencias
int	4	()	5
sort	1	[]	7
x	7	,	4
n	3	{}	3
i	8	;	11
j	7	if	2
save	3	<	2
im1	3	return	2
2	2	for	2
1	3	=	6
0	1	<=	2
		++	2
		-	1
$\mu_1 = 11$	$N_1 = 42$	$\mu_2 = 13$	$N_2 = 49$

SENTENCIAS	10
APARICIÓN DE OPERANDOS	42
APARICIÓN DE OPERADORES	49
OPERANDOS ÚNICOS	11
OPERADORES ÚNICOS	13
LONGITUD	91
LONGITUD DE VOCABULARIO	24
VOLUMEN	417,23
TAMAÑO	86,15
CONTENIDO DE INTELIGENCIA	16,81
NÚMERO ESTIMADO DE ERRORES	0,15
TIEMPO DE PROGRAMACIÓN	575,08
NIVEL DE PROGRAMA	0,04
DIFICULTAD DE PROGRAMA	24,81
ESFUERZO MENTAL	10351,47
NIVEL DE LENGUAJE	0,67

Maintenance

En esta parte se va a tratar de recuperar el modelo de clases correspondiente al siguiente código:

```
public class MyClass1
{
    public int Atribute11;
    public String Attribute12;
    public MyClass3 myClass3;

    public MyClass1(){
        super();
    }

    public boolean operation11(int x) {
        return false;
    }
}

public class MyClass2 extends MyClass1
{
    public int Atribute21;

    public MyClass2(){
        super();
    }

    public void operation21(int x){}
}

public class MyClass3 extends MyClass1
{
    public MyClass3 Atribute31;
    public MyClass1 myClass1;

    public MyClass3(){
        super();
    }

    public void operation31(int x, MyClass2 c2) {
    }
}
```

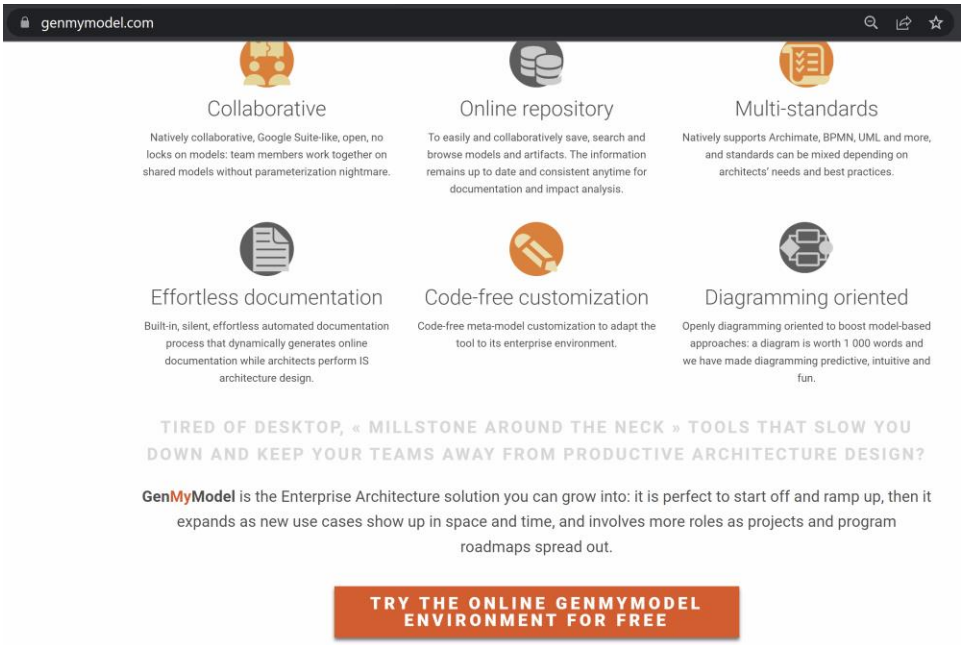
Con la herramienta Genmymodel[1] se dibujarán las clases con todos sus atributos, métodos y relaciones de todo tipo con notación UML correcta. Una vez creado el diagrama completo, se

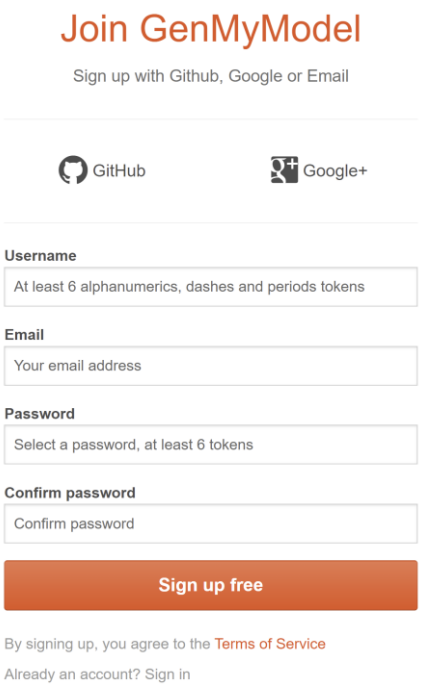
[1] https://www.genmymodel.com/

generará el código Java con la herramienta y se hará una comparación con el de partida detallando diferencias y similitudes. En Genmymodel se incluyen tres generadores de código Java: uml2java, simpleUML2java y advancedUML2Java. A continuación, se realizará un informe detallando todo el proceso con capturas de pantalla.

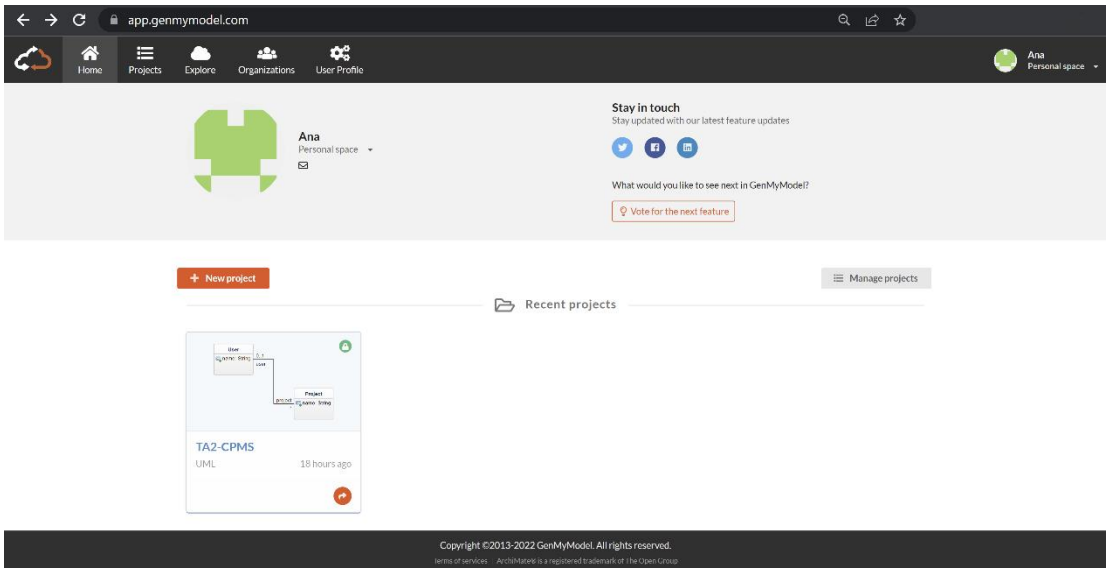
Modelo de clases

Para crear el modelo de clases hay que acceder a la herramienta que se va a usar. En la página web de Genmymodel hay que buscar el botón “Try the online Genmymodel environment for free” y registrarse. En este caso se utilizará la cuenta de GitHub para el registro.

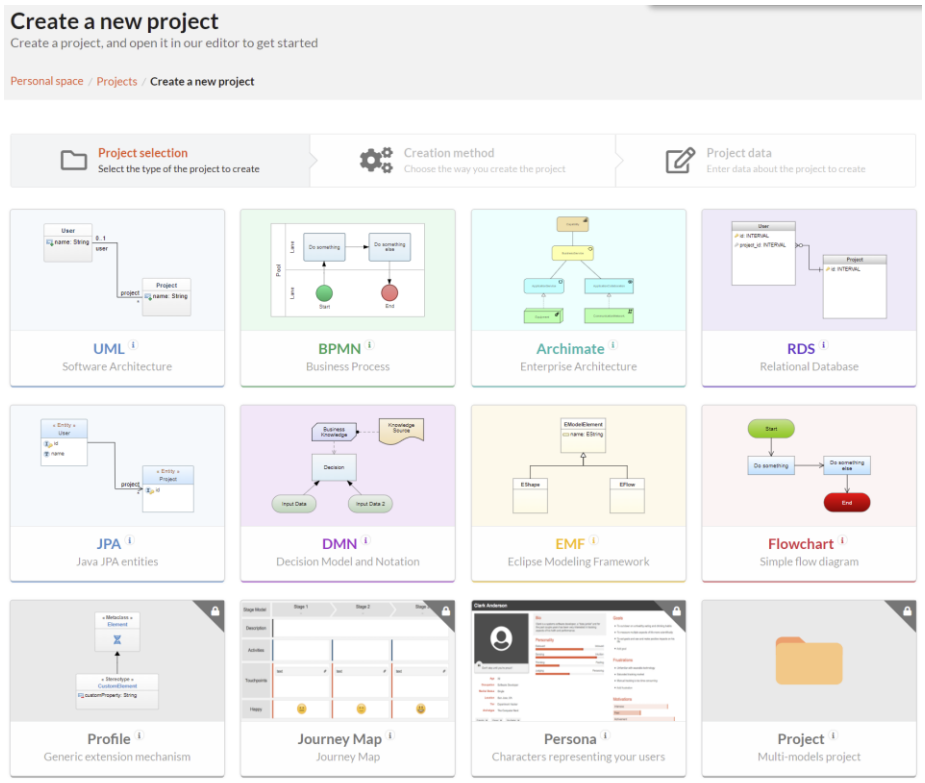




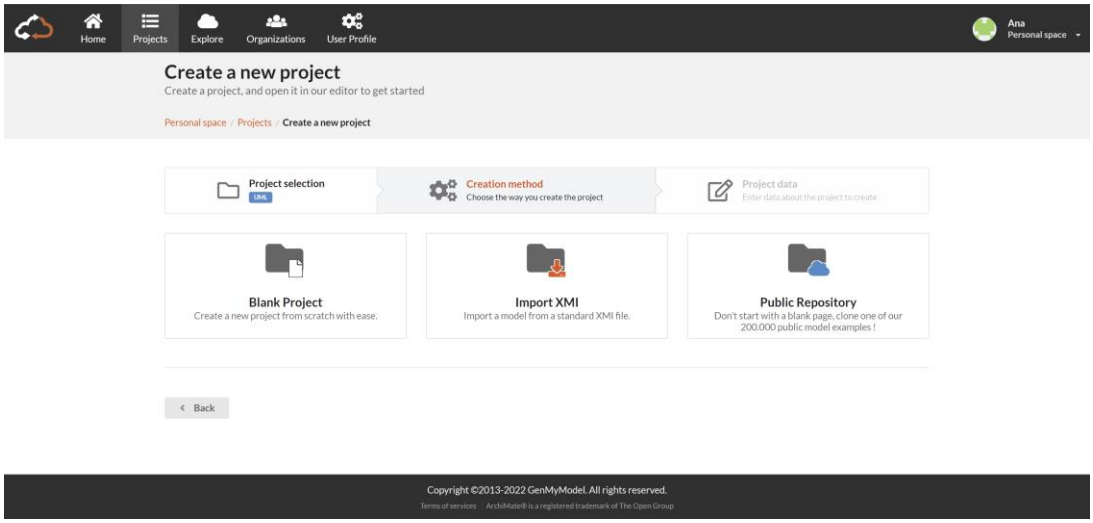
Después de realizar el registro, hay que crear un nuevo proyecto. Para ello encontramos el botón de “New project”.



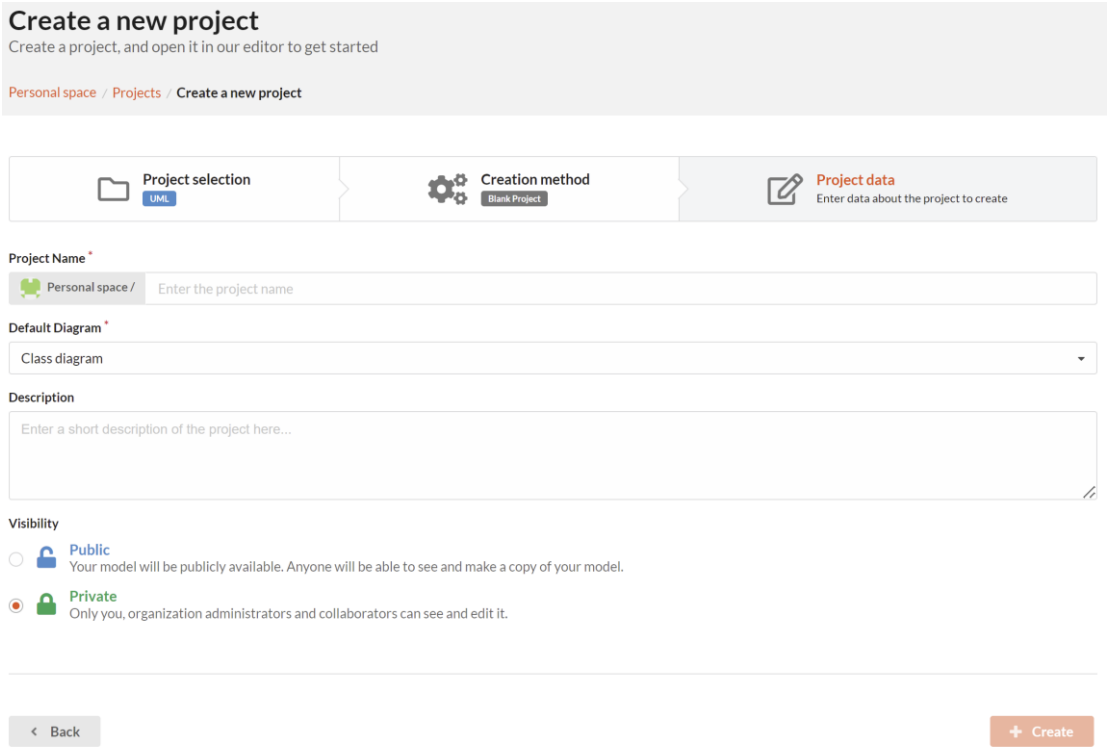
Aparecen varios tipos de diagramas entre los que se pueden elegir. En nuestro caso el diagrama que queremos diseñar es la primera opción, UML Software Architecture.



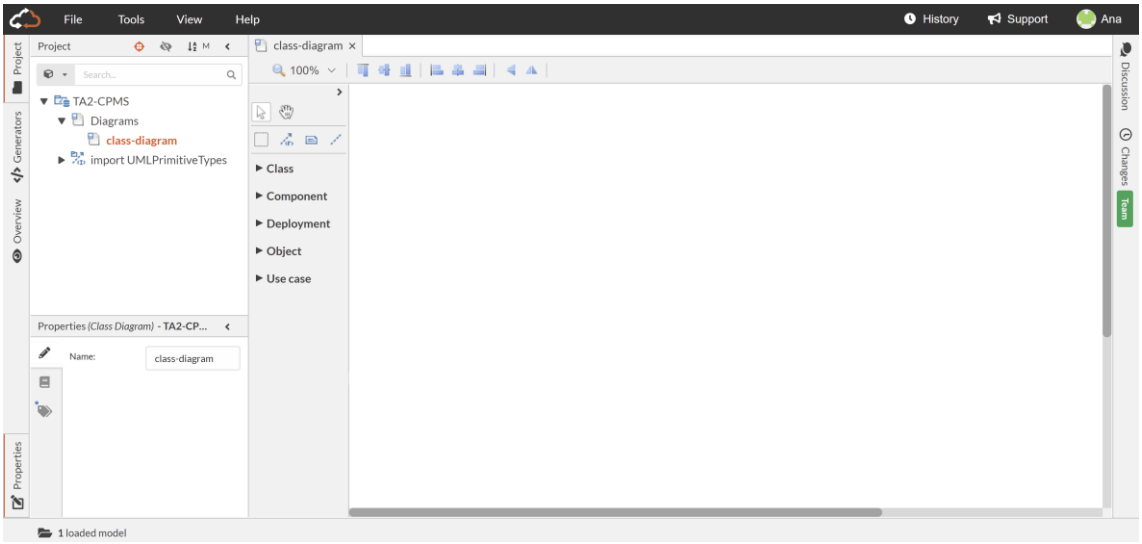
Se pulsa “Blank Project” para empezar un proyecto desde cero.



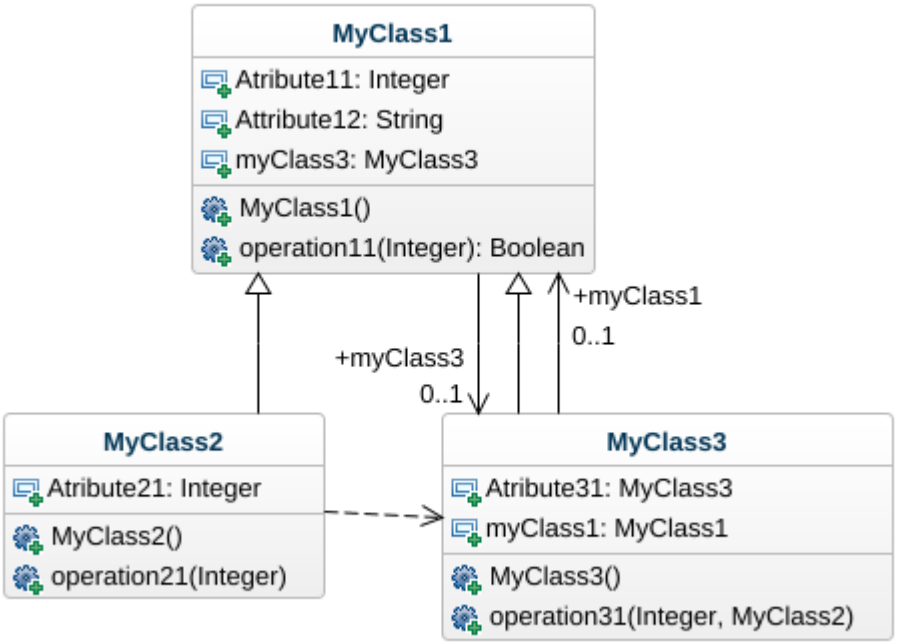
Hay que asignarle un nombre, un tipo de diagrama (el que queremos hacer es el diagrama de clases) y pulsar el botón “+ Create” que se desbloquea en la parte baja de la pantalla.



Finalmente se abre la zona de trabajo donde se puede crear el diagrama de clases.

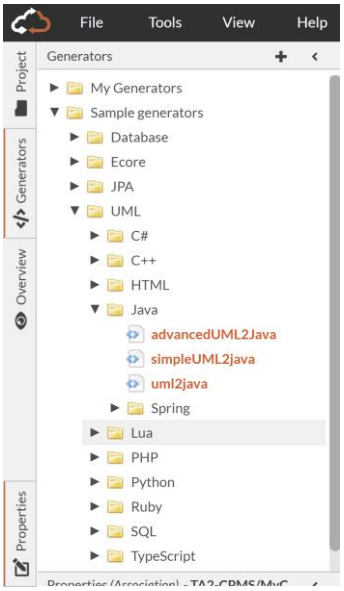


En el diagrama se dibuja las clases con todos sus atributos, métodos y relaciones de todo tipo con notación UML correcta.



Código Java generado

En la herramienta Genmymodel hay tres generadores para código Java: uml2java, simpleUML2java y advancedUML2java. Los códigos que se nombrarán a continuación se encuentran en el documento Model2Cortés-Ferreiro-Hellín.docx en diferentes apartados dentro de la sección “Diagrama de clases 1”. En la siguiente imagen, en la pestaña “Generators”, se muestran estos generadores marcados en naranja.



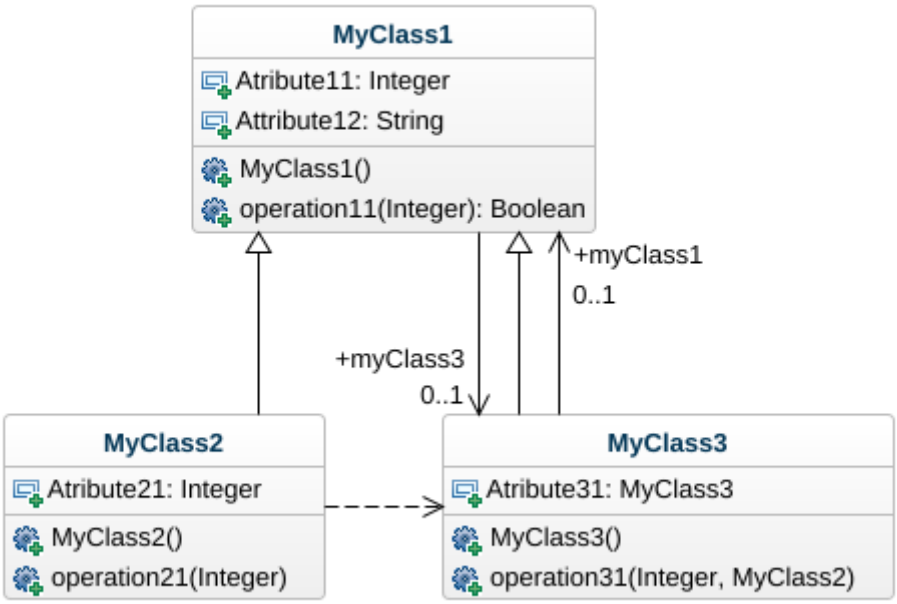
Con el generador de uml2java se crea un código igual al inicial. La gran diferencia entre estos dos es que al generar el código aparecen muchos comentarios. Estos comentarios son una guía para escribir la documentación de las clases, los atributos, el constructor y los métodos además de indicar donde hay que implementar los métodos y los constructores.

Con el segundo generador, simpleUML2java, también se crean los comentarios. Este generador añade los métodos get y set de cada atributo de la clase automáticamente. Tiene un error importante al crear los métodos set. Les añade el mismo tipo de retorno que el parámetro de entrada cuando este tipo de métodos no tienen retorno (void). Otra diferencia con el generador anterior es que ha creado automáticamente el atributo asociado a la relación entre las clases MyClass1 y MyClass3, por lo tanto, aparecen repetidos, el generado automáticamente y el que ya está en el diagrama como atributo. También han desaparecido el método operation21() de la clase MyClass2 y el método operation31() de la clase MyClass3.

El último generador, advancedUML2java, crea un código y unos comentarios iguales a los del anterior. La única diferencia es que, además de los get y los set de los atributos de las clases, se crean también de los creados automáticamente. A esto se debe que aparezcan duplicados estos métodos.

Modelo de clases quitando atributos repetidos

Para evitar atributos y métodos duplicados se opta por eliminar del diagrama anterior los atributos relacionados con otras clases y dejar las relaciones que luego los crearán automáticamente en el código. El nuevo diagrama quedaría como se muestra en la imagen.



Código Java generado

Los códigos generados se encuentran en el documento Model2Cortés-Ferreiro-Hellín.docx en los apartados dentro de la sección “Diagrama de clases 2”.

Con el generador de uml2java ahora no aparecen los atributos borrados myClass3 que estaría en MyClass1 y myClass1 de la clase MyClass3.

Con el segundo generador, simpleUML2java, ya se crean los atributos mencionados antes automáticamente evitando las duplicidades. El método operation21() de la clase MyClass2 y el método operation31() de la clase MyClass3 siguen sin aparecer con este generador.

El último generador, advancedUML2java, también crea automáticamente los atributos de la asociación y además sus métodos get y set. Tampoco aparecen los métodos mencionados anteriormente.

Conclusiones

La herramienta Genmymodel da opciones para generar una plantilla de código en varios lenguajes. En este caso se ha realizado sobre el lenguaje de programación Java. Se disponen de tres generadores que se han usado y estudiado en esta sección.

Después de haber probado todos ellos, el código más cercano al inicial es el generado por uml2java del primer diagrama, donde se encuentran todos los atributos iniciales, sin duplicados, y todos los métodos incluso con la implementación correcta (nótese en el método operation11() de la clase MyClass1).

Bibliografía

[1] Diagrams. https://app.diagrams.net [Último acceso 15/mayo/2022]

[2] code2flow https://app.code2flow.com [Último acceso 15/mayo/2022]

[3] Source lines of code https://en.wikipedia.org/wiki/Source_lines_of_code [Último acceso 15/mayo/2022]

[4] Halstead complexity measures https://en.wikipedia.org/wiki/Halstead_complexity_measures [Último acceso 15/mayo/2022]

[5] Classification of Source Code Metrics https://www.ristancase.com/html/dac/manual/2.12.01-Software-Metrics-Classification.html [Último acceso 15/mayo/2022]