Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Grado en ingeniería informática

Algorítmica interactiva

Memoria

Autor: Michelle Cabrera Carranza

Tutor: Enric Sesa Nogueras

Otoño 2017

TecnoCampus
Mataró-Maresme

Resum

Es desenvoluparà un software que permeti demostrar de manera interactiva l'execució de codi java, introduït per l'usuari mitjançant un element extern, per ajudar a comprendre que succeix quan s'executa un linea de codi. S'utilitzará l'API de Java anomenada *reflection,* per a conèixer els valors de les variables en temps d'execució, i s'utilitzarà l'eina de desenvolupament anomenada Processing per a poder crear les animacions.

Resumen

Se desarrollará un software que permita mostrar de manera interactiva la ejecución de código java, introducido por el usuario mediante un elemento externo, para ayudar a comprender que está pasando cuando se ejecuta una línea de código. Se utilizará la API de Java llamada *reflection,* para conocer los valores de las variables en tiempo de ejecución, y la herramienta de desarrollo llamada Processing para poder crear las animaciones.

Abstract

It will be implemented a software that allows an interactive display of the execution of Java code, introduced by the user through an external element, to help understand what is happening when a code line is executed. It will be used Java Reflection API to know the variables values at runtime and the development tool called Processing to be able to create animations.

Índice

Índice de figuras

Índice de tablas

Glosario

Algoritmo: es una secuencia finita de pasos lógicos que permiten solucionar un problema.

Compilador: programa que traduce los lenguajes de programación a un código intermedio (*byteCode*) o a lenguaje máquina.

JRE (J*ava Runtime Environment*): es un conjunto de utilidades que permite la ejecución de programas Java. El entorno en tiempo de ejecución de Java está conformado por una Máquina Virtual de Java o JVM, un conjunto de bibliotecas Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada. El JRE actúa como un "intermediario" entre el sistema operativo y Java. [1]

JDK: es un software que provee herramientas de desarrollo para la creación de programas en Java. (En cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador). [2]

Java Virtual Machine (JVM): La máquina virtual de Java es el entorno en el que se interpreta y ejecutan programas escritos en java. Es capaz de entender tanto el *bytecode* como el sistema sobre el que se pretende ejecutar.

Token: Cadena de caracteres que sirve para delimitar o identificar partes del código

1 INTRODUCCIÓN

1.1 Contexto

En los últimos años gracias a los avances tecnológicos ha habido un aumento de demanda de profesionales técnicos. Y en particular de programadores, desarrolladores de software, analistas, etc.

Se puede decir que vivimos en un mundo gobernado por la tecnología dado que es necesaria para el día a día de las personas, ya que se utiliza en el hogar, el trabajo y en el tiempo libre.

La tecnología aporta grandes beneficios al ser humano, gracias a ella se mejora la calidad de vida de los individuos. Ayuda a progresar más rápido y mejor a campos como los de la medicina, empresariales, comunicación, educación, etc.

Actualmente los estudios técnicos han adquirido mucha relevancia en la sociedad. Por lo tanto, una de las habilidades con alta demanda en el mercado laboral es saber programar.

Actualmente existen muchas empresas y gobiernos que están haciendo un gran esfuerzo para fomentar las carreras técnicas a los futuros alumnos. Para ello llevan a cabo talleres, charlas, publican vídeos atractivos y animados, etc. Una de las habilidades que se promueven es la de la programación, ya que estudios afirman que ayuda a la resolución de problemas, mejora la capacidad de atención, y es algo que se puede aplicar en clases.

Muchos expertos creen que el aprendizaje de la programación debería iniciarse desde edades tempranas y no como en la mayoría de casos cuando se termina la educación obligatoria y/o bachillerato (si decides continuar estudiando informática).

Al comenzar en el mundo de la programación tarde o temprano los estudiantes se enfrentan a la lectura de líneas de código. Aunque este es un paso posterior al aprendizaje de los conceptos algorítmicos, es bastante importante y se domina a base de práctica.

1.2 Identificación de la necesidad/problema a resolver

En el ámbito de la algorítmica y la programación existen conceptos que bien y ser plasmados de manera estática, responden a procesos de naturaleza dinámica. Las iteraciones son un buen ejemplo.

También existen conceptos de naturaleza abstracta, por decirlo de alguna manera, que son mejor comprendidos mostrándolos de alguna manera gráficamente. En este terreno, las metáforas gráficas ayudan a los estudiantes a asimilar estos conceptos y razonar sobre ellos. Métodos y procedimientos en general, son un buen ejemplo. [3]

Para resumir, la idea del párrafo anterior es que el aprendizaje y la compresión de nuevos conceptos son más efectivos si se realiza de manera visual e interactiva.

1.3 Idea principal

Lo que se desea es ayudar a la compresión de la programación de una manera visual e interactiva, ya que cuando las personas inician en este mundo la experiencia suele ser un poco difícil ya sea porque no se acaban de comprender las estructuras de datos o porque los estudiantes se suelen perder al mirar y reseguir el código escrito.

Así que la idea principal es la creación de una aplicación que ayuda al aprendizaje de los conceptos algorítmicos mediante animación interactiva. La aplicación se centra en la lectura de código, es decir, ayudará a reseguir el código mostrando que está pasando en la ejecución de cada sentencia. Por otro lado, está la parte de funcionamiento de estructuras de datos, esta mostrará de manera visual las acciones que se pueden realizar con las estructuras como las colas, pilas y listas utilizando sus respectivos métodos en java.

Al hablar cliente y proyectista se definieron características de la aplicación que se explica con detalle en el apartado análisis del proyecto. A continuación, se explica una breve definición de la aplicación.

La idea inicial es que el usuario pueda introducir código mediante un fichero externo y a partir de este se cree la animación, la cual resiga/espíe lo que está pasando en cada momento. Por otro lado, se desea que la aplicación pueda mostrar el funcionamiento de las estructuras de datos.

El cliente deja en manos del proyectista la elección de las tecnologías.

2 ANÁLISIS DE REFERENTES

En la actualidad hay varias webs y herramientas que tratan de impartir formación de una manera visual e interactiva para la rápida comprensión del alumno. A continuación, se describen algunas de estas aplicaciones web.

2.1 Aplicaciones webs analizadas

2.1.1 Penjee

Aplicación web, pensada para niños, que enseña de forma visual e interactiva a programar en Python. Ofrece un panel de control para mirar los progresos de cada usuario y el apartado de aprendizaje, que es en donde se encuentran las actividades que los usuarios deben realizar para poder avanzar en sus progresos.

Al realizar actividades la pantalla se divide en 3 zonas. La primera contiene una pequeña explicación sobre lo que se debe realizar. En la segunda zona se encuentra el editor de texto en donde el usuario puede escribir el código y la tercera zona tiene la animación hecha a partir del código introducido por el usuario. [4]

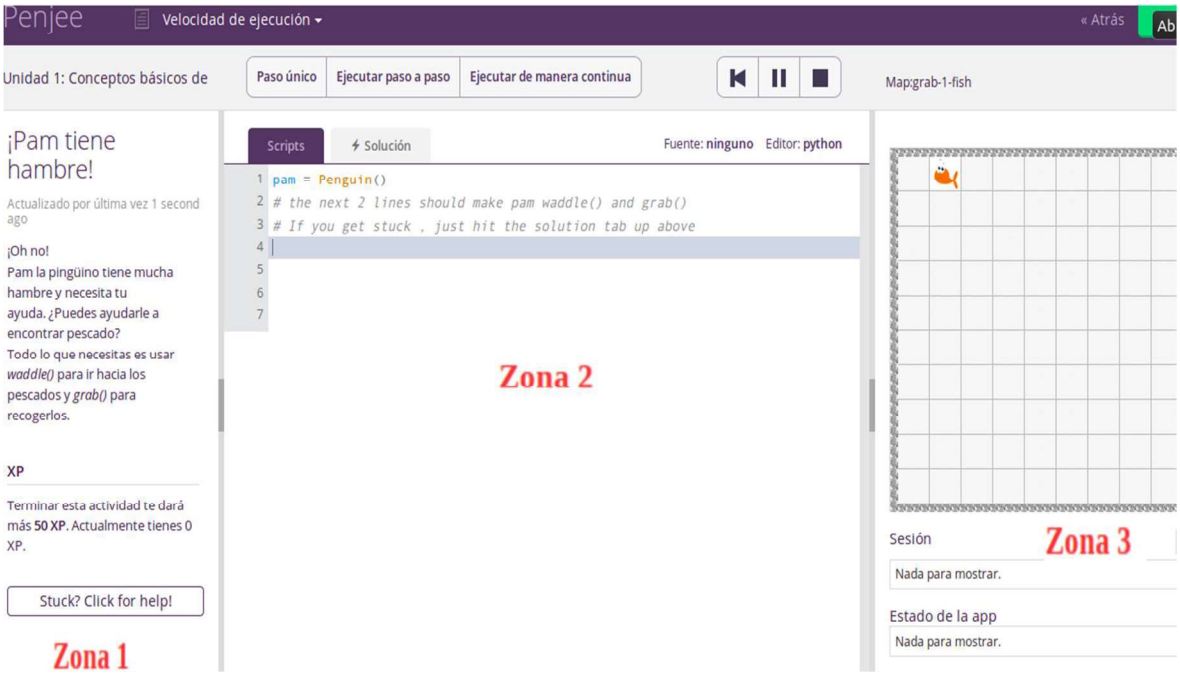


Imagen 2.1 Apartado de aprendizaje en donde se realizan las actividades

La zona del editor posee unos botones para controlar la ejecución esto permite saber que está pasando por cada línea de código ejecutada.

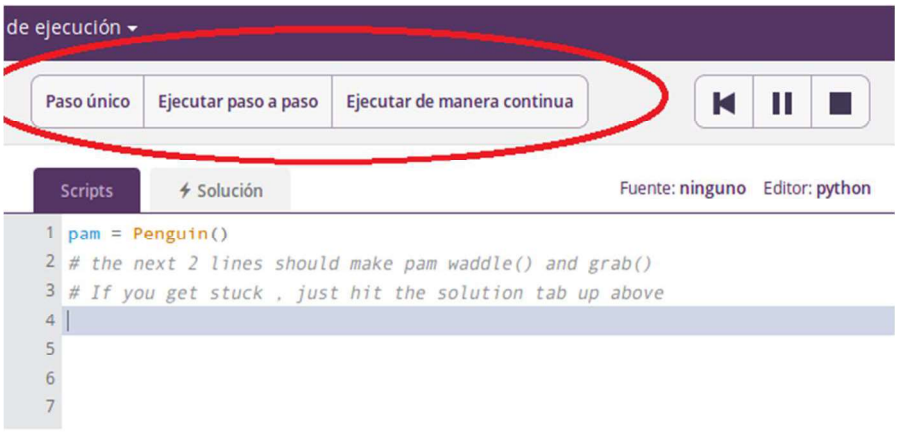


Imagen 2.2 Botones para ir paso por paso a través de la ejecución

2.1.2 Codecademy

Aplicación web que ofrece cursos gratuitos para aprender a programar. El funcionamiento es sencillo hay una lista de cursos en las que el usuario puede inscribirse. Una vez inscrito el usuario puede acceder a las actividades de dicho curso. El apartado de actividades cuenta con tres zonas, una de ellas detalla la actividad que se debe de realizar, la segunda zona cuenta con un editor para que el usuario introduzca el código y por último está la zona de reproducción en la cual está la consola o una emulación del *browser*, dependiendo las necesidades del curso. En la web el usuario también puede acceder al *dashboard* el cual tiene el progreso de los cursos inscritos. [5]

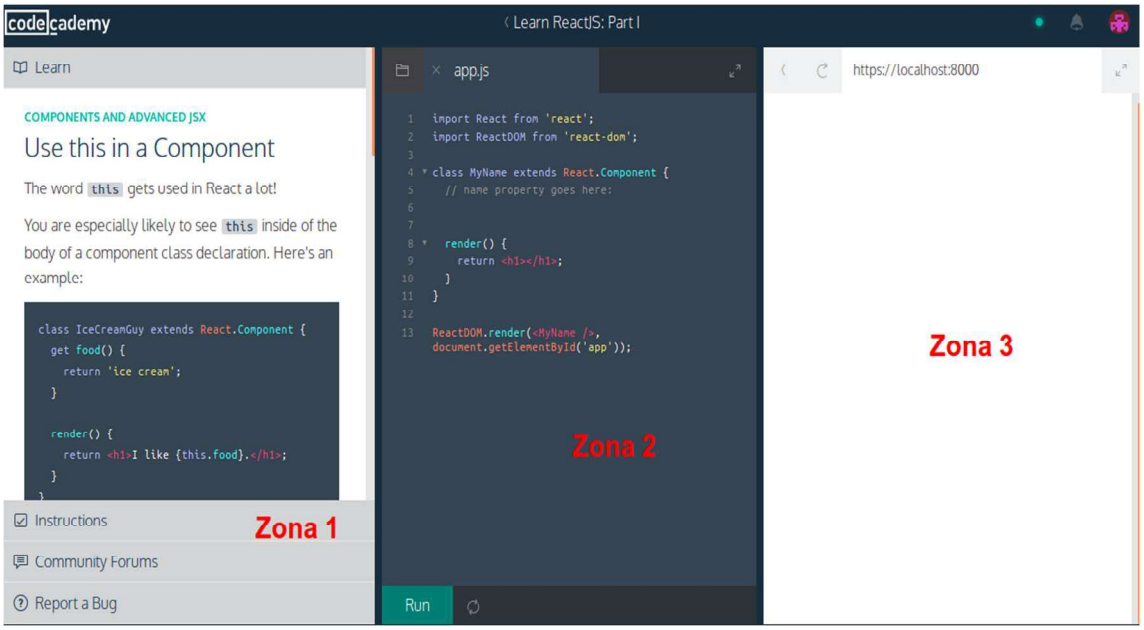


Imagen 2.3 Zonas de actividades Codecademy con emulador de browser

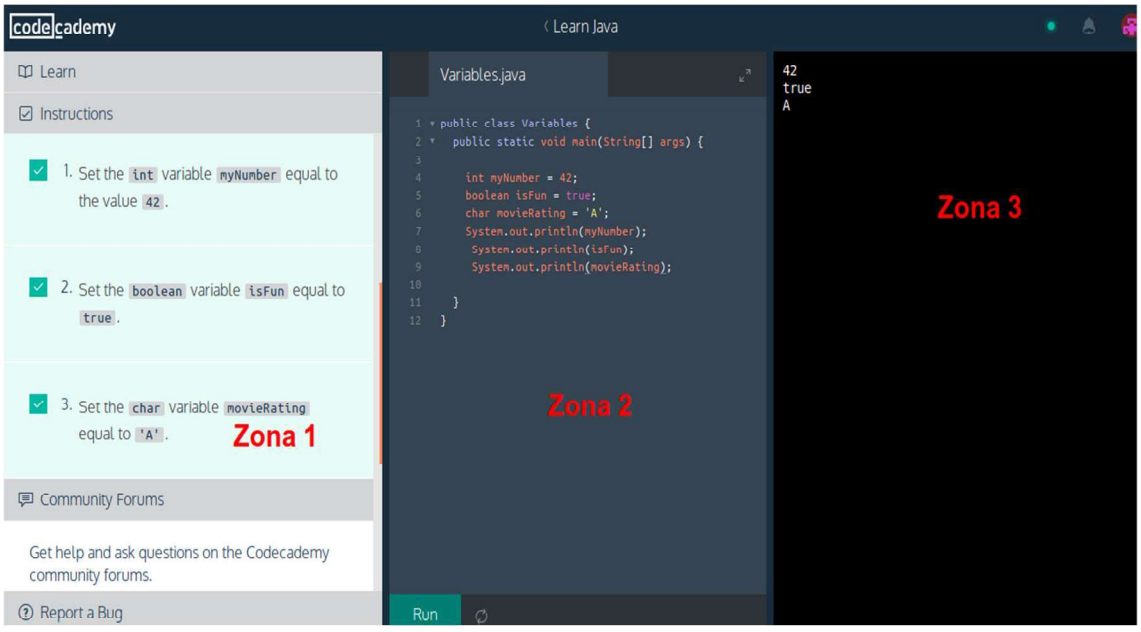


Imagen 2.4 Zonas de actividades Codecademy con emulador de consola

La zona 2 tiene un botón *run* para ejecutar el código que el usuario introduce en el editor y así poder ver los cambios efectuados.

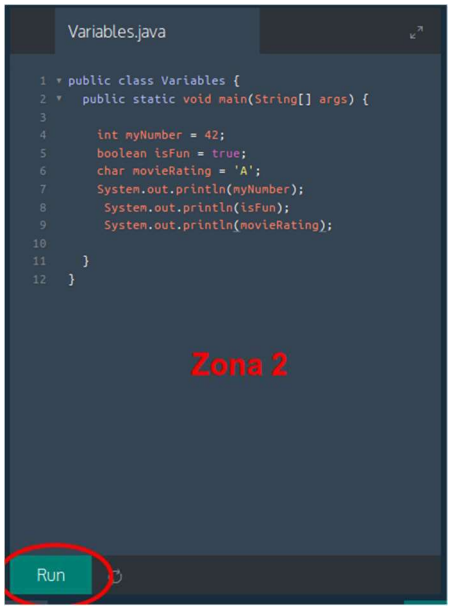


Imagen 2.5 Botón run para ejecutar el código escrito en el editor

2.1.3 W3school

Sitio web que posee tutoriales de tecnologías web como html, css. Javascript, php, etc. Tiene el apartado *Try it yourself* que se divide en dos zonas una de ellas tiene el editor de texto y por otro lado tiene el emulador del *browser.* [6]

En la zona del editor también cuenta con un botón run que ejecuta el código introducido por el usuario.

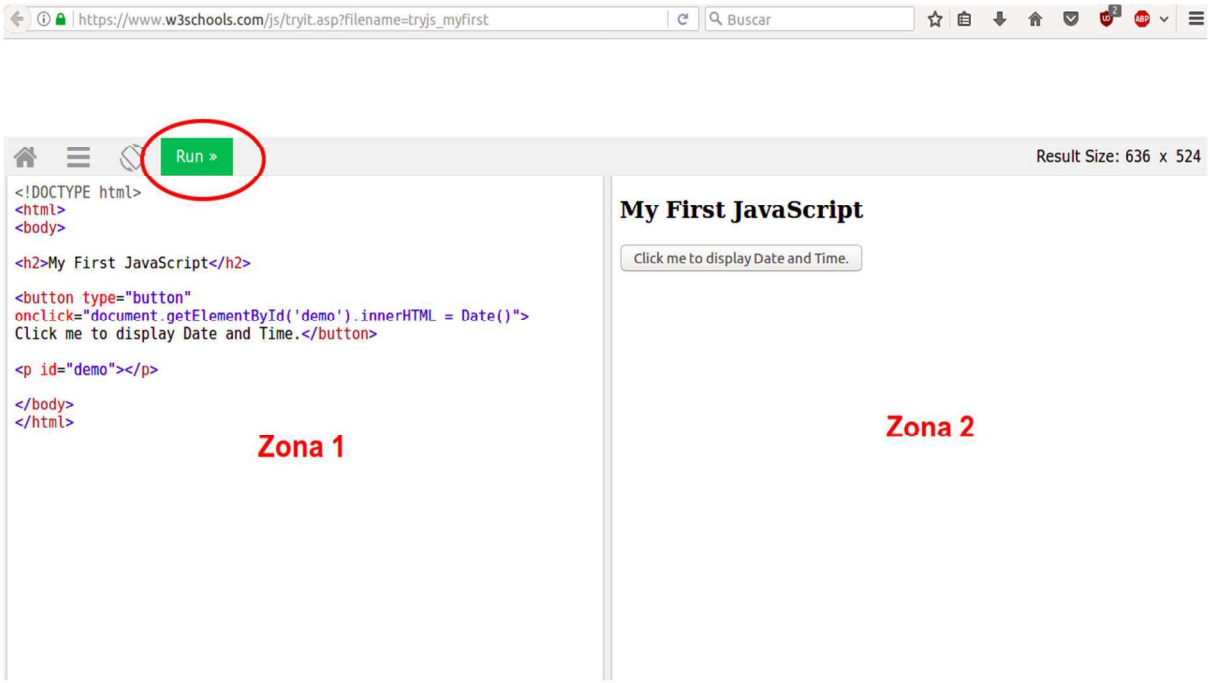


Imagen 2.6 Apartado de *Try it yourself* de w3school

2.1.4 Conclusiones del de análisis referentes

A continuación, se analiza el estado actual del mercado y se extrae las similitudes y diferencias de las aplicaciones web que han sido analizadas.

Se ha podido observar que todas las aplicaciones tienen el objetivo de facilitar la comprensión de la programación, para ello cuentan con un apartado para poder realizar actividades.

En el apartado de aprendizaje se pueden diferenciar 3 zonas (en algún caso solo hay 2 zonas), una de las zonas suele tener la explicación de la actividad a realizar, otra de las zonas tiene un elemento para introducir el código y finalmente está la zona de animación en donde se encuentra la consecuencia del código introducido, en algunos casos la consola en otros una animación para poder ver lo realizado

También se cuenta con botones para poder controlar la ejecución del código. En la mayoría de los casos hay un botón *run* el cual ejecuta todo el código introducido por el usuario. En el caso de Penjee existe la posibilidad de avanzar a través del código paso a paso.

3 HERRAMIENTAS ESCOGIDAS

3.1 Herramientas de desarrollo

Processing Development Environment (PDE)

Para crear las animaciones se utilizará el lenguaje y entorno de desarrollo de Processing [7] además de Java. A continuación, se hace una breve explicación de las funcionalidades y partes que tiene su entorno.

Processing es un software y lenguaje de código abierto basado en Java con una gran comunidad de usuarios. Posee numerosas librerías y herramientas que ayudan a crear animaciones de manera fácil y rápida.

Las aplicaciones/programas en el ámbito de Processing se llaman sketches los cuales permiten dibujar animaciones (ya sea de dos o tres dimensiones).

Es posible programar con Processing en diferentes lenguajes de programación dependiendo de las necesidades en particular de cada usuario. Para ello se debe de añadir el modo en el que se quiere programar, esta opción se modifica desde el propio editor.

Por otro lado, si se habla del funcionamiento referente a la ejecución de código este se debe escribir en el editor de texto y proceder con la ejecución.

Una característica adicional es que Processing posee su propia sintaxis que facilita y agiliza la escritura de código, por ejemplo, si se desea escribir por consola alguna información con java se escribiría la sentencia *System.out.println("Imprimir por pantalla")*, en cambio con la sintaxis de Processing seria *println(Imprimir por pantalla");* la cual tendría el mismo comportamiento que el primero. Las dos formas de escribir son correctas para Processing y esto es debido a que antes de compilar el código hay un pretratamiento.

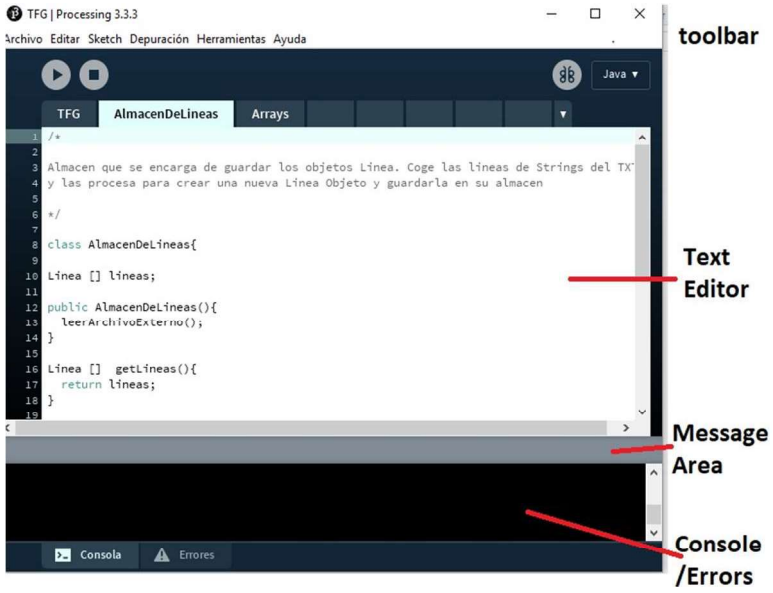


Imagen 3.1 Entorno desarrollo de Processing

Al escribir código en el editor este se guarda en un archivo con extensión.pde que posteriormente, en el momento de la ejecución, será tratado y Processing lo convertirá en un archivo .java y finalmente se procede a la compilación creando su respectivo archivo .class para que la *Java Virtual Machine* lo pueda ejecutar.

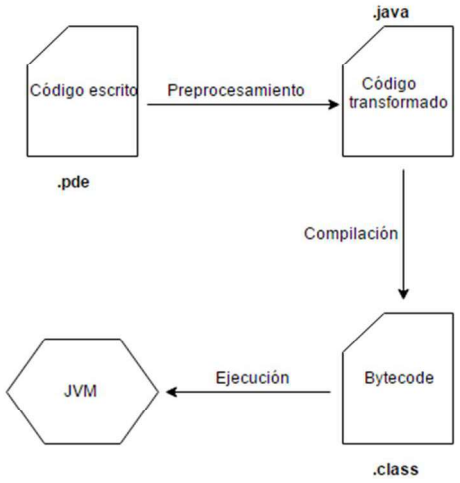


Imagen 3.2 Flujo compilación de Processing

3.2 Reflection

Uno de los requisitos del proyecto es saber que está pasando en cada momento y que valores tienen las variables al avanzar la ejecución del código. Para poder conocer los valores de las variables se utilizará *reflection*.

Se le llama *reflection* a la capacidad de que un programa examine y modifique su estructura y comportamiento en tiempo de ejecución

Con *reflection* se puede explorar las clases en busca de sus características para saber el tipo de cada atributo, su valor, sus modificadores de acceso, el número de parámetros de las funciones y de que tipo es su retorno.

Como usar *reflection* en Java

Gracias a la API de Java se puede examinar:

- java.lang.Class [Se puede obtener el nombre de la clase y el nombre de la superclase]
- java.lang.reflect.Field [Se puede obtener los nombre de los atributos, de que tipo son, sus modificadores de acceso y su valor en tiempo de ejecución]
- java.lang.reflect.Methods [Se puede obtener el número y los tipos de parámetros, el tipo de valor retornado. También se puede invocar a dichos métodos]
- java.lang.reflect.Constructor [Se puede obtener el número de parámetros y sus respectivos tipos. Se pueden crear objetos a partir de dichos constructores en tiempo de ejecución]

A continuación, se ilustrará, con un pequeño ejemplo, el funcionamiento de *reflection*.

Se crea una clase llamada ClassIWantExaminate la cual se quiere examinar.

```java
package Examples;

public class ClassIWantExaminate {

// Attributes
  public int id;
  private String name;

 // Contructors
  ClassIWantExaminate(){
    this.setId(0);
    this.setName("");
  }

   ClassIWantExaminate(int id, String name){
     this.setId(id);
     this.setName(name);
   }

 // getters
    public int getId(){
      return this.id;
    }

    public String getName() {
      return name;
    }

 // setters
     public void setId(int id){
       this.id = id;
    }

    public void setName(String name) {
      this.name = name;
    }
}
```

Imagen 3.3 Clase a examinar - ClassIWantExaminate

Para iniciar el proceso de inspección es necesario conocer el java.lang.Class de la clase a examinar. Hay diversas formas de conseguir un class de una clase especifica.

```
Class classIWantExaminate = ClassIWantExaminate.class;
```

O también:

```
Class classIWantExaminate = Class.forName("ClassIWantExaminate");
```

Una vez se ha conseguido el objeto class se puede comenzar a analizar la clase.

Para conseguir la lista de constructores es necesario utilizar el método getConstructors(). Y se puede obtener los parámetros de cada constructor utilizando el método getParameterTypes():

```
Constructor[] constructors = classIWantExaminate.getConstructors();
 for(Constructor c : constructors){
    for(Class param :c.getParameterTypes()){
        System.out.println( param);
    }
 }
```

Imagen 3.4 Cómo conseguir el constructor de una clase

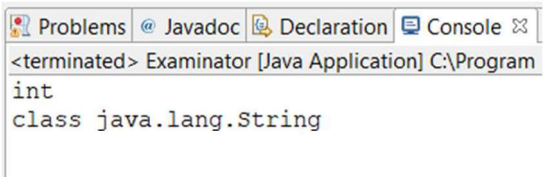


Imagen 3.5 Parámetros del constructor de la clase examinada

Con el constructor y los parámetros se puede instanciar objetos si fuera necesario.

Por otro lado, para la examinación de métodos se utiliza el método getDeclaredMethods:

```
Method[] method = classIWantExaminate.getDeclaredMethods();
```

Una vez obtenidos se puede proseguir a su inspección (Conocer el tipo de retorno, los parámetros de entrada) y también se tiene la posibilidad de su invocación en tiempo de ejecución

```
ClassIWantExaminate objectClassIWantExaminate = new ClassIWantExaminate(1,"Cabrera");
Method method = classIWantExaminate.getDeclaredMethod("getId", null);
Object value =  method.invoke(objectClassIWantExaminate);
System.out.println("value: "+value);
```

Imagen 3.6 Cómo invocar métodos de la clase examinada

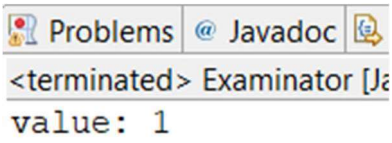


Imagen 3.7 Valor retornado del método invocado

Para examinar los atributos de la clase se debe de obtener los *Fields*.

Una vez obtenidos se puede mirar el nombre de los atributos y de que tipo son.

```
Field[] fields = classIWantExaminate.getDeclaredFields();
for(Field field : fields){
  System.out.println(field);
}
```

Imagen 3.8 Cómo conseguir información sobre los atributos de la clase examinada

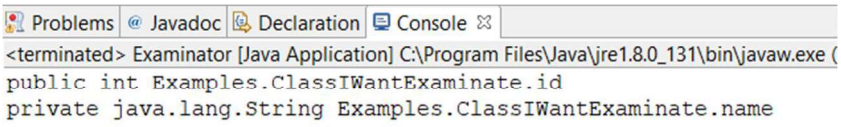


Imagen 3.9 Información sobre los atributos de la clase examinada

Obtener el valor de los atributos:

A continuación, se detallan los pasos a seguir para conocer los valores de las variables en tiempo de ejecución.

1 – Creación de una clase propia, con atributos y métodos. En este caso la clase llamada ClassIWantExaminate

2 - Un objeto Class es construido automáticamente por la máquina virtual de Java al cargarse las clases. Este objeto contiene los metadatos de la clase, información sobre sus atributos y métodos.

3 - Instanciar un objeto de la clase que se ha creado. En este caso creación de un objeto ClassIWantExaminate

4 – Obtener el class de la clase ClassIWantExaminate (Obtención anteriormente descrita).

5- Y finalmente para saber los valores de las variables del objeto instanciado:

- Es necesario obtener las Fields del objeto Class para ello se utiliza el método getDeclaredFields.
- Posteriormente se utiliza el método get y se pasa por parámetro el objeto instanciado. Así obtenemos el valor de dicho atributo

```
ClassIWantExaminate objectClassIWantExaminate = new ClassIWantExaminate(1,"Cabrera");
Field[] fields = classIWantExaminate.getDeclaredFields();
for(Field field : fields){
  field.setAccessible(true);
  Object value = field.get(objectClassIWantExaminate);
  System.out.println(value);
}
```

Imagen 3.10 Cómo obtener el valor de los atributos de la clase examinada

Imagen 3.11 Valores de los atributos de la clase examinada

En la clase a examinar hay atributos no públicos, para tener acceso a ellos se debe de utilizar el método *setAccessible()*. En caso de no utilizar el método al querer acceder desde otra clase se lanzaría una excepción.

4 ANÁLISIS

4.1 Definición de la aplicación

El objetivo de la aplicación es que pueda ser de ayuda a la hora de explicar la algorítmica de una manera visual e interactiva ya que, a veces, resulta complicado para personas que nunca han programado ver lo que está pasando en cada momento en la ejecución de cada línea de código.

Para resolver esta problemática, o mejor dicho para ayudar a la compresión de estos nuevos conceptos se desarrollará está aplicación.

A continuación, se explicará las características de la aplicación establecidas por el tutor de proyecto que en este caso tiene el rol de cliente.

La aplicación tiene como objetivo saber que está pasando en cada momento con las líneas de código anteriormente introducidas en un elemento externo, en nuestro caso un elemento TXT, llamado allCodeToExaminate.txt o en su defecto en el editor al entrar en la aplicación.

La aplicación estará programada en Processing, que es una herramienta para poder crear animaciones.

Requisito principal es saber el valor de cada variable en cada una de las diferentes etapas de la ejecución.

4.2 Requisitos funcionales

A continuación, se define el funcionamiento que debería tener el software demandado.

4.2.1 Menú principal

Al abrir la aplicación se dispondrá de 3 apartados. Los cuáles serán accesibles mediante el clic del botón del mouse.

Los apartados serán:

Code Spy: parte del programa el cual leerá el contenido del fichero txt y posteriormente lo analizará y lo utilizará para mostrar el valor de cada variable en los diferentes momentos en la ejecución.

Estructura de datos: parte del programa el cual mostrará, de una manera visual, el funcionamiento de las estructuras de datos (colas, pilas, listas)

Tutorial: parte del programa que describirá el funcionamiento del software, es decir, su documentación.

4.2.2 Pantalla *Code Spy*

La pantalla *CodeSpy* se encargará de leer, analizar y procesar las sentencias de las líneas introducidas en el editor. Esta parte de la aplicación construye dos escenarios, uno de ellos correspondiente a la visualización del código y el otro a la visualización de los valores de las variables.

En caso de que en el editor hubiera un error causado en tiempo de compilación el sistema muestra un mensaje de error para que el usuario pueda corregir el código.

Por otro lado, existen también los errores en tiempo de ejecución en cuyo caso se ocasionaría si se produce algún error al ir avanzando en la ejecución, El sistema también alertaría al usuario de que ha habido una excepción de tipo *Runtime*.

4.2.3 Pantalla Estructura de datos

Pantalla encargada de mostrar de forma interactiva el funcionamiento de las estructuras de datos.

Para cada estructura especifica hay una animación y en ella se puede hacer una acción correspondiente a los métodos comunes de dicha estructura.

4.2.4 Pantalla Tutorial

Pantalla encargada de documentar/informar al usuario de las características del software en la cual hay una breve explicación de cada pantalla.

4.2.5 Settings

Esta funcionalidad se refiere a las características del entorno ya que se permitirá a través de un elemento externo cambiar el color de fondo, las letras, el tamaño de la pantalla. Todo esto para que el usuario tenga la capacidad de decidir que entorno le es más agradable. Así se garantiza que el entorno se adapta a sus necesidades.

4.3 Casos de uso

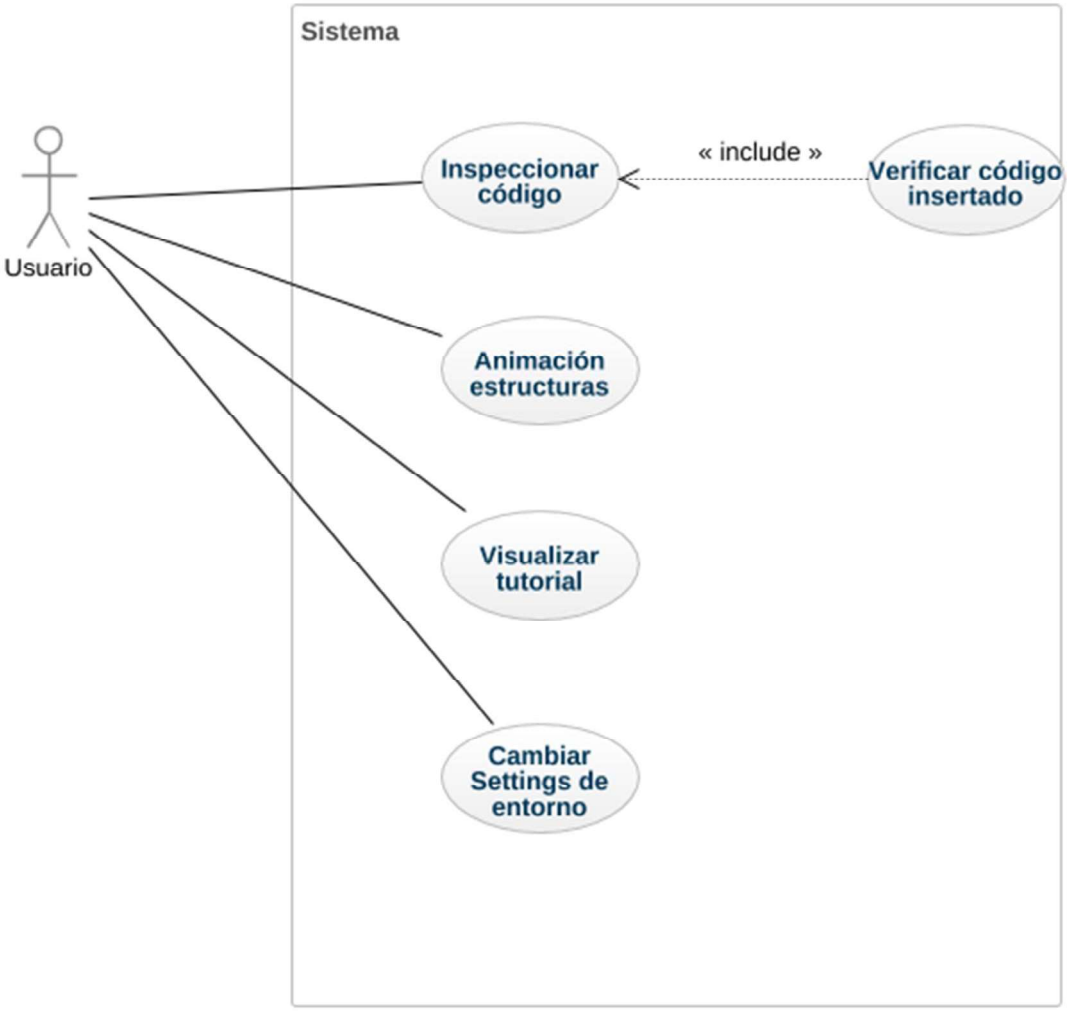


Imagen 4.1 Diagrama casos de uso

A continuación, se escriben los casos de uso de la aplicación. Estos sirven para especificar el comportamiento del sistema, es decir describen el dialogo entre el usuario y sistema.

Caso de uso: Accediendo a la pantalla Tutorial
Actor: Usuario
1) El usuario ejecuta la aplicación
2) El sistema muestra un menú con las opciones 'Code Spy', 'Estructura de datos' y 'Tutorial'
3) El usuario accede a la pantalla Tutorial
4) El sistema muestra la pantalla Tutorial la cual tiene información útil para el usuario referente al programa además de un link para ver la documentación detallada del programa.
5) El usuario accede al link para obtener más información (documentación más detallada)
6) El sistema abre un navegador el cual muestra la documentación

Tabla 4.1 Caso de uso accediendo a la pantalla tutorial

Caso de uso: Ejecutando pantalla Estructura de datos	
Actor: Usuario	
Curso Normal	**Alternativas**
1) El usuario ejecuta la aplicación	
2) El sistema muestra un menú con las opciones 'Code Spy', 'Estructura de datos' y 'Tutorial'	
2) El usuario accede a la pantalla Estructura de datos	
3) El sistema muestra las estructuras disponibles	
3) El usuario selecciona la estructura de datos la cual quiere ver su funcionamiento	
4) El sistema muestra una animación interactiva correspondiente con la estructura de datos seleccionada y sus respectivas acciones.	
5) El usuario clica un botón correspondiente a las acciones que se pueden llevar a cabo con la estructura seleccionada.	
6) El sistema muestra la animación	6.1 Si se produce una excepción el sistema informa de que tipo se trata. Y continua la animación. Volver al paso 5
7) Se repite el paso 5 hasta que el usuario desee	

Tabla 4.2 Caso de uso ejecutando pantalla Estructura de datos

Caso de uso: Examinar código deseado	
Actor: Usuario	
Curso Normal	**Alternativas**
1) El usuario ejecuta la aplicación.	
2) El sistema muestra un menú con las opciones 'Code Spy', 'Estructura de datos' y 'Tutorial'	
3) El usuario accede a la pantalla *Code Spy*	
4) El sistema muestra la pantalla dividida en dos escenarios; en uno de ellos hay un pequeño editor con un botón para activar la edición y en el otro escenario se muestra el valor de los atributos.	
5) El usuario hace clic el botón editar	
6) El sistema refresca el editor, oculta el botón editar y muestra el botón *save.* En el editor se muestra cuatro apartados diferenciados con tokens. Uno para insertar los imports, el otro para los atributos, el otro para insertar el *main* y el último apartado para poner los métodos que el *main* va a utilizar.	
7) El usuario inserta los atributos, el contenido del *main* y los métodos en el editor. Y pulsa el botón *save*	
8) El sistema oculta el botón *save* y muestra el botón editar y compilar	
9) El usuario clica el botón compilar	
10) El sistema se encarga de analizar y procesar el texto insertado en el editor y guarda las líneas en archivos TXTs.	10.1 Si ha ocurrido un error en tiempo de compilación el sistema informa al usuario del error. Saltar al paso 5

Comprueba el código insertado. Ver caso de uso verificar código insertado. Por otro lado, se oculta el botón compilar y se muestra el botón *next* y editar	
11) En usuario clica el botón *next*	
12) El sistema muestra iluminada la línea que será ejecutada con los valores correspondientes de sus variables	12.1 Hay un error en tiempo de ejecución. El sistema te advierte del error. Saltar al paso 5
13) Se repite el paso 11 hasta que ya no haya más líneas de código por ser ejecutadas.	

Tabla 4.3 Caso de uso examinar código deseado

Caso de uso: Verificar código insertado	
Actor: Sistema	
Curso Normal	**Alternativas**
1) El sistema localiza la carpeta data y extrae el código de los txt	
2) El sistema construye a partir del código introducido por el usuario y unos tokens definidos por el programador un archivo .java y posteriormente lo trata de compilar.	2.1 Hay un error en tiempo de compilación. El sistema guarda el tipo de excepción para mostrar posteriormente al usuario. Finaliza caso uso
3) El sistema crea los archivos .java y .class correspondiente	

Tabla 4.4 Caso de uso verificar código insertado

Caso de uso: Cambiando Settings del entorno	
Actor: Usuario	
Curso Normal	**Alternativas**
1) El usuario localiza la carpeta data del proyecto. Posteriormente abre la carpeta *settings* y modifica el fichero EnvironmentSettings.txt	
2) El usuario modifica el valor de las variables ya predefinidas con el valor deseado	2.1 El usuario no encuentra la característica que desea cambiar. Finaliza caso de uso.
3) El usuario ejecuta la aplicación.	
4) El sistema se muestra con los cambios efectuados por el usuario.	4.1 En caso de haber algún error en los valores introducidos por el usuario el sistema deja los valores por defecto

Tabla 4.5 Caso de uso cambiando *settings* del entorno

4.4 Descripción de las clases

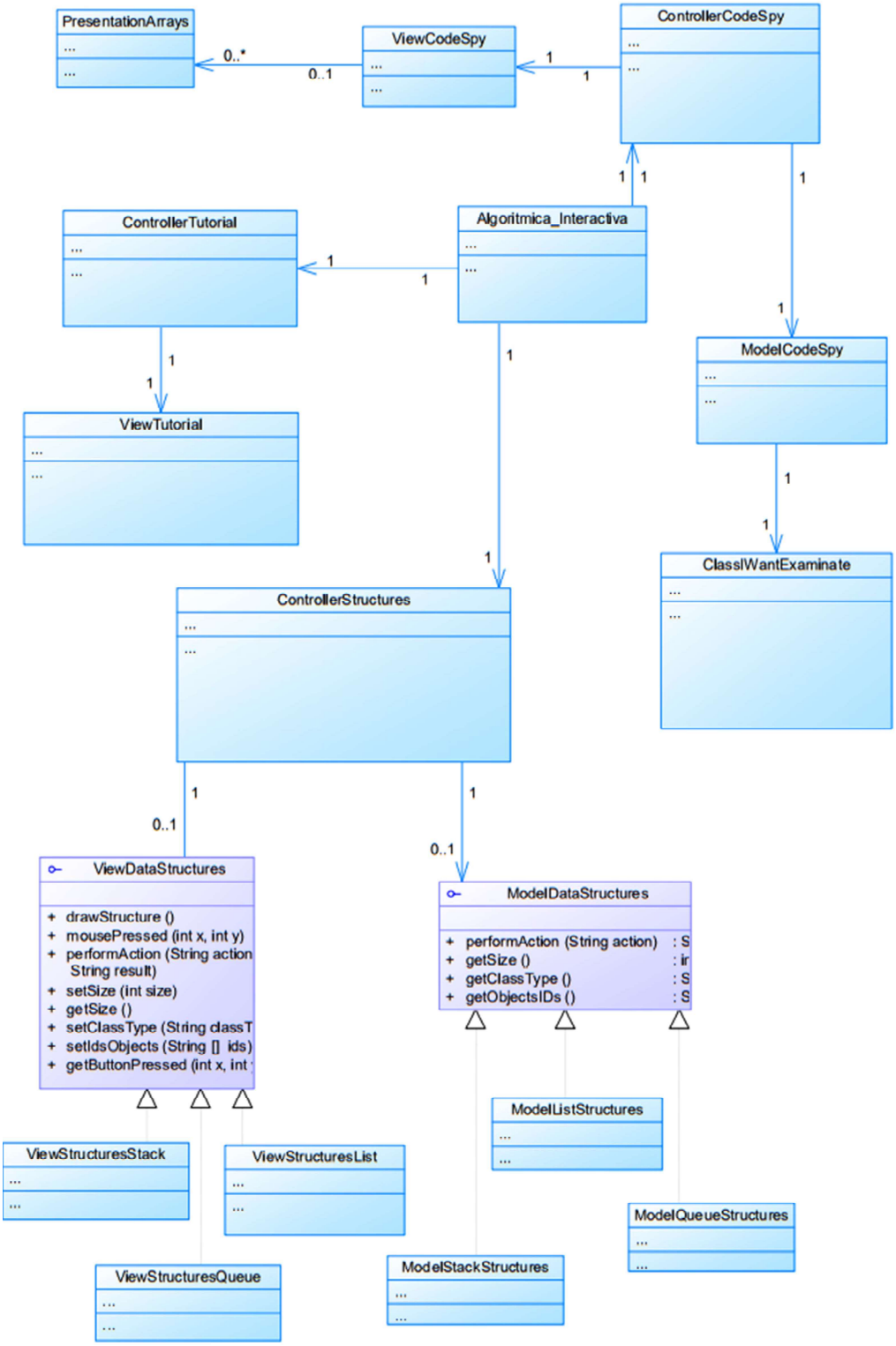


Imagen 4.2 Diagrama de clases simplificado

A continuación, se explica la función de cada clase del proyecto

Algoritmica_Interactiva:

Clase principal encargada de ejecutar e iniciar la animación, carga tanto las características del entorno (color de la letra y el tamaño de la pantalla) como la animación del Code Spy, Estructuras de datos y Tutorial.

Esta clase posee métodos propios de Processing como:

setup: Solo se ejecuta una vez y se invoca justo después de iniciar la aplicación. Este método será el encargado de inicializar todos los controladores, vistas y modelos.

draw: Método iterativo siempre en ejecución. Se ejecuta después del *setup* automáticamente y deja de ejecutarse al cerrar el sketch.

mousePressed: Método propio que se ejecuta al hacer clic al botón del mouse.

Al no existir botones en Processing estos deben de dibujarse o en su defecto exportarse usando librerías externas. El método *mousePressed* verifica en que lugar de la pantalla se ha clicado y posteriormente se lo comunica al controlador oportuno.

ControllerTutorial:

Clase encargada de procesar la información del tutorial y posteriormente pasarla a la vista.

ViewTutorial:

Clase encargada de mostrar por pantalla la información del tutorial.

ControllerStructures:

Clase encargada de pasar información del modelo a la vista y viceversa. Si el usuario realiza una acción esta clase se lo comunica al modelo y posteriormente le pasa el resultado a la vista.

ViewStructures:

Clase encargada de animar las estructuras de datos, es decir, representar las acciones de dichas estructuras de forma visual.

ModelStructures:

Clase que posee la lógica de las estructuras. En ella se pueden realizar las acciones de la estructura seleccionada y esta devuelve el valor del resultado de dicha acción. Por ejemplo, si la estructura es una queue y se desea recuperar y eliminar la cabeza de la cola se pueden utilizar las acciones *poll* y *remove*. Si se utiliza la acción poll() en una cola vacía esta devolverá un null pero en cambio si se utiliza un remove() en una cola vacía esta acción nos lanzará una excepción. Entonces el modelo será el encargado de realizar la acción correspondiente y el resultado de dicha acción lo devolverá en forma de *String*

ControllerCodeSpy:

Clase responsable de la comunicación entre la vista y el modelo. Se encarga de pasar a la vista las líneas de código y valores correspondientes.

ViewCodeSpy:

Clase encargada de mostrar las líneas de código y valores que le pasa el controlador.

ModelCodeSpy:

Clase encargada de la construcción de la clase ClassIWantExaminate con sus respectivos tokens y métodos para permitir el análisis de su comportamiento en tiempo de ejecución.

Por otro lado, instancia el objeto ClassIWantExaminate usando *reflection* para obtener los valores de los atributos y posteriormente poder devolver esas líneas.

ClassIWantExaminate:

Clase construida a partir de los ficheros txt con código que el usuario quiere examinar, además de métodos de espera y tokens introducidos por el programa para permitir su control

4.5 Diagramas de secuencia

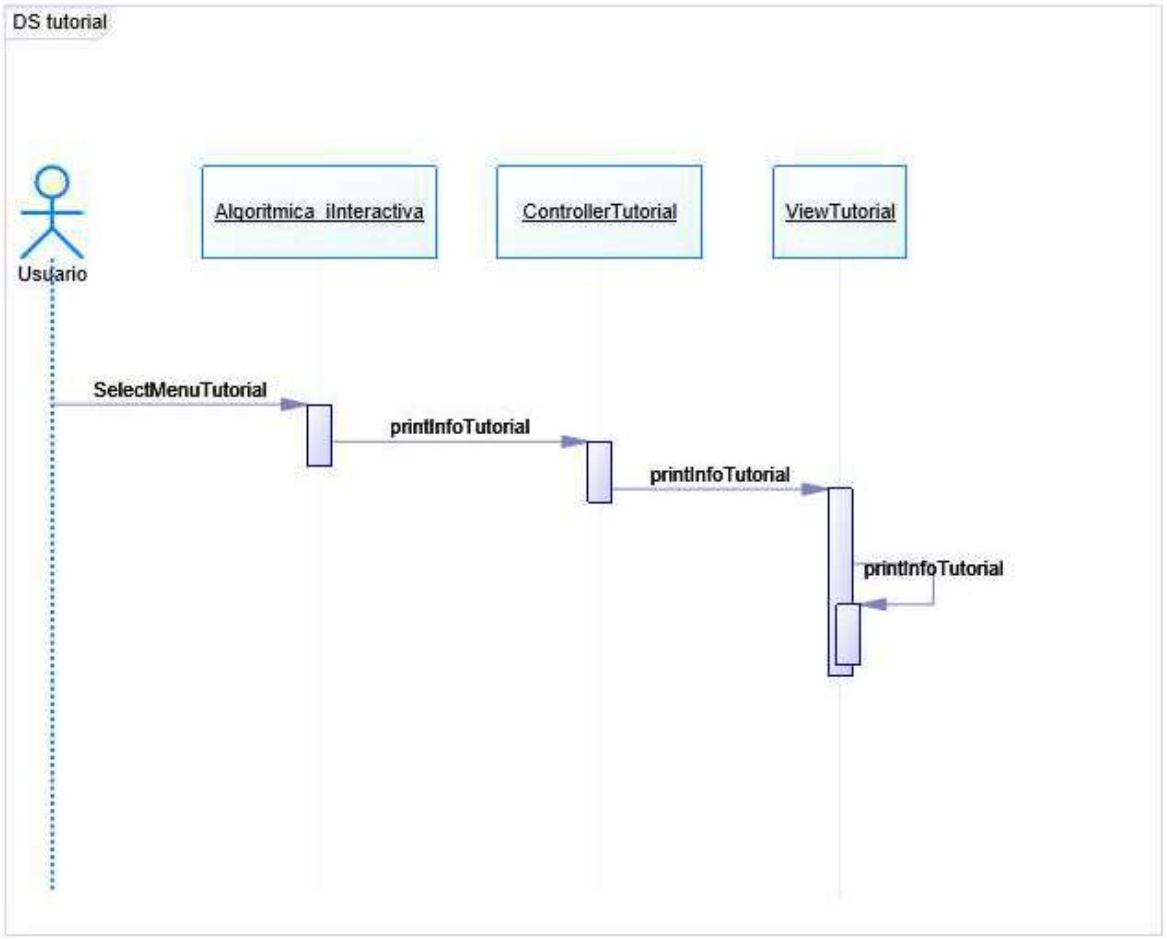


Imagen 4.3 Diagrama secuencia Tutorial

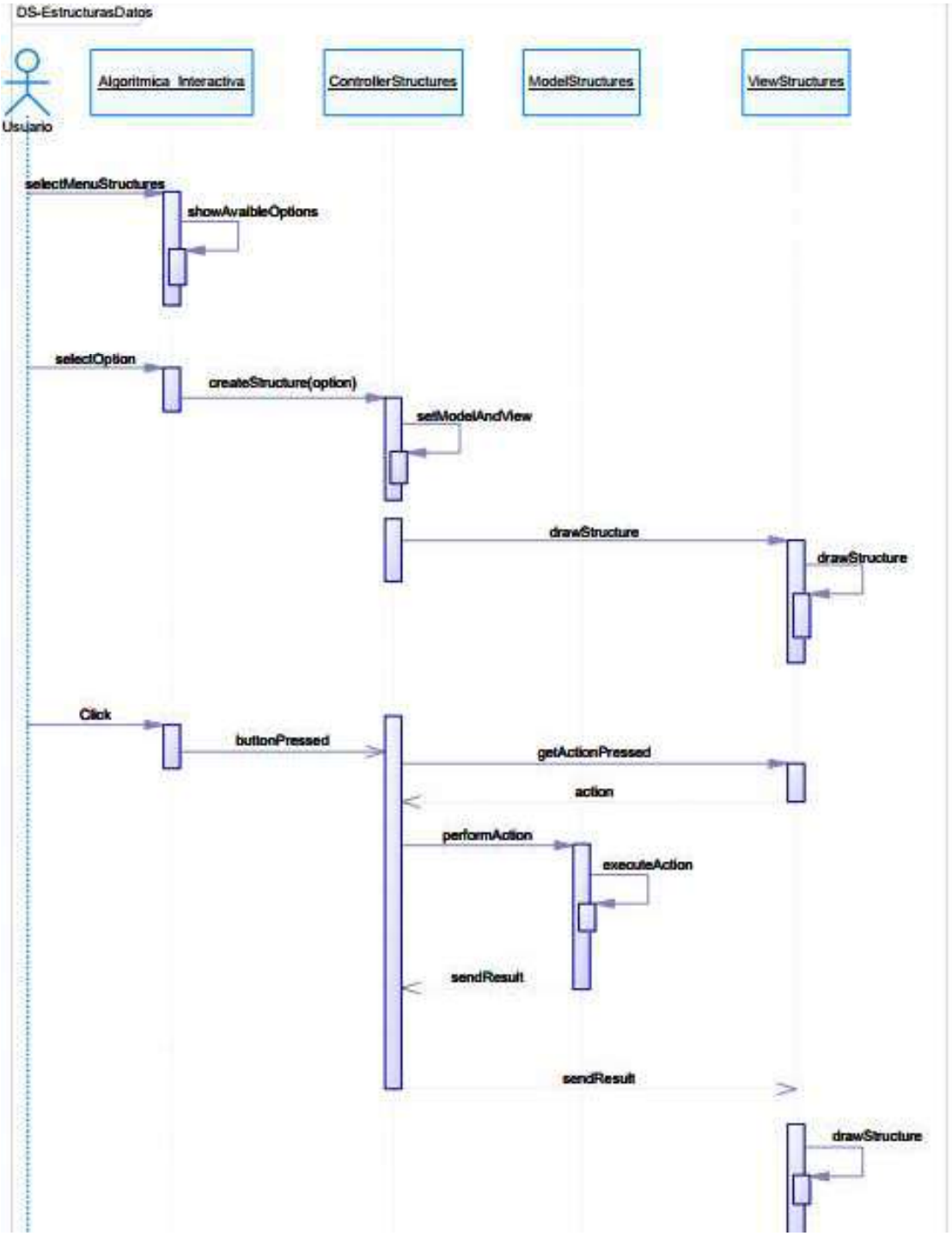


Imagen 4.4 Diagrama secuencia Estructuras de Datos

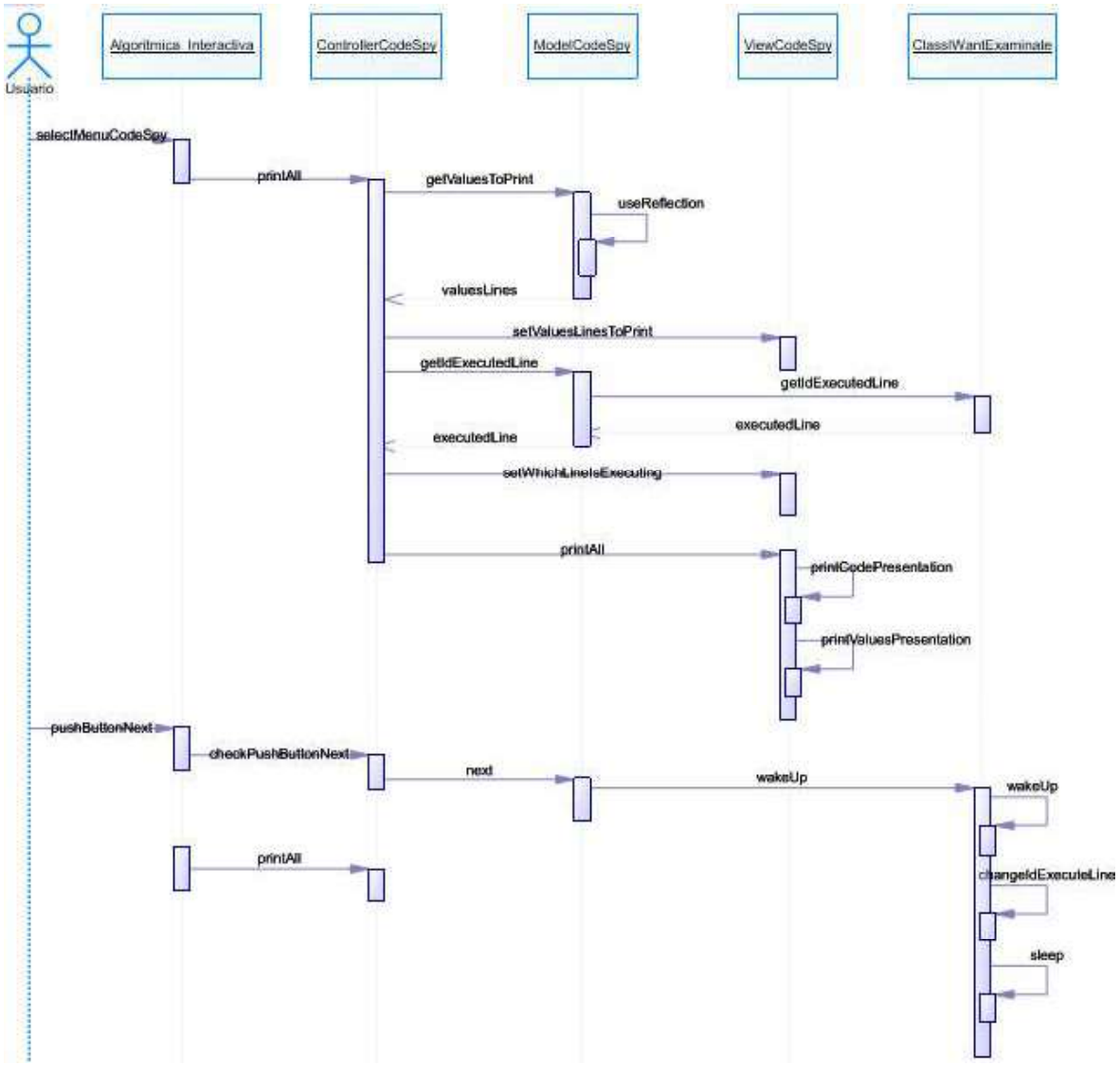


Imagen 4.5 Diagrama secuencia CodeSpy

5 IMPLEMENTACIÓN

Se ha llegado al apartado implementación. A continuación, se describe cómo se ha afrontado la concurrencia y sincronización en el programa. Por otro lado, se muestra el funcionamiento del software

5.1 Concurrencia en el programa

Para la implementación de la parte *Code Spy* es necesario utilizar dos hilos de ejecución.

Uno de ellos, el principal, se encargará de hacer la presentación. Y el otro se encargará de ejecutar la clase creada por el usuario. En otras palabras, el hilo principal se encarga de inicializar todos los componentes necesarios, la creación de las animaciones y el control de la interacción con la clase a examinar. Mientras, el hilo secundario se encarga de ir ejecutando línea por línea el código de la clase ClassIWantExaminate, clase construida a partir de los txt con el código del usuario.

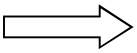
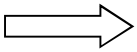
	Interacción todas las clases del proyecto	Comunicación	ClassIWantExaminate	
Thread 1	Ejecución programa. Comienza hilo de ejecución principal			
	Inicializa los componentes.	Creación de hilo 2 para ejecutar y poder examinar la clase que contiene el código del usuario		**Thread 2**
	El usuario clica por primera vez el botón next		Se ejecuta la primera línea de código del objeto creado que contiene un sleep introducido por los métodos de control del programador.	
	Continua con la ejecución. Se muestra por pantalla los valores y el código que está a punto de ejecutarse		Sleep mientras nadie diga lo contrario	
	Usuario presiona botón *next*	Es cambiado el valor de un atributo que controla el cambio de estado entre sleep y no sleep de la clase examinada. 	Ejecuta la primera sentencia. Sleep mientras nadie diga lo contrario	
	Continua ejecución		Sleep mientras nadie te diga lo contrario	
…	…	…	…	

Tabla 5.1 Representación concurrencia en el programa

5.2 Sincronización entre clases

Existen dos escenarios bien diferenciados uno de ellos es la vista (lo que se visualiza por pantalla) y el otro es la ejecución real del código del usuario.

A continuación, se explica cómo se consigue que estas dos partes estén sincronizadas. Para ello es necesario conocer la construcción de la clase que contiene el código del usuario, es decir, la clase a examinar.

Construcción de la clase ClassIWantExaminate:

La clase ClassIWantExaminate se crea a partir del código de usuario introducido en el editor y de tokens y métodos de control insertados por el programa.

El editor tiene 4 partes diferenciadas con tokens. El usuario debe rellenar debajo de cada token las líneas de código que corresponda a los *imports*, atributos, el contenido del código que se quiere ejecutar (el *main*) y finalmente los métodos que se utilizarán en la ejecución.

El código de usuario introducido debajo del token *main* se insertará en el método *ejecutar* de la clase ClassIWantExaminate (método creado por el programa). Antes de cada línea de código del usuario habrá un método de control llamado *esperar* que se le pasará por parámetro la línea que va ser ejecutada. Con ello se conseguirá saber en que línea específica está la ejecución.

Fichero código.txt	Método ejecutar de la clase IWantExaminate
Sentencia1 // linea0 **Sentencia2** //linea1	esperar(-1) /* método control insertado por el programa. Todavía no ha empezado la ejecución */ esperar(0) // el parámetro es referente a qué línea será ejecutada **Sentencia1** esperar(1) **Sentencia2**

Tabla 5.2 Construcción método ejecutar

La clase ClassIWantExaminate también posee atributos de control. Uno de ellos es para identificar qué método se va a ejecutar y en qué número de línea está la ejecución. Y el otro es un flag para cambiar el estado entre sleep y no sleep del thread que ejecutará la clase.

Fichero código.txt	Método ejecutar de la clase IWantExaminate
métodoPropio// linea0 **Sentencia2** //linea1	esperar(-1) esperar(0) whoIsExecutingNow = "nombreDelMétodoPropio" **métodoPropio()** whoIsExecutingNow = "main" /*atributo de control. Le dice a la clase que el método principal se va a ejecutar */ esperar(1) **Sentencia2**

Tabla 5.3 Construcción método ejecutar con invocación de métodos propios

5.3 Librería externa G4P

Para facilitar la inserción de código en Code Spy se ha utilizado la librería G4P [9] para poder utilizar tanto un TextArea como los botones.

Dicha librería proporciona un conjunto de controles GUI 2D y soporta múltiples ventanas. Cuenta con elementos como botones, *textfields, textareas, checkbox*, *sliders*, cadena de caracteres con estilos, etc. Y se exporta desde el propio editor de Processing

5.4 Funcionamiento del programa

Al ejecutar la aplicación aparece un menú con tres apartados *Code spy*, Estructuras y Tutorial

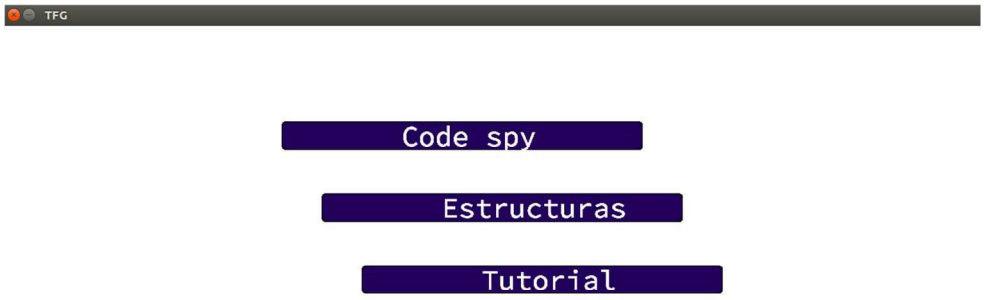


Imagen 5.1 Menú de la aplicación

Después de pulsar el botón Code Spy aparece un pequeño editor con un botón editar.

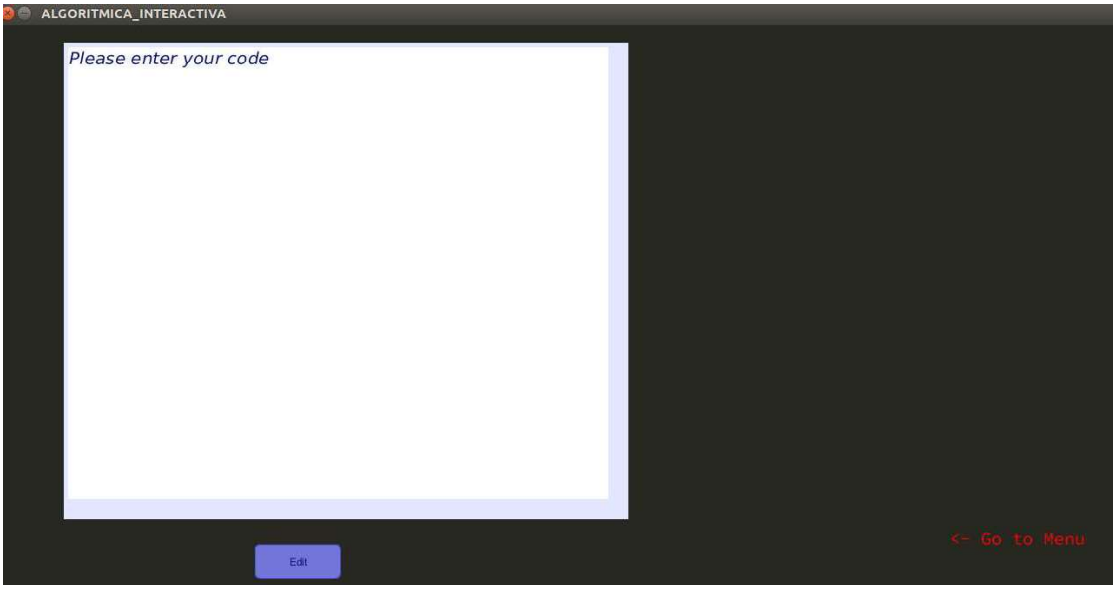


Imagen 5.2 Presentación CodeSpy

Si se pulsa el botón editar aparecen 3 tokens en el editor, aparece el botón *save* y desaparece el botón *edit*.

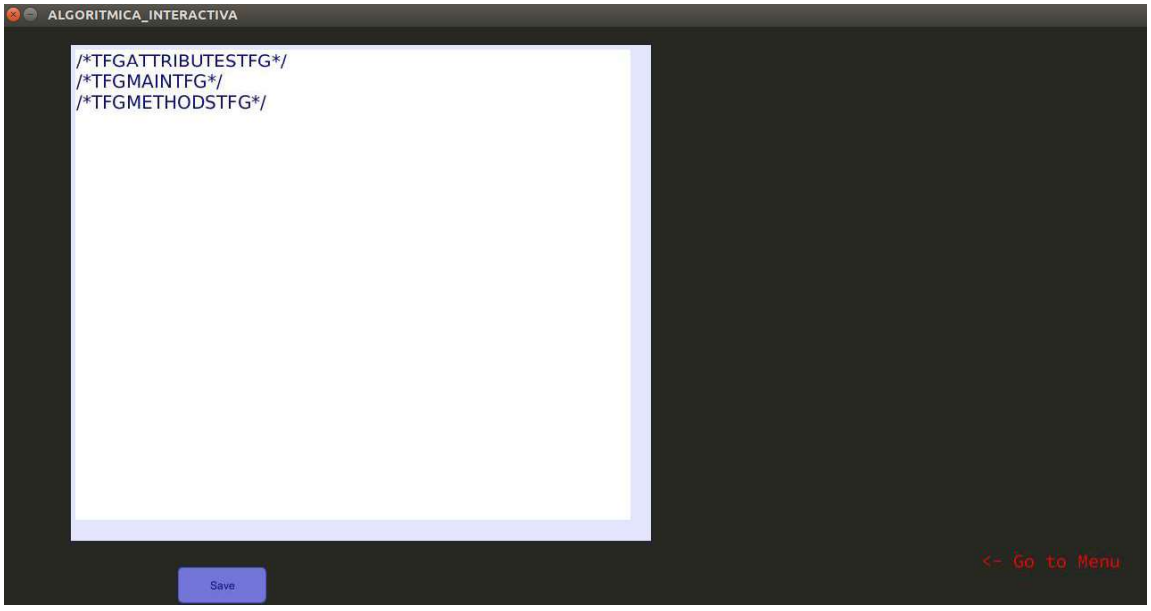


Imagen 5.3 Editor que muestra los 3 tokens

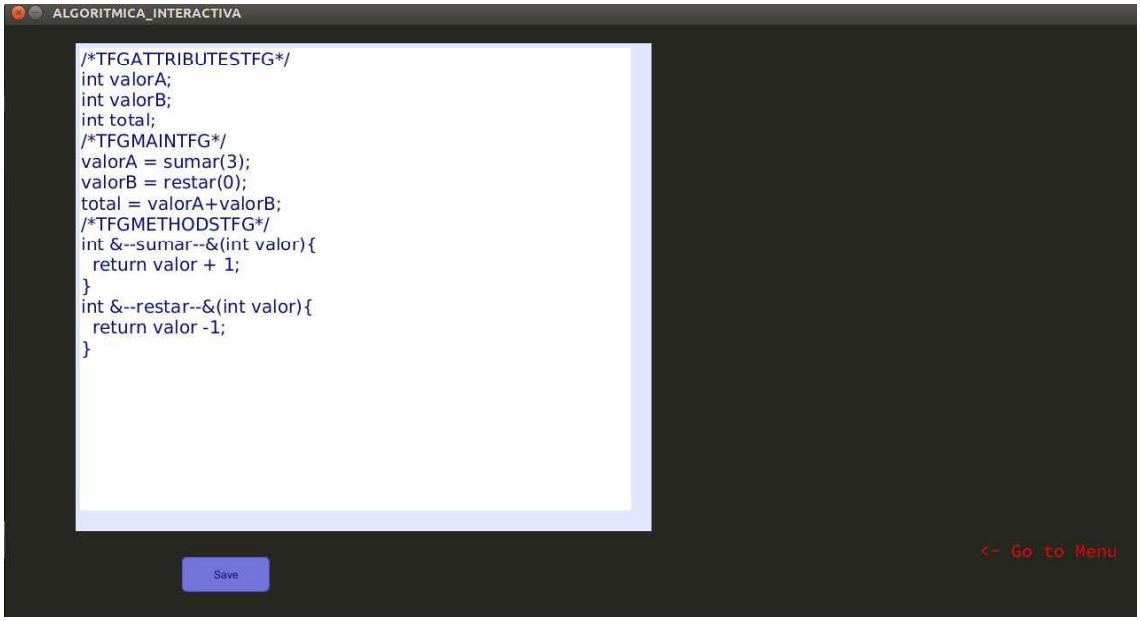


Imagen 5.4 Editor con el código de usuario

Y se puede proseguir a guardar, compilar y posteriormente a espiar el código pulsando el botón de *next*.

En la imagen 4.8 se pueden ver dos escenarios. Uno de ellos correspondiente a presentación del código por pantalla y la otra corresponde a presentación de valores.

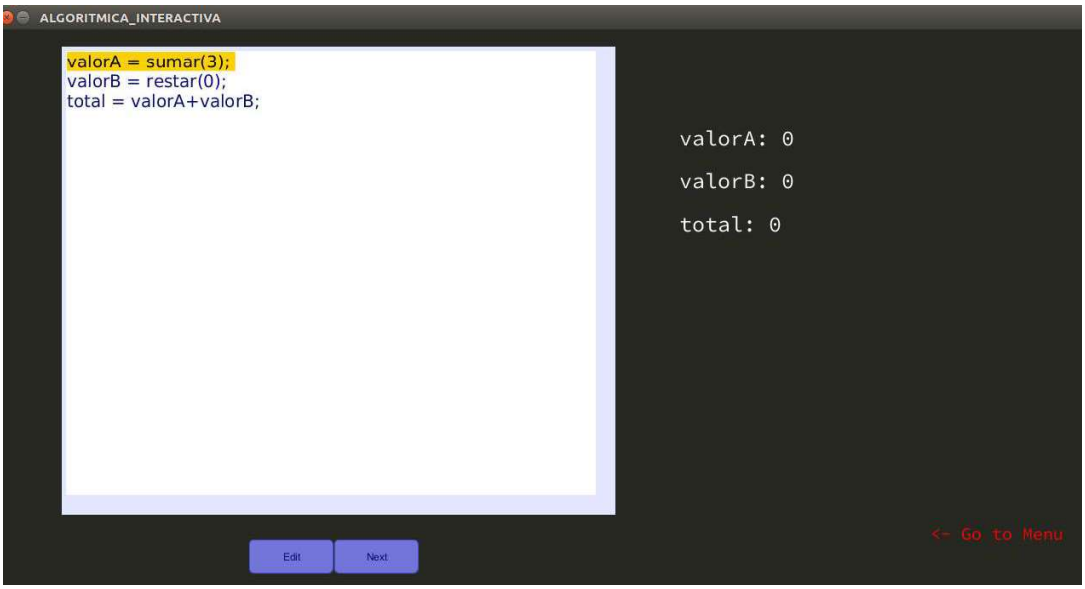


Imagen 5.5 Presentación Code Spy a punto de ejecutar la primera línea

Cuando se está a punto de espiar un método se cambia de presentación de código por el del método correspondiente. Ver imagen 5.6.

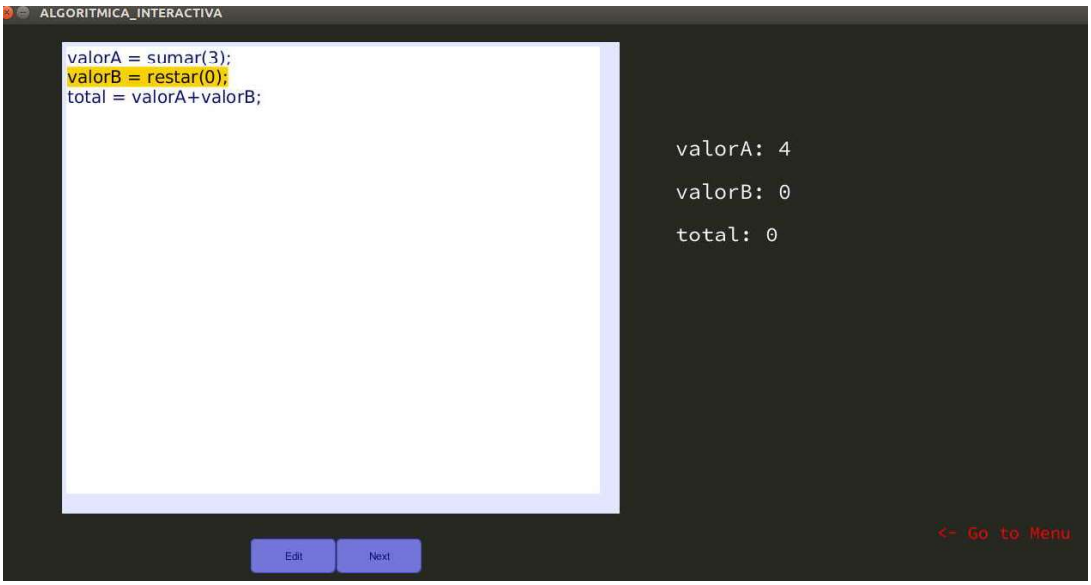


Imagen 5.6 Presentación Code Spy a punto de ejecutar un método

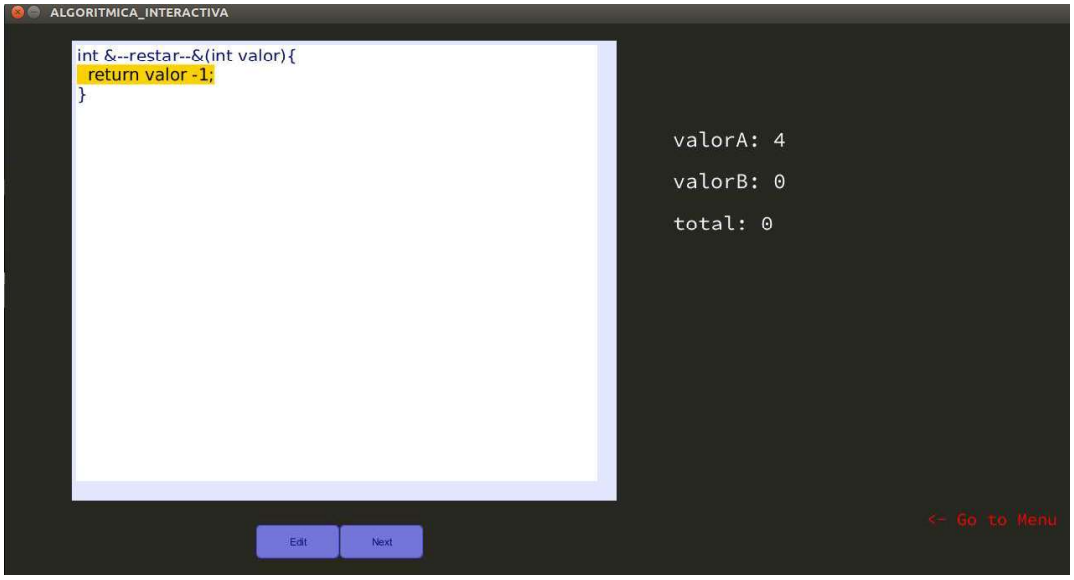


Imagen 5.7 Presentación Code Spy dentro del método a ejecutar

Por otro lado, si se elige la opción Estructuras de datos del menú aparece una lista con las estructuras implementadas.

A continuación, se muestra la pantalla para animar una cola. En ella se encuentran la animación con representaciones de objetos, las acciones que se pueden realizar con dicha estructura, el retorno de los métodos utilizados y finalmente la clase que se ha utilizado para implementar la cola

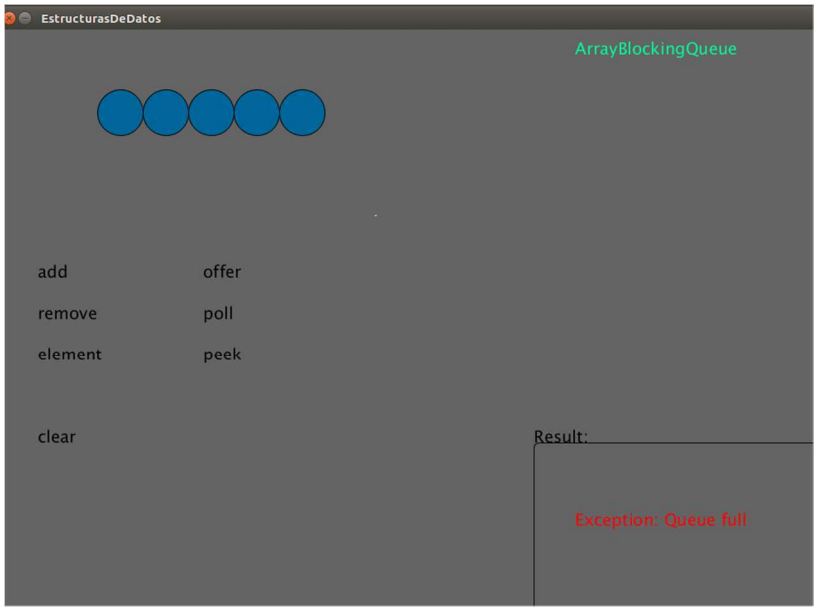


Imagen 5.8 Animación Estructuras de datos - Queue

Imagen 5.9 Animación Estructuras de datos - Stack

Por otro lado, la pantalla tutorial cuenta con una breve descripción del funcionamiento del software y un link para consultar información más exhaustiva.

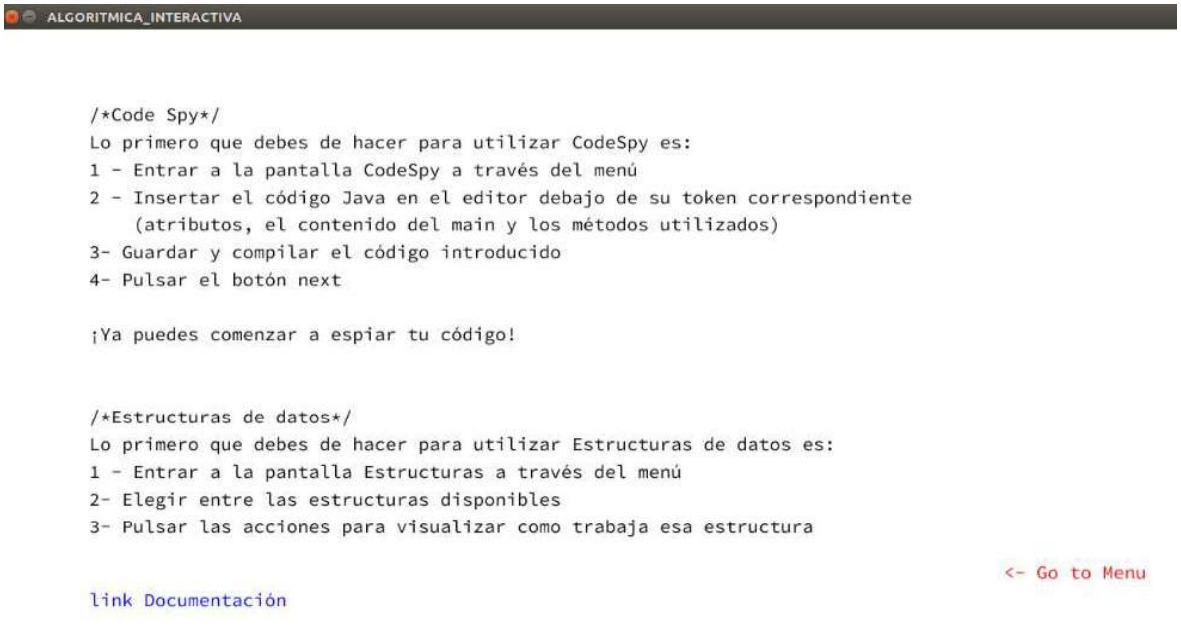


Imagen 5.10 Pantalla tutorial

6 POSIBLES AMPLIACIONES

En este apartado se describe las posibles futuras mejoras y ampliaciones del software. Por el momento se ha elaborado un prototipo sencillo para que cumpla las funcionalidades de lectura y ejecución de código además de mostrar el funcionamiento de algunas estructuras de datos.

Una posible ampliación sería en el apartado *Code Spy* mostrar la animación de todas las estructuras de datos y no solo del array como se hace en este momento. También se podría ampliar la parte de estructura de datos y mostrar más animaciones de otras estructuras no implementadas, como por ejemplo conjuntos, mapas, grafos, árboles, etc.

7 BIBLIOGRAFÍA

[1] https://es.wikipedia.org/wiki/Java_Runtime_Environment, Wikipedia, 13 de abril 2017

[2] https://es.wikipedia.org/wiki/Java_(lenguaje_de_programación), Wikipedia, 31 de julio 2017

[3] Document GEI_PropostesProfessorsTFG T17, Enric Sesa, 2017

[4] https://penjee.com, Penjee, 28 de agosto 2017

[5] https://www.codecademy.com, Codecademy, 28 de agosto 2017

[6] https://www.w3schools.com, W3schools, 28 de agosto 2017

[7] https://processing.org, Processing, 6 de septiembre 2017

[8] https://docs.oracle.com/javase/tutorial/reflect, Oracle, 1 de septiembre 2017

[8] https://docs.oracle.com/javase/tutorial/reflect, Oracle, 1 de septiembre 2017

[9] http://www.lagers.org.uk/g4p/, G4P, 15 de septiembre 2017

Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Grado en ingeniería informática

Algorítmica interactiva

Estudio de viabilidad

Autor: Michelle Cabrera Carranza

Tutor: Enric Sesa Nogueras

Otoño 2017

TecnoCampus
Mataró-Maresme

Índice

Índice de tablas

1 ANÁLISIS DE VIABILIDAD TÉCNICA

Para realizar la aplicación se utilizará el lenguaje de programación y herramienta de desarrollo de Processing. Sobre Processing se puede decir que es un software de código abierto está en constante desarrollo y mejoras. Dispone de muchas librerías creadas por sus propios usuarios.

Por otro lado, es necesario tener un ordenador con un sistema operativo Windows (versiones igual o posteriores a Windows 7), MAC OS o Linux, dado que Processing los soporta. Además se debe de tener instalado el JDK en las carpetas de Processing

Para finalizar es necesario internet para obtener información y ayuda.

2 ANÁLISIS DE VIABILIDAD ECONÓMICA

A continuación, se elabora una estimación con la técnica de puntos de caso de uso. El objetivo es estimar las horas necesarias para la realización del software.

Lo primero que se realizará es el cálculo de los puntos de caso de uso no ajustados (UUCP) para ello es necesario calcular el peso de los actores (AUW) y el peso de los casos de uso (UUCW).

Casos de uso:	**AUW**	**UUCW**	**UUCP**
1- Accediendo a la pantalla Tutorial	3	5	8
2- Ejecutando pantalla Estructura de datos	3	10	13
3- Examinar código deseado	3	10	13
4- Cambiando Settings del entorno	3	5	5
		Total	**36**

Tabla 2.1 Calculo de puntos de uso no ajustado (UUCP)

El peso de los actores en todos los casos de uso es de 3 dado que el actor interactúa a través de una interfaz gráfica, así que es un actor complejo.

Por otro lado, para determinar el factor de peso de los casos de uso sin ajustar (UUCW) se ha tenido en cuenta las clases en análisis. Por lo tanto, en el caso de uso 1 y 4 se han utilizado menos de 5 clases por lo que tiene una complejidad simple así que, su valor es 5. En cambio, para el caso de uso 2 y 3 se han empleado entre 5 a 10 clases así que el valor que le corresponde es 10.

Posteriormente se determinará el peso de los factores técnicos.

Factor	**Descripción**	**Peso**	**Influencia**	**Resultado**
R1	Sistema Distribuido	2	0	0
R2	Objetivos de rendimiento	1	0	0
R3	Eficiencia respecto al usuario final	1	1	1
R4	Procesamiento complejo	1	2	2
R5	Código reutilizable	1	1	1
R6	Instalación sencilla	0,5	4	2
R7	Fácil utilización	0,5	0	0
R8	Portabilidad	2	3	6
R9	Fácil de cambiar	1	0	0

R10	Uso concurrente	1	0	0
R11	Características de seguridad	1	0	0
R12	Accesible por terceros	1	0	0
R13	Se requiere formación especial	1	2	2
			Resultado Total	**14**

Tabla 2.2 Calculo para determinar el peso de los factores técnicos

Y se determina el peso de los factores ambientales.

Factor	**Descripción**	**Peso**	**Influencia**	**Resultado**
R1	Familiaridad con el modelo del proyecto utilizado	1,5	3	4,5
R2	Experiencia en la aplicación	0,5	3	1,5
R3	Experiencia con orientación a objetos	1	5	5
R4	Capacidad del analista líder	0,5	5	2,5
R5	Motivación	1	5	5
R6	Requisitos estables	2	5	10
R7	Trabajadores a tiempo parcial	-1	5	-5
R8	Dificultad en el lenguaje de programación	-1	3	-3
			Total:	**20,5**

Tabla 2.3 Calculo para determinar el peso de los factores ambientales

A continuación, se calcula el EF para así poder calcular el valor del UCP.

EF=1.4+(-0,03*EFACTOR) → 0,785

UCP = UUCP * TCF * EF → 20,9124

Para estimar las horas hombres primero se debe obtener el CF. Para ello hay que contar los factores ambientales del R1 a R6 que tienen un valor menor a 3. Por otro lado, se debe de contar los factores ambientales R7 y R8 que tienen un valor mayor de 3

Horas-Persona (CF)	Descripcion
20	contador <= 2
28	contador <= 4
36	contador >= 5

Tabla 2.4 Estimación hora–hombre

Así que se determina que el valor horas-Persona (CF) es 20

Posteriormente se puede establecer las horas-hombre totales del proyecto

Cálculo del esfuerzo horas hombre: E = UCP*CF → 418,248

Por lo tanto, para este sistema son necesarias 21 semanas de trabajo para finalizarlo. (Con una sola persona trabajando siendo su jornada de 20 horas semanales)

En resumen:

Resumen del proyecto		
Actividad	**Porcentaje de tiempo**	**Horas-Hombre**
Análisis	10%	104,562
Diseño	20%	209,124
Programación	40%	418,248
Pruebas	15%	156,843
Sobrecarga	15%	156,843
Total	**100%**	**1045,62**

Tabla 2.5 Tabla resumen horas-hombres totales para la finalización del proyecto

Dado el total de horas hombre necesarias para finalizar el proyecto y si los desarrolladores cobrasen un sueldo de 18 euros/hora se puede determinar que el coste total del proyecto sería

Coste total = 1045,62 x 18 = 18.821,16 euros

3 ASPECTOS LEGALES

El estatus legal con el que se protege este trabajo final de carrera es Reconocimiento-Compartirigual. CC BY-SA. Esta licencia permite a otros re-mezclar, modificar y desarrollar sobre la obra incluso para propósitos comerciales, siempre que te atribuyan el crédito y licencien sus nuevas obras bajo idénticos términos. [1]

Escola Universitària Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Grado en ingeniería informática

Algorítmica interactiva

Anexos

Autor: Michelle Cabrera Carranza

Tutor: Enric Sesa Nogueras

Otoño 2017

TecnoCampus Mataró-Maresme

Índice

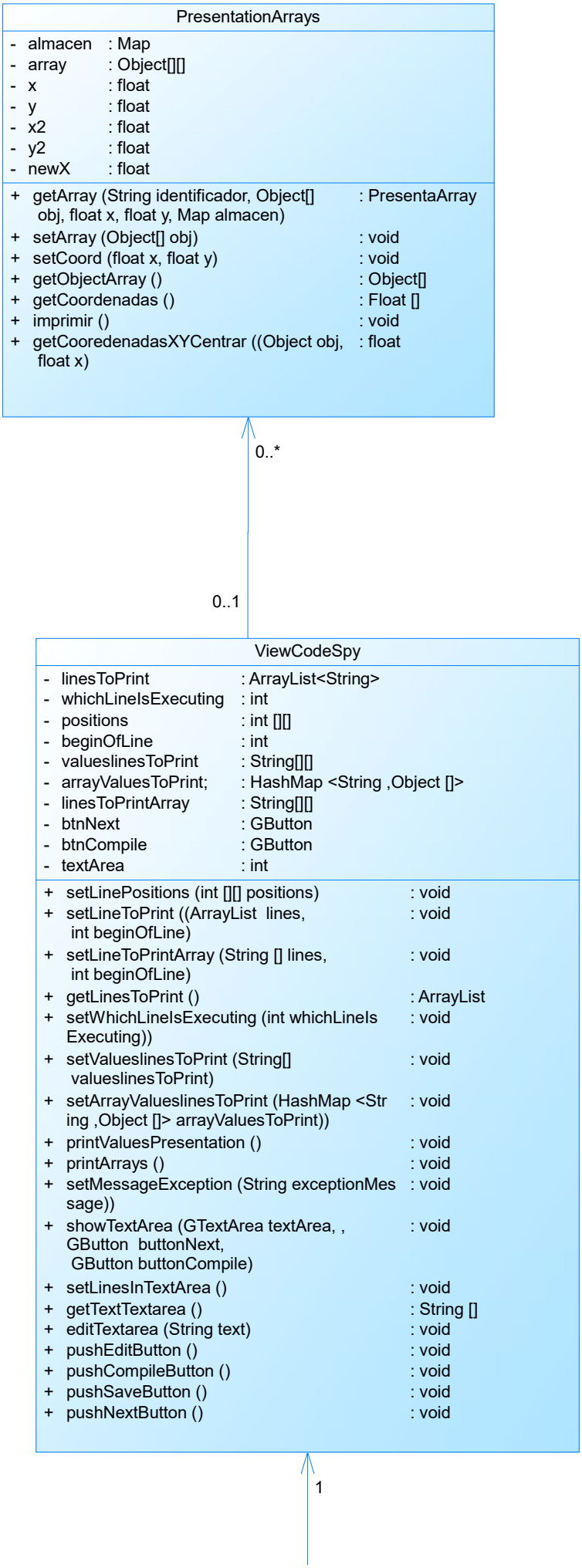
ANEXO 1. CONTENIDO DEL CD-ROM

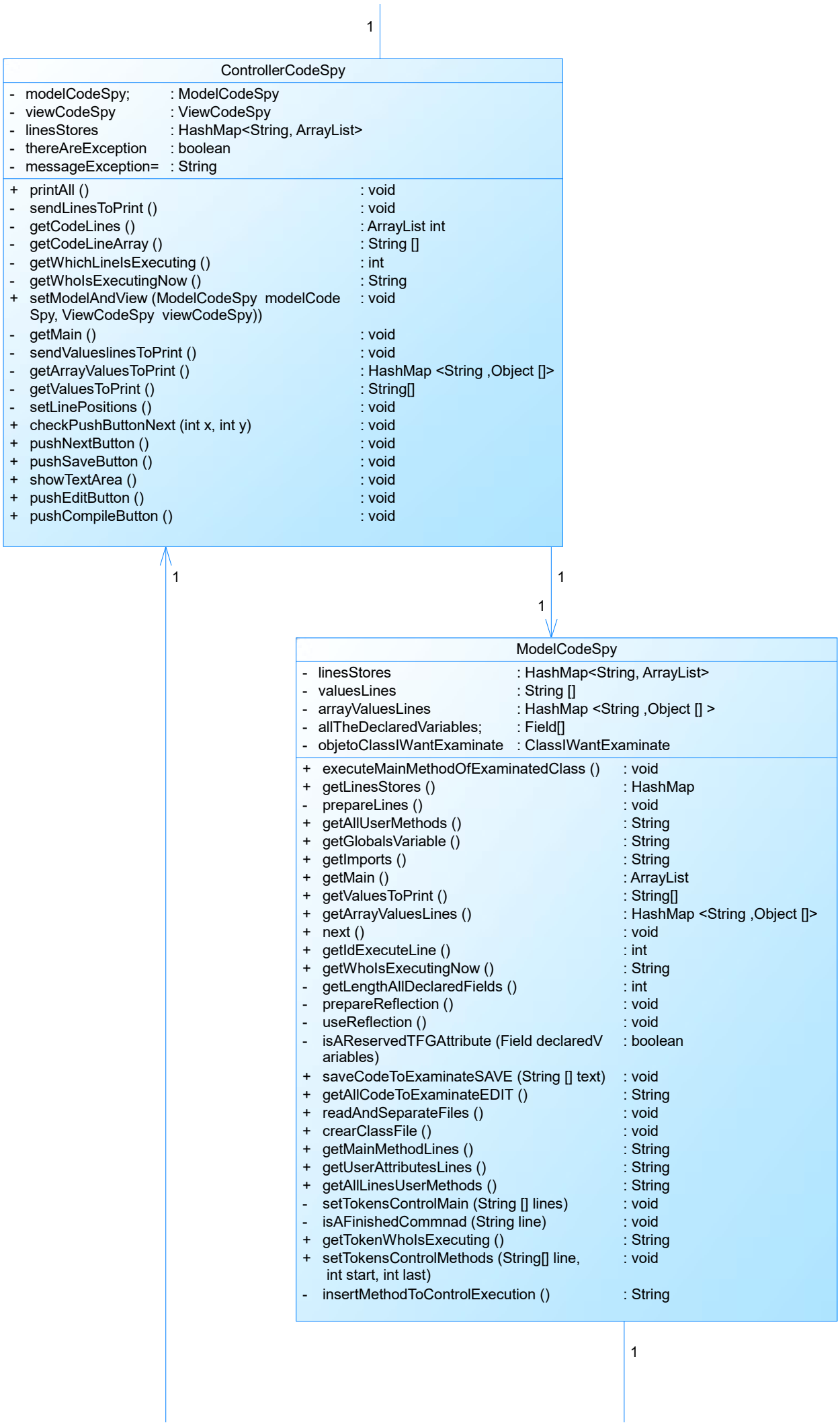
- Documentación del proyecto (memoria y presupuesto)

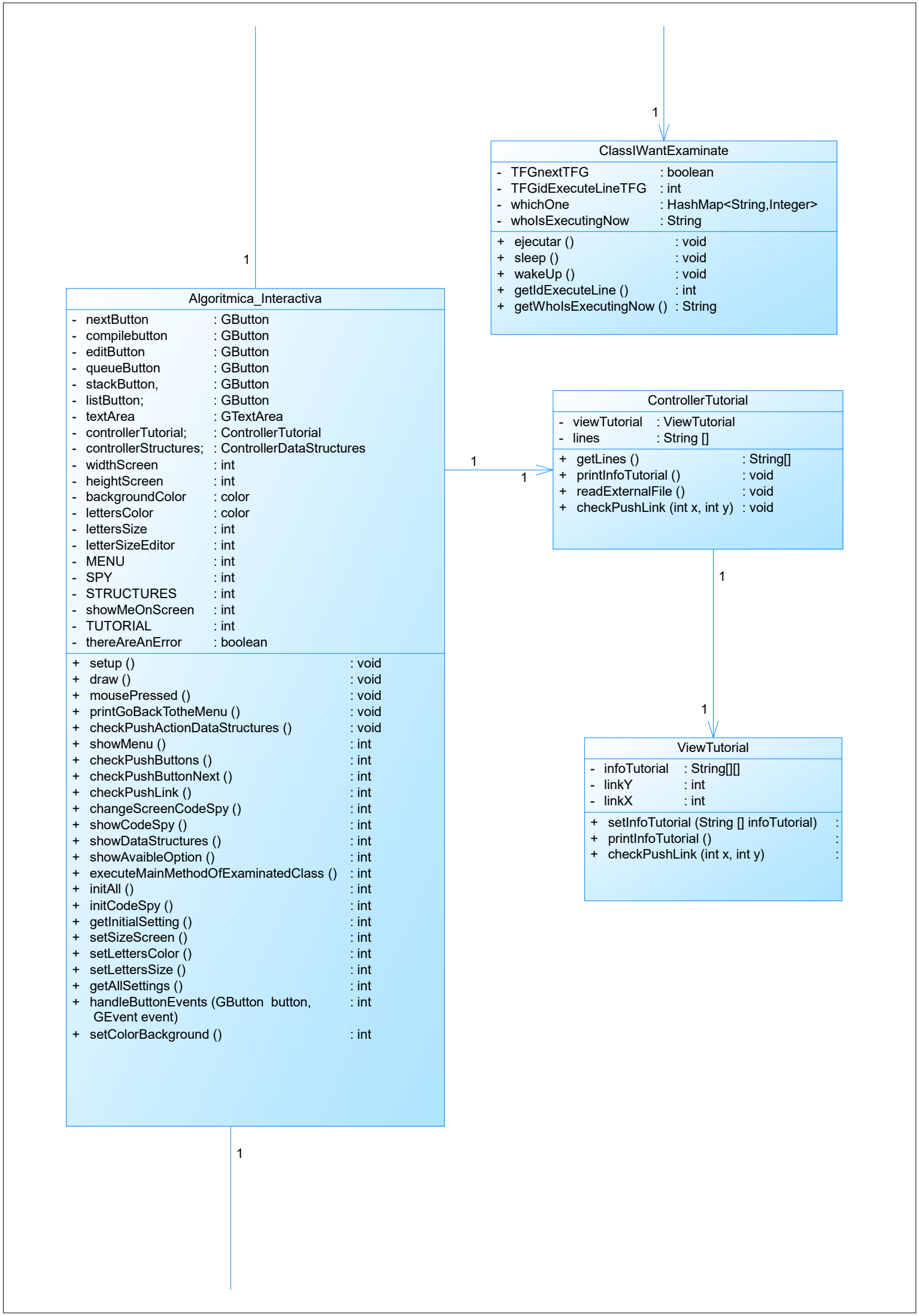
- Diagramas

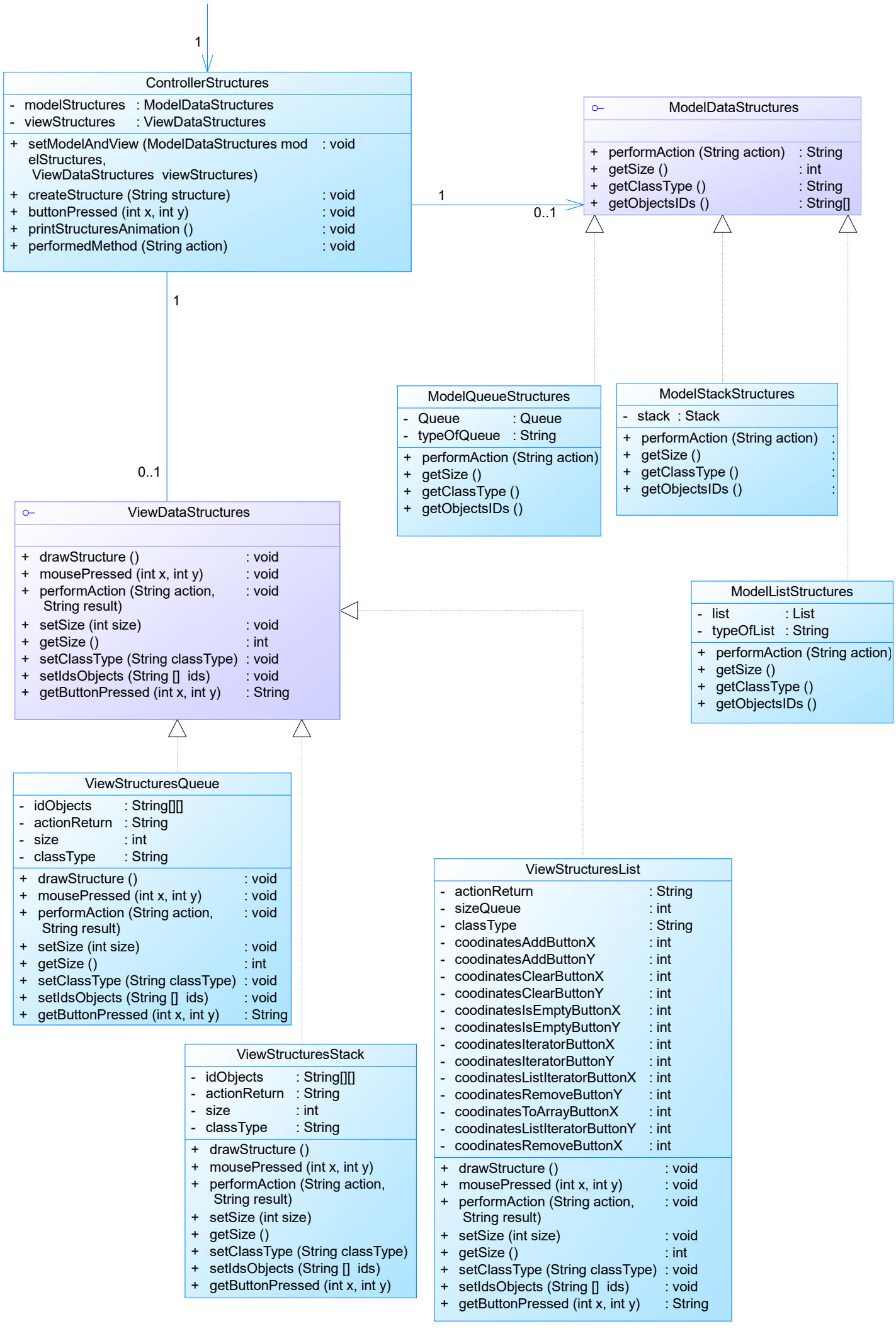
- Proyecto Processing con el código

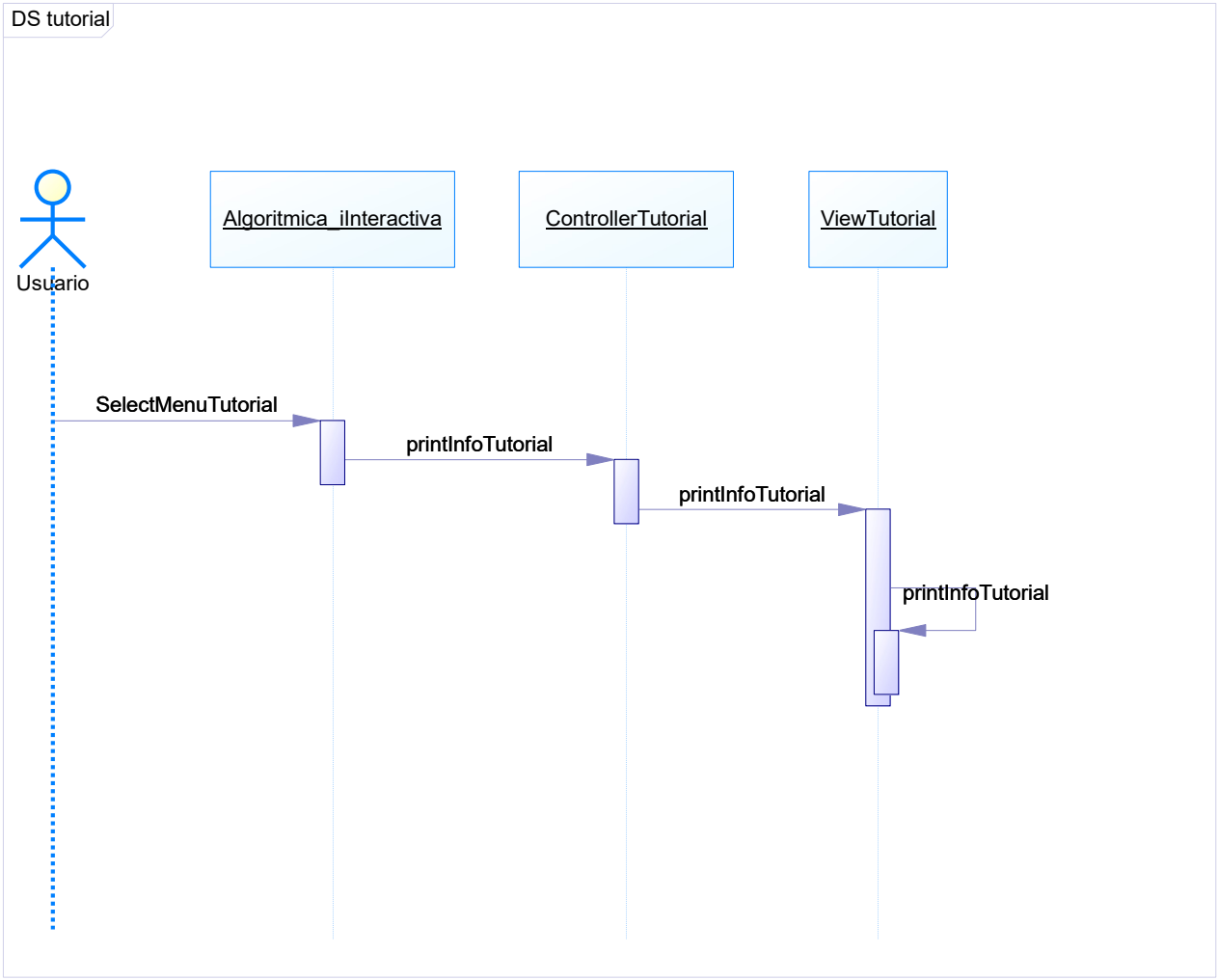
ANEXO 2. DIAGRAMAS

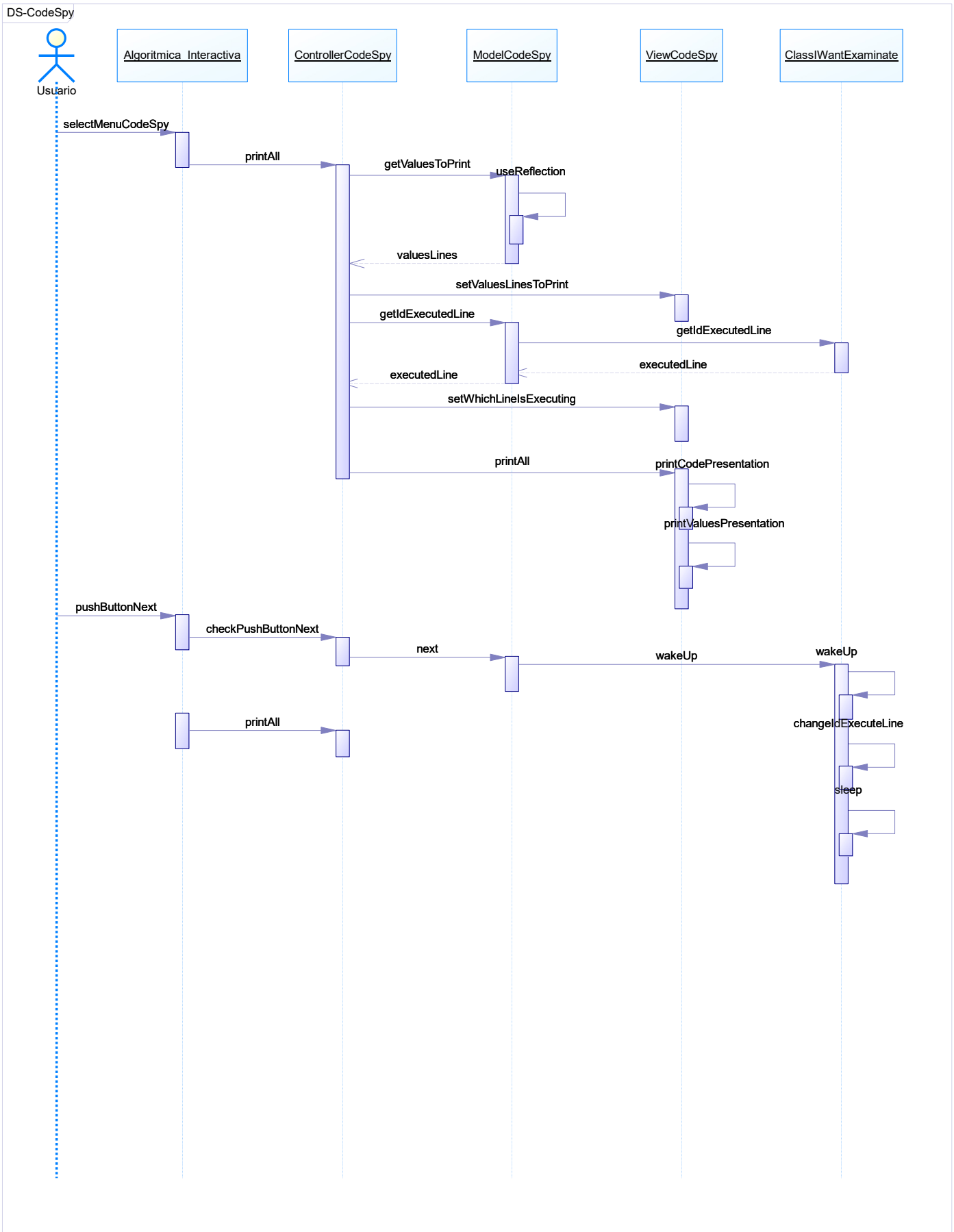

PresentationArrays
- almacen : Map
- array : Object[][]
- x : float
- y : float
- x2 : float
- y2 : float
- newX : float
+ getArray (String identificador, Object[] obj, float x, float y, Map almacen) : PresentaArray
+ setArray (Object[] obj) : void
+ setCoord (float x, float y) : void
+ getObjectArray () : Object[]
+ getCoordenadas () : Float []
+ imprimir () : void
+ getCooredenadasXYCentrar ((Object obj, float x) : float
0..*
0..1
ViewCodeSpy
- linesToPrint : ArrayList<String>
- whichLineIsExecuting : int
- positions : int [][]
- beginOfLine : int
- valueslinesToPrint : String[][]
- arrayValuesToPrint; : HashMap <String ,Object []>
- linesToPrintArray : String[][]
- btnNext : GButton
- btnCompile : GButton
- textArea : int
+ setLinePositions (int [][] positions) : void
+ setLineToPrint ((ArrayList lines, int beginOfLine) : void
+ setLineToPrintArray (String [] lines, int beginOfLine) : void
+ getLinesToPrint () : ArrayList
+ setWhichLineIsExecuting (int whichLineIs Executing)) : void
+ setValueslinesToPrint (String[] valueslinesToPrint) : void
+ setArrayValueslinesToPrint (HashMap <Str ing ,Object []> arrayValuesToPrint)) : void
+ printValuesPresentation () : void
+ printArrays () : void
+ setMessageException (String exceptionMes sage)) : void
+ showTextArea (GTextArea textArea, , GButton buttonNext, GButton buttonCompile) : void
+ setLinesInTextArea () : void
+ getTextTextarea () : String []
+ editTextarea (String text) : void
+ pushEditButton () : void
+ pushCompileButton () : void
+ pushSaveButton () : void
+ pushNextButton () : void
1

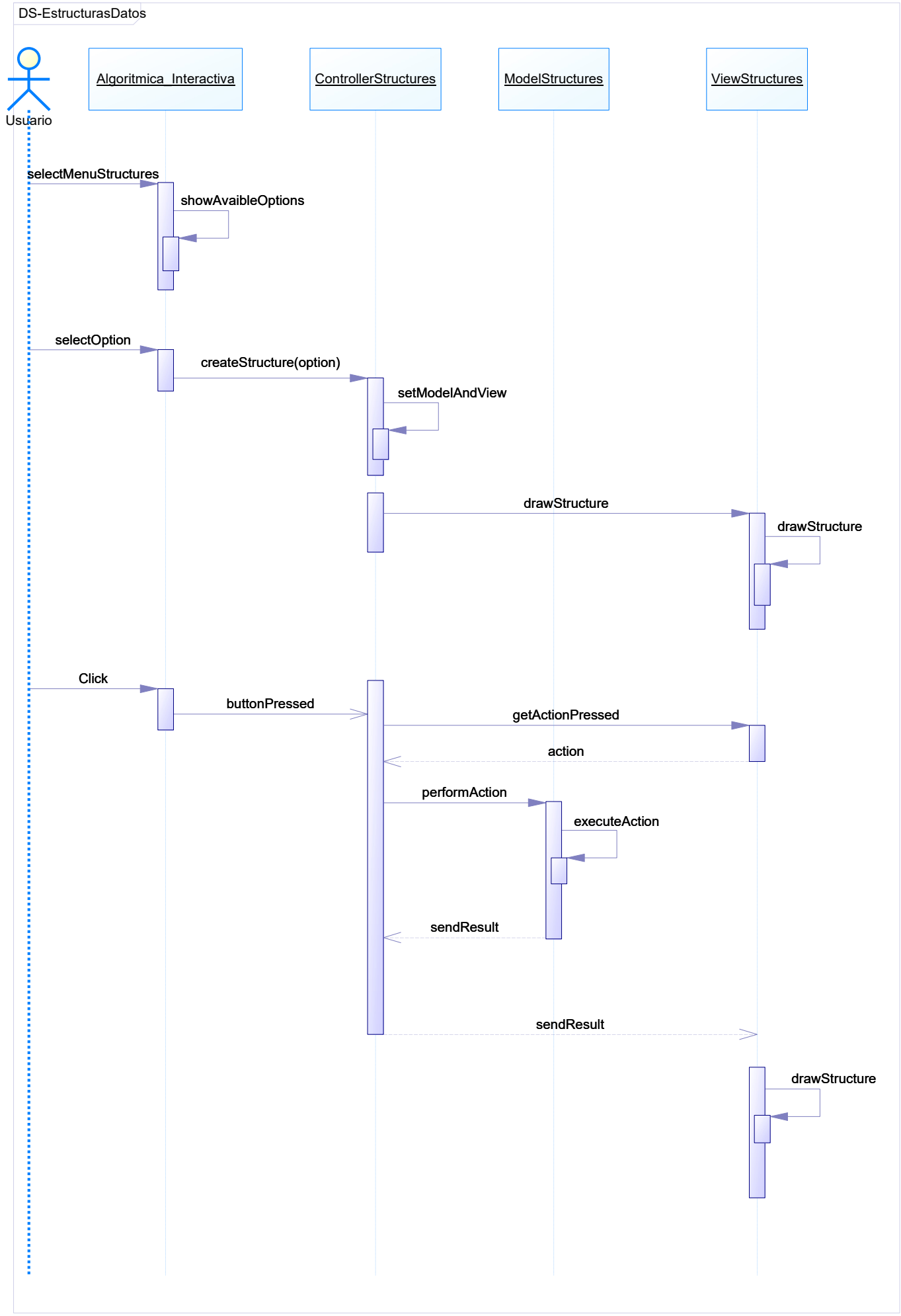

1
ControllerCodeSpy
- modelCodeSpy; : ModelCodeSpy
- viewCodeSpy : ViewCodeSpy
- linesStores : HashMap<String, ArrayList>
- thereAreException : boolean
- messageException= : String
+ printAll () : void
- sendLinesToPrint () : void
- getCodeLines () : ArrayList int
- getCodeLineArray () : String []
- getWhichLineIsExecuting () : int
- getWhoIsExecutingNow () : String
+ setModelAndView (ModelCodeSpy modelCode Spy, ViewCodeSpy viewCodeSpy)) : void
- getMain () : void
- sendValueslinesToPrint () : void
- getArrayValuesToPrint () : HashMap <String ,Object []>
- getValuesToPrint () : String[]
- setLinePositions () : void
+ checkPushButtonNext (int x, int y) : void
+ pushNextButton () : void
+ pushSaveButton () : void
+ showTextArea () : void
+ pushEditButton () : void
+ pushCompileButton () : void
1
1
1
1
ModelCodeSpy
- linesStores : HashMap<String, ArrayList>
- valuesLines : String []
- arrayValuesLines : HashMap <String ,Object [] >
- allTheDeclaredVariables; : Field[]
- objetoClassIWantExaminate : ClassIWantExaminate
+ executeMainMethodOfExaminatedClass () : void
+ getLinesStores () : HashMap
- prepareLines () : void
+ getAllUserMethods () : String
+ getGlobalsVariable () : String
+ getImports () : String
+ getMain () : ArrayList
+ getValuesToPrint () : String[]
+ getArrayValuesLines () : HashMap <String ,Object []>
+ next () : void
+ getIdExecuteLine () : int
+ getWhoIsExecutingNow () : String
- getLengthAllDeclaredFields () : int
- prepareReflection () : void
- useReflection () : void
- isAReservedTFGAttribute (Field declaredV ariables) : boolean
+ saveCodeToExaminateSAVE (String [] text) : void
+ getAllCodeToExaminateEDIT () : String
+ readAndSeparateFiles () : void
+ crearClassFile () : void
+ getMainMethodLines () : String
+ getUserAttributesLines () : String
+ getAllLinesUserMethods () : String
- setTokensControlMain (String [] lines) : void
- isAFinishedCommnad (String line) : void
+ getTokenWhoIsExecuting () : String
+ setTokensControlMethods (String[] line, int start, int last) : void
- insertMethodToControlExecution () : String
1


1
1
ClassIWantExaminate
- TFGnextTFG : boolean
- TFGidExecuteLineTFG : int
- whichOne : HashMap<String,Integer>
- whoIsExecutingNow : String
+ ejecutar () : void
+ sleep () : void
+ wakeUp () : void
+ getIdExecuteLine () : int
+ getWhoIsExecutingNow () : String
1
Algoritmica_Interactiva
- nextButton : GButton
- compilebutton : GButton
- editButton : GButton
- queueButton : GButton
- stackButton, : GButton
- listButton; : GButton
- textArea : GTextArea
- controllerTutorial; : ControllerTutorial
- controllerStructures; : ControllerDataStructures
- widthScreen : int
- heightScreen : int
- backgroundColor : color
- lettersColor : color
- lettersSize : int
- letterSizeEditor : int
- MENU : int
- SPY : int
- STRUCTURES : int
- showMeOnScreen : int
- TUTORIAL : int
- thereAreAnError : boolean
+ setup () : void
+ draw () : void
+ mousePressed () : void
+ printGoBackTotheMenu () : void
+ checkPushActionDataStructures () : void
+ showMenu () : int
+ checkPushButtons () : int
+ checkPushButtonNext () : int
+ checkPushLink () : int
+ changeScreenCodeSpy () : int
+ showCodeSpy () : int
+ showDataStructures () : int
+ showAvaibleOption () : int
+ executeMainMethodOfExaminatedClass () : int
+ initAll () : int
+ initCodeSpy () : int
+ getInitialSetting () : int
+ setSizeScreen () : int
+ setLettersColor () : int
+ setLettersSize () : int
+ getAllSettings () : int
+ handleButtonEvents (GButton button, GEvent event) : int
+ setColorBackground () : int
1
1
1
ControllerTutorial
- viewTutorial : ViewTutorial
- lines : String []
+ getLines () : String[]
+ printInfoTutorial () : void
+ readExternalFile () : void
+ checkPushLink (int x, int y) : void
1
1
ViewTutorial
- infoTutorial : String[][]
- linkY : int
- linkX : int
+ setInfoTutorial (String [] infoTutorial) :
+ printInfoTutorial () :
+ checkPushLink (int x, int y) :
1


1
ControllerStructures
- modelStructures : ModelDataStructures
- viewStructures : ViewDataStructures
+ setModelAndView (ModelDataStructures modelStructures, ViewDataStructures viewStructures) : void
+ createStructure (String structure) : void
+ buttonPressed (int x, int y) : void
+ printStructuresAnimation () : void
+ performedMethod (String action) : void
1
0..1
ModelDataStructures
+ performAction (String action) : String
+ getSize () : int
+ getClassType () : String
+ getObjectsIDs () : String[]
1
0..1
ModelQueueStructures
- Queue : Queue
- typeOfQueue : String
+ performAction (String action)
+ getSize ()
+ getClassType ()
+ getObjectsIDs ()
ModelStackStructures
- stack : Stack
+ performAction (String action) :
+ getSize () :
+ getClassType () :
+ getObjectsIDs () :
ModelListStructures
- list : List
- typeOfList : String
+ performAction (String action)
+ getSize ()
+ getClassType ()
+ getObjectsIDs ()
ViewDataStructures
+ drawStructure () : void
+ mousePressed (int x, int y) : void
+ performAction (String action, String result) : void
+ setSize (int size) : void
+ getSize () : int
+ setClassType (String classType) : void
+ setIdsObjects (String [] ids) : void
+ getButtonPressed (int x, int y) : String
ViewStructuresQueue
- idObjects : String[][]
- actionReturn : String
- size : int
- classType : String
+ drawStructure () : void
+ mousePressed (int x, int y) : void
+ performAction (String action, String result) : void
+ setSize (int size) : void
+ getSize () : int
+ setClassType (String classType) : void
+ setIdsObjects (String [] ids) : void
+ getButtonPressed (int x, int y) : String
ViewStructuresStack
- idObjects : String[][]
- actionReturn : String
- size : int
- classType : String
+ drawStructure ()
+ mousePressed (int x, int y)
+ performAction (String action, String result)
+ setSize (int size)
+ getSize ()
+ setClassType (String classType)
+ setIdsObjects (String [] ids)
+ getButtonPressed (int x, int y)
ViewStructuresList
- actionReturn : String
- sizeQueue : int
- classType : String
- coodinatesAddButtonX : int
- coodinatesAddButtonY : int
- coodinatesClearButtonX : int
- coodinatesClearButtonY : int
- coodinatesIsEmptyButtonX : int
- coodinatesIsEmptyButtonY : int
- coodinatesIteratorButtonX : int
- coodinatesIteratorButtonY : int
- coodinatesListIteratorButtonX : int
- coodinatesRemoveButtonY : int
- coodinatesToArrayButtonX : int
- coodinatesListIteratorButtonY : int
- coodinatesRemoveButtonX : int
+ drawStructure () : void
+ mousePressed (int x, int y) : void
+ performAction (String action, String result) : void
+ setSize (int size) : void
+ getSize () : int
+ setClassType (String classType) : void
+ setIdsObjects (String [] ids) : void
+ getButtonPressed (int x, int y) : String


DS tutorial
Usuario
Algoritmica_iInteractiva
ControllerTutorial
ViewTutorial
SelectMenuTutorial
printInfoTutorial
printInfoTutorial
printInfoTutorial


DS-CodeSpy
Usuario
Algoritmica_Interactiva
ControllerCodeSpy
ModelCodeSpy
ViewCodeSpy
ClassIWantExaminate
selectMenuCodeSpy
printAll
getValuesToPrint
useReflection
valuesLines
setValuesLinesToPrint
getIdExecutedLine
getIdExecutedLine
executedLine
executedLine
setWhichLineIsExecuting
printAll
printCodePresentation
printValuesPresentation
pushButtonNext
checkPushButtonNext
next
wakeUp
wakeUp
printAll
changeIdExecuteLine
sleep


DS-EstructurasDatos
Usuario
Algoritmica_Interactiva
ControllerStructures
ModelStructures
ViewStructures
selectMenuStructures
showAvaibleOptions
selectOption
createStructure(option)
setModelAndView
drawStructure
drawStructure
Click
buttonPressed
getActionPressed
action
performAction
executeAction
sendResult
sendResult
drawStructure