Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

DISEÑO Y DESARROLLO DE UN REPOSITORIO DE EJERCICIOS AUTO-CORREGIBLES

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: CARLOS MARTÍNEZ PONS

Tutor: JOSEP FRANCESC SILVA GALIANA

2020/2021

Resumen

El equipo de profesores de la Escuela Técnica de Ingeniería Informática (ETSINF) cuenta con una aplicación llamada ASys la cual permite la corrección automatizada de ejercicios y exámenes de programación para el lenguaje Java.

Para la realización de dichos ejercicios también poseen un software llamado CPS, el cual les agiliza la tarea de crear unas plantillas de corrección para los ejercicios o exámenes a evaluar mediante ASys.

En este proyecto se diseña y desarrolla una taxonomía para que, de una forma secuencial, una persona sin conocimientos en el ámbito de la programación pueda aprender de forma práctica conceptos de programación con el lenguaje Java. La taxonomía clasifica las diferentes características que debe aprender un programador de Java; y se lleva a la práctica implementando un repositorio con 50 ejercicios auto-corregibles, mediante la aplicación ASys, para que los alumnos tengan un recurso más a la hora de enfrentarse al aprendizaje de la programación. En dichos ejercicios se incluyen los conceptos Java que se imparten, en su mayoría, durante los dos primeros cursos del Grado en Ingeniería Informática en la UPV.

Palabras clave: Programación, Java, taxonomía, ejercicios, corrección automática, ASys, CPS.

Abstract

The team of teachers at the Technical School of Computer Engineering (ETSINF) has an application called ASys that allows for the automated correction of Java exercises and programming exams.

To build these exercises they also have a software called CPS, which streamlines the task of creating correction templates for the exercises or exams to be evaluated using ASys.

In this project we design and develop a taxonomy so that, in a sequential way, a person with no knowledge in the field of programming can learn programming concepts with Java language in a practical way. The taxonomy provides a classification of characteristics that any Java programmer must learn. It has been implemented producing a repository with 50 self-assessing exercises, through the ASys application, so that students have one more resource when facing learning programming. In these exercises, the practice of Java concepts will be carried out, which will be taught, mostly, during the first two years of the Degree in Computer Engineering from the UPV.

Keywords: Programming, Java, taxonomy, exercises, automatic assessment, ASys, CPS.

Tabla de contenidos

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. Introducción

Durante las últimas décadas las tecnologías de la información han tenido un auge exponencial teniendo una importancia fundamental en el mundo actual. Debido a esta evolución, la demanda en todos los campos de profesionales en estos campos es enorme, por lo que la formación en dichas tecnologías es fundamental.

Este crecimiento y la creciente demanda de plazas en el grado de Ingeniería Informática ha supuesto un incremento en la variedad de ejercicios y exámenes que los alumnos deben superar para adquirir los conocimientos deseados por el programa de estudios.

A consecuencia de estos hechos, se realizó el desarrollo de unas aplicaciones con dos objetivos:

I. Uno de ellos ayudar a los docentes a corregir tanto los ejercicios como los exámenes de una forma más automatizada y homogénea.
II. El otro objetivo es dotar a los alumnos de un recurso más, para facilitar el aprendizaje de la programación.

Dichas aplicaciones son las siguientes: ASys (acrónimo de *AssessmentSystem*) y CPS (*CodePropertiesSpecification*).

ASys permite la evaluación de código no restringido; es decir, el usuario puede usar libremente el lenguaje de programación completo sin restricciones. ASys permite la evaluación tanto del resultado final como de procesos intermedios todo depende de las características que el docente quiera evaluar. Es decir, puede centrarse en que un ejercicio deba devolver un resultado y comprobar si dicho resultado es el mismo sin tener en cuenta los procesos realizados para la obtención de dicho resultado. A su vez también puede evaluar el interior de dicha ejecución; pueden comprobarse las clases que se han utilizado, interfaces, métodos, atributos…y en definitiva cualquier característica que permita el lenguaje de programación, en este caso Java.

Para que ASys permita la corrección de los ejercicios se debían diseñar e implementar unas plantillas de corrección, donde el docente decidía qué características evaluar. Las plantillas se creaban mediante un lenguaje específico de dominio (DSL). Este proceso era muy complejo y costoso por lo que se desarrolló una aplicación para agilizar y facilitar la creación de dichas plantillas. El programa en cuestión es CPS.

CPS usa una interfaz amigable para la realización de una tarea dificultosa. En dicho programa,se carga el fichero o conjunto de ficheros de los que consta la solución del ejercicio o examen. El programa analiza el código y proporciona una interfaz para seleccionar las características del lenguaje que se desean evaluar. A su vez, también se permite elegir el coste de cada tarea; *por ejemplo, si quiero evaluar que el alumno ha declarado un método correctamente el cual puntúa con 0.5,esto puede especificarse.* Una vez terminados todos los criterios evaluables, se generan las plantillas de corrección. Con las plantillas ubicadas en la estructura organizativa correcta del ejercicio, ASys permite la evaluación automática.

Con estos dos programas el profesorado posee las herramientas para agilizar y homogenizar la corrección de los ejercicios y exámenes al alumnado.

Para poder entregarles la herramienta a los alumnos, es necesario el análisis y desarrollo de un repositorio de ejercicios. Basado en una taxonomía que les permita de forma autónoma y practica, adquirir la comprensión de los conceptos impartidos en el grado de Ingeniería Informática.

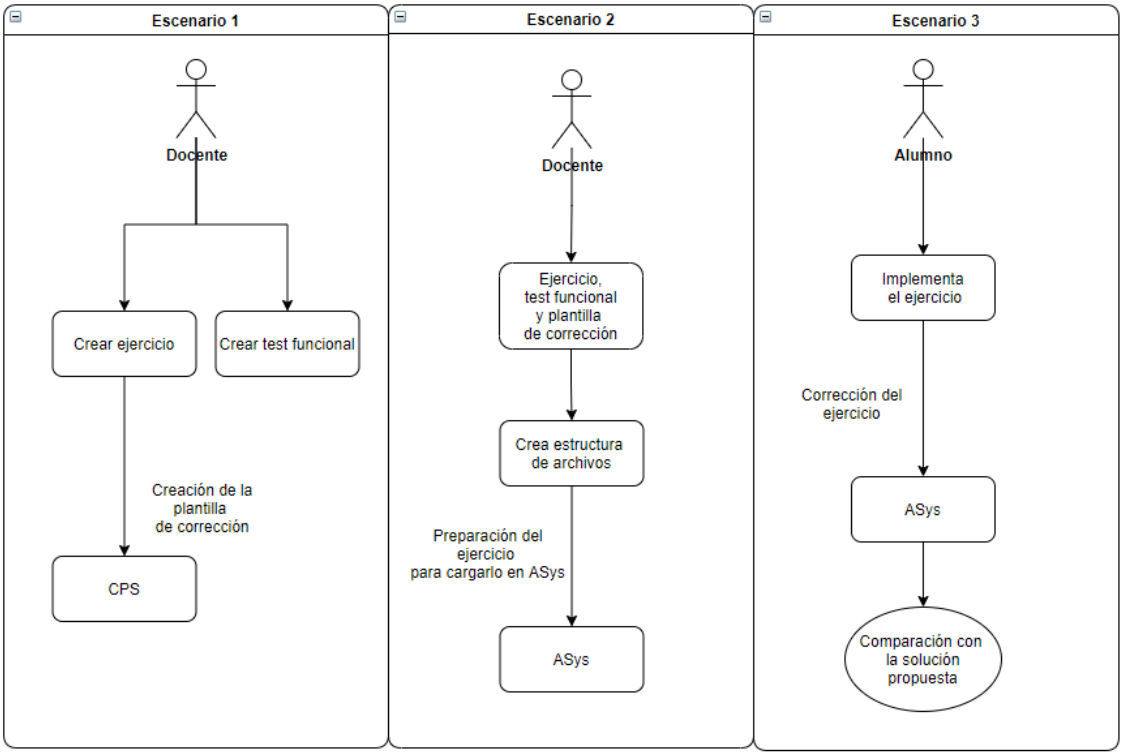

Ilustración 1 - Diagrama de casos de uso

2. Objetivo

El principal objetivo de este proyecto es el desarrollo de un repositorio de ejercicios, basado en una taxonomía que permita a los alumnos asentar los conocimientos impartidos en las asignaturas de programación pertenecientes al grado en Ingeniería Informática.

La idea de este repositorio surge de la demanda de ejercicios para la herramienta ASys. Dado que ASys ya se ha utilizado, los alumnos han encontrado carencias a la hora de tener un abanico de ejercicios a su disposición de manera que estos les ayude a familiarizarse con los conceptos de programación que les han impartido.

A su vez, este repositorio pretende cubrir otra serie de necesidades reales que han ido surgiendo durante el transcurso de los años en la docencia.

Primero de todo, hay que tener claro que este repositorio, no es un repositorio avanzado de las características del lenguaje de programación (en este caso Java). Este repositorio está pensado para ser usado por un alumno que acaba de entrar en la Universidad. Como premisa, se debe aceptar que este alumno no posee ningún conocimiento del mundo de la programación. Por lo que, en esta tesitura es conveniente reflejar que una de las materias más complicadas al empezar es la programación ya que el desconocimiento en esta área suele ser total.

Después de realizar las oportunas clases teóricas de la asignatura, el alumno puede haber entendido a la perfección lo que el docente le ha explicado, pero cambia mucho la tesitura de una explicación teórica a cuando él se enfrenta por primera vez contra la parte practica.

En el actual sistema, cuando un alumno ha terminado la lección de un concepto nuevo, para afianzarlo puede repasar la lección, intentar hacer los ejercicios de ejemplo que les han enseñado, buscar algunos ejercicios para hacer, sin la certeza de si la forma en que está procediendo es la correcta.

A su vez, puede llegar el momento en el que se quede atascado y no sepa cómo realizar una parte del ejercicio. Ante este problema solamente puede seguir investigando con la incertidumbre de saber si estará bien o no, enviar un mail al profesor y esperar hasta que pueda atenderle para despejar las dudas o comunicarse con sus compañeros y confiar en que lo que ellos hayan realizado sea la forma correcta.

Como se puede ver, hay formas de proceder, ya que es lo que se ha hecho hasta ahora y ha funcionado en mayor o menor medida dependiendo del individuo en cuestión. Pero hay dificultades, en el caso de los docentes, pueden no disponer del suficiente tiempo para atender todas las solicitudes de tutorías presenciales o responder a los múltiples correos que puedan recibir. En el caso de los alumnos, si un docente no dispone de tiempo para atenderle, puede que el alumno no consiga resolver las dudas de una forma inmediata.

Lo que se pretende con este repositorio es incidir en esos puntos. El alumno dispondrá de un método de aprendizaje practico, basado en el programa docente impartido por la UPV, donde afianzar de forma autónoma los conceptos. Siendo una solución a las 2 problemáticas comentadas.

En primer lugar una forma clara de practicar las lecciones: el alumno se enfrentará a un ejercicio, hará sus investigaciones y pensará una forma de desarrollarlo. Llegado el punto en que no supiera cómo seguir o lo finalizará y quisiera corregirlo, puede usar el programa ASys para su autocorrección. Además de mostrar la nota obtenida por su desarrollo, también visualizara la solución del ejercicio propuesta por el profesor, de esta forma el alumno puede compararlo y analizar la solución. Si lo deseara, puede volver a realizar el ejercicio para ver si ha comprendido el concepto correctamente.

Por otra parte, al tener esta herramienta, el alumno puede evitar el realizar preguntas masivas a un docente, ya que le quedan más claros ciertos conocimientos, por lo que al disminuir el número de preguntas y tener más claros algunos puntos de las lecciones, se vuelven unas preguntas más especificas y el docente dispone de más tiempo para poder profundizar y aplicarse mejor dándole así a los alumnos una mejor formación.

Como se ha comentado, este repositorio no pretende ser una herramienta para perfeccionar los conocimientos ya latentes de programadores junior sino asentar las lecciones impartidas, estos 50 ejercicios en los que se basa este proyecto, se centrarán en las características del lenguaje de programación que se imparte en la UPV (Java), y cubrirá conceptos de Java que se ven durante la mitad del grado, en concreto, durante 1º y 2º de grado.

Como todo proyecto, este tiene sus procesos de desarrollo que se describirán usando el esquema clásico del ciclo de vida en cascada:

- Especificación de Requisitos
- Análisis
- Diseño
- Implementación
- Pruebas

3. Especificación de requisitos

3.1Introducción

La especificación de requisitos va a representar las condiciones detallas que debe cumplir el proyecto. En este caso, al no tratarse de un desarrollo convencional de software como puede ser la creación de una aplicación, diferirá en algunos puntos de los requisitos de software convencionales.

Las especificaciones de este proyecto provienen del cliente, en este caso de la UPV. Se ha realizado un análisis de las expectativas y necesidades del cliente, así como de los límites del desarrollo y de las herramientas que se complementan con el proyecto.

3.2 Ámbito del proyecto

El cliente tiene desarrolladas dos herramientas que combinándolas pueden proporcionar a los alumnos un medio de autoaprendizaje con la realización de ejercicios autocorregibles que les permita ver el origen de sus errores y una solución a estos, así como la nota obtenida. Por otro lado, también es una herramienta pensada para la realización de los exámenes. De cara a los docentes, las herramientas les proporcionan una forma de preparar tanto los ejercicios como los exámenes para los alumnos, de una forma más automatizada y una forma de corregir mucho más rápida y homogénea.

La primera herramienta, ASys[Insa& Silva, 2018]es una aplicación de evaluación de código. La aplicación permite la evaluación del código tanto de las propiedades usadas como del resultado final; en otras palabras, el espectro en el que permite evaluar la aplicación (tanto ejercicios como exámenes) es extremadamente amplio.

La segunda herramienta es CPS (*CodePropertiesSpecification*), esta herramienta se usa como complemento de ASys, ya que, desde la propia aplicación de ASys se permite la creación de ejercicios nuevos, así como de configurar los diferentes elementos evaluables y su respectiva nota. Por otro lado, la realización de esta tarea desde ASys era bastante compleja; y debido a ello se utilizaba de forma esporádica. CPS surge de esa necesidad, es una aplicación de escritorio que dada una interfaz grafica al cargar el directorio o fichero con el código a evaluar, muestra una serie de selectores más amigables para configurar la plantilla de corrección y dotarlos de una nota para que después únicamente debas mover la plantilla que se crea al lugar correspondiente en el proyecto a evaluar para que ASys la reconozca y utilice.

Para clarificar los ejemplos tanto de ASys como de CPS, se añaden unas figuras donde puede verse la interfaz gráfica que va a ser necesaria utilizar para la realización de este proyecto.

Ilustración 2 - Interfaz ASys

La utilización de la interfaz se realiza de la siguiente forma: en primer lugar se selecciona "Mark exercises" → "Load exercise…", se selecciona el proyecto a evaluar y se visualiza la interfaz que tenemos en la "Ilustración 2".

En cuanto a la finalidad de los botones es la siguiente:

- *Open Instructions* - Nos abre el PDF que contiene las instrucciones de la tarea.
- *Continue Marking* - Evalúa el ejercicio. Si tuviera algún error, se abriría una nueva interfaz detallando el error. También muestra la solución de la tarea y un menú para incluir una explicación y modificar la nota que se le resta si se desea.
- *Open Resources* - Abre una carpeta con los elementos (en caso de que fueran necesarios) que se proporcionan para la realización de la tarea.
- *View report* - Se muestran los ejercicios ya evaluados y su nota obtenida.
- *Generate test cases* - No se utiliza para este proyecto por lo que no es relevante.
- *Exit exercise* - Se cierra el ejercicio.

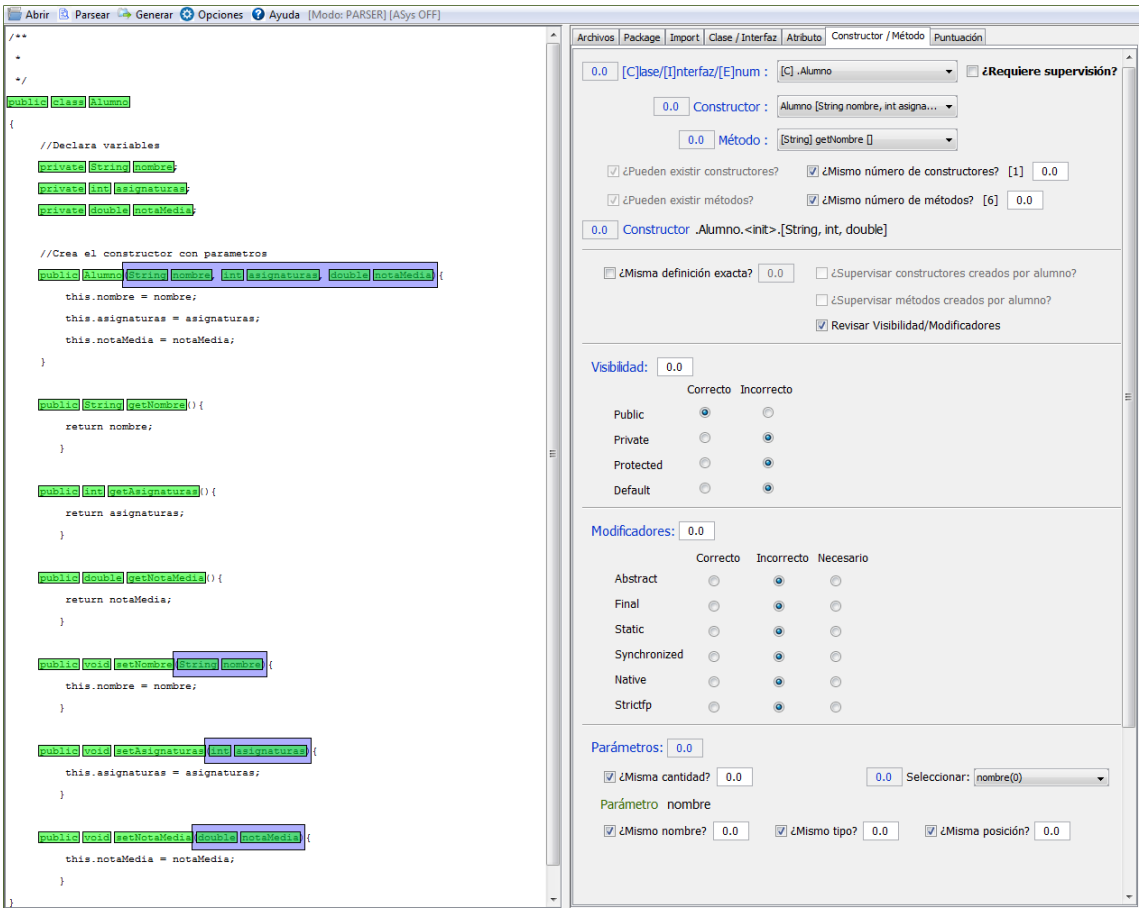

Ilustración 3 - Interfaz CPS

La utilización de la interfaz se realiza de la siguiente forma, en primer lugar se selecciona "Abrir" → "Abrir fichero..."/"Abrir directorio..." se selecciona el proyecto a evaluar y se visualiza la interfaz que tenemos en la "Ilustración3".

Una vez abierta la interfaz se accede a la pestaña "Parsear" → "Parsear todo" y veremos los elementos evaluables remarcados en el código en verde como vemos en la figura.

Esos elementos son a los que se les puede adjudicar que se revisen en la evaluación y con qué restricciones. Tenemos las diferentes pestañas para evaluar todas las características del código:

- Package
- Import
- Clases / Interfaz
- Atributos
- Constructor / Método

En cada pestaña se seleccionan los elementos a evaluar: visibilidad, modificadores, tipo de parámetro, cantidad de parámetros, tipo de interfaz..., se otorga la nota a cada función y en la última pestaña "Puntuación" se puede ver la totalidad de la nota sumando todos los componentes.

Al terminar, se selecciona la pestaña de la aplicación "Generar" → "Plantilla de corrección" y se habrá creado la plantilla que se usará en ASys.

Al tener una herramienta de evaluación (ASys) y otra que le permite crear las plantillas de corrección (CPS) que empleará ASys para evaluar los ejercicios, para la utilización de estas herramientas en la docencia, se necesita un estudio del actual programa educativo referente a las asignaturas de programación, y una propuesta de un repositorio de ejercicios donde se pondrán a prueba los conocimientos adquiridos durante los dos primeros años.

Se requiere que el repositorio posea ejercicios desde un nivel de programación nulo hasta los conocimientos que se dan por sentado en un alumno que va a comenzar 3º de carrera.

Se ha decidido optar con una primera versión de un repositorio de 50 ejercicios, en los cuales podemos diferenciar 2 bloques. El primero de ellos contendrá 20 ejercicios en los que se practicarán y evaluarán los conocimientos basados en la introducción a la programación. El segundo bloque compuesto por los 30 ejercicios restantes, se centrará en las características estudiadas principalmente en el 2º año de carrera.

Se contemplará también la creación de una tabla basada en la taxonomía empleada en el repositorio, donde se mostrará el nombre de los ejercicios y las características de la programación que evaluará dicho ejercicio con un valor numérico que exprese la dificultad del ejercicio para dicha propiedad.

3.3 Definiciones, acrónimos y abreviaturas

- **ASys:** AssessmentSystem. Aplicación capaz de diseñar plantillas de corrección y de corregir ejercicios de forma automática, comparando el ejercicio con una solución, basados en el lenguaje Java.
- **CPS:** CodePropertiesSpecification. Aplicación para la creación de plantillas de corrección con una interfaz grafica más intuitiva para la utilización del programa ASys en la corrección de ejercicios.
- **ETSINF:** EscolaTècnica Superior d'EnginyeriaInformàtica.
- **UPV:** UniversitatPolitècnica de València.
- **Java:** Lenguaje de programación orientado a objetos

3.4 Descripción general

Para la realización del repositorio de ejercicios se deben seguir unas pautas. Un estudio donde se investigue las características de la programación orientada a objetos impartidas en los dos primeros años de carrera. Y que los ejercicios posean una progresividad de forma que los iniciales estarían orientados a un alumno sin conocimiento de la materia.

Claro está, un alumno no debe realizar todo el repositorio empezando desde el primero. Si un alumno ya posee cierta experiencia y quiere por ejemplo practicar el polimorfismo, le bastará con consultar la tabla taxonómica donde muestra las características de la programación que se ejercitan con el repositorio, ver los diferentes ejercicios que practican dicha propiedad y realizarlo.

Una vez realizado el estudio se procederá a realizar los ejercicios. Se debe proporcionar tanto la solución del ejercicio, como el punto de partida (si lo hubiera) desde donde el alumno debe comenzar a realizar la tarea; además de un PDF con las instrucciones donde se explicará lo que debe realizar el alumno. Además se deberán añadir unos archivos de configuración y una estructura de directorios para el correcto funcionamiento de dicho ejercicio en ASys como autocorregible.

La estructura que debe seguir todo ejercicio del repositorio es la siguiente:

Ilustración 4 - Estructura ejercicio: carpeta inicial

Ilustración 5 - Estructura ejercicio: interior carpeta inicial

La carpeta inicial como se ve en la "Ilustración 4" en este caso es el "Ejercicio 1".Este es el punto de partida en su interior desglosamos 3 subcarpetas.

- **Exercise**: Contiene todo lo referente a la solución y los archivos de configuración para la autocorrección.
- **Instructions**: Contiene el PDF con la explicación de la tarea y los archivos (si los hubiera) que se le proporcionan al alumno como punto de partida.
- **Project**: Contiene la solución del alumno para evaluarla, y también se generará la corrección de dicha solución una vez se haya ejecutado en ASys.

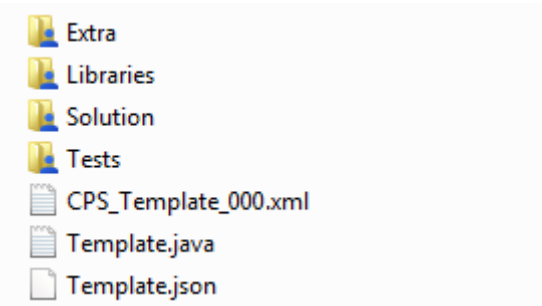

Ilustración 6 - Estructura ejercicio: interior carpeta "Exercise"

Siguiendo la siguiente ruta (Ejercicio 1 →Exercise) encontramos los siguientes apartados:

- **Extra**: En este directorio se incorporarán documentos adicionales que deban incorporarse en el ejercicio para su correcta ejecución. Si los hubiera.
- **Libraries**: En este directorio se incorporarán las librerías externas que deban usarse para la correcta ejecución del ejercicio. Si las hubiera.
- **Solution**: En este directorio se deben encontrar todas las clases Java que componen la solución del ejercicio.
- **Tests**: Todos los ejercicios deben tener tests funcionales que aseguren el correcto funcionamiento de la solución y de los ejercicios a evaluar, en este directorio se encuentran dichos test funcionales.
- **CPS_Template_000.xml[mkyong,2021]&Template.java&Template.json**: Estos tres ficheros se crean al ejecutar la herramienta CPS y determinan qué elementos se deben tener en cuenta para la evaluación generando así las plantillas de corrección.

Como dato adicional, una vez se tienen todos estos ficheros, se ha generado la plantilla y se poseen los test funcionales; para que ASys los tenga en cuenta se debe modificar el fichero "Template.java" añadiendo las propiedades necesarias para la evaluación de dichos test funcionales y sus métodos (en el caso de que hubiera varios). A su vez, a cada propiedad se le asignará un valor numérico que hace referencia a la nota.

```
55          properties.add(new Object[]{ "P28","Functionality",null,new String[]{},"Alumno.java","Test Functional","Test Case",3.0});
56          return properties;
57      }
58  /*****************PROPERTIES TEST SECTION ***********************************/
59      public boolean checkP28(){
60      return true;
61  }
62
63      public boolean solveP28(){
64      return false;
65  }
66
67  public String[] testP28(){
68      final String testCaseName = "Test";
69          final String testMethodName = "test1";
70          final Boolean expectedResult = true;
71          final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName);
72
73          if (expectedResult.equals(testCaseResult)){
74              return null;
75          }else{
76              return new String[] { testCaseName, testMethodName };
77          }
78  }
79  /*******************END PROPERTIES TEST SECTION*********************************/
```

Ilustración 7 - Ejemplo de inserción de test funcional en Template.java

En la "Ilustración 7" se puede ver un ejemplo de inserción de test funcional en el Template.java. En el caso del testP28, se puede ver que el nombre de la clase es "Test", el método a evaluar "test1" y el resultado esperado "true", si no se cumpliera daría un error en dicho método.

En la primera fila vemos cómo se añade la propiedad y cómo el valor asignado a dicha propiedad es de 3.0.

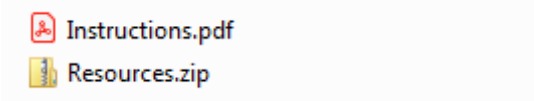

Ilustración 8 - Estructura ejercicio: interior carpeta "Instructions"

Siguiendo la siguiente ruta (Ejercicio 1 →Instructions) encontramos los siguientes apartados:

- **Instructions.pdf**: Este documento contiene la explicación de toda la tarea (los puntos que la componen y la nota de cada apartado) para que el alumno la realice.
- **Resources.zip:** Este fichero comprimido contiene el estado inicial del proyecto con todos los ficheros del cual debe partir el alumno para su desarrollo.

Ilustración 9 - Estructura ejercicio: interior carpeta "Projects"

Siguiendo la siguiente ruta (Ejercicio 1 →Projects) encontramos los siguientes apartados:

- **Assessed**: En esta carpeta se encuentran las soluciones de los alumnos una vez han sido evaluadas por ASys, con los documentos que indican la nota final y los fallos que ha obtenido.
- **Handovers**: En esta carpeta se introducen las carpetas con las soluciones de los alumnos para que ASys las pueda evaluar.

3.5 Requerimientos específicos

3.5.1 Interfaces externas

En un desarrollo de software estándar, como una aplicación tanto de escritorio como web también se tendría en cuenta el desarrollo de una interfaz de usuario, tanto gráfica, como interfaz de hardware y software.

En este caso especifico, al tratarse de diseñar ejercicios para realizar con unas aplicaciones ya existentes, la interfaz gráfica nos viene dada por ellas. Se trata de una interfaz intuitiva y sencilla que nos facilita su uso.

A nivel de interfaz de hardware, es una aplicación de escritorio, por lo que a la utilización de elementos hardware que se requiere son únicamente el ratón, teclado y una pantalla que permite resolución de 1920x1080p para facilitar la accesibilidad a personas con diversidad funcional.

A nivel de interfaz de software, tanto las aplicaciones como los ejercicios están desarrollados en Java, por lo que cualquier sistema (Windows, Linux, OSX...) que tenga instalada una versión de la API de Java 8 o superior será suficiente para su ejecución o en el caso de los ejercicios su desarrollo.

3.5.2 Propiedades

Al tratarse de un proyecto con fines didácticos para la UPV. Se debe tener en cuenta a la hora de implementar los ejercicios un nivel de dificultad moderado y el uso de librerías o conceptos de Java y de programación que se hayan podido ver en el transcurso del Grado en Ingeniería Informática.

Hay que tener en cuenta que aunque a un alumno de último curso se le debe considerar con los conocimientos suficientes para poder entrar en el mundo laboral y

de tener una autonomía para la investigación de los problemas y la resolución de estos de forma propia, este proyecto se centrará en la primera parte del grado, recurriendo así en los conceptos de programación y de Java dados durante los dos primeros cursos. De ahí las restricciones antes mencionadas.

4. Análisis

Para la realización de un desarrollo de software siempre es necesario realizar un análisis y estudiar toda la complejidad del problema antes de abordarlo.

En esta etapa nos encargamos de construir un cuerpo robusto, y de planificar, preparar y pulir las estrategias que vamos a seguir en su posterior implementación. De esta forma, podemos detectar carencias o posibles problemas que vayamos a encontrarnos durante el avance del proyecto y preparar unas soluciones para optimizar el tiempo de desarrollo y poder cumplir con los objetivos pactados.

El enfoque de este proyecto es la creación de una taxonomía que permita el autoaprendizaje de los conceptos de programación estudiados en la UPV, podemos subdividir esta sección en 3 subsecciones.

- Definición
- Estado del Arte
- Taxonomía propuesta.

4.1 Definición

El primer objetivo que tenemos que tener claro es ¿qué es una taxonomía?

La búsqueda de información acerca de la palabra "taxonomía" obtiene, principalmente, datos relativos a la biología ya que es un término recurrente en dicho campo. Un ejemplo de definición que puede encontrarse es el siguiente:

> *La **taxonomía** (del griego ταξις, taxis, "ordenamiento", y νομος, nomos, "norma" o "regla") es, en su sentido más general, la ciencia de la clasificación. Usualmente se emplea el término para designar a la **taxonomía biológica**, la ciencia de ordenar a los organismos en un sistema de clasificación compuesto por una jerarquía de taxones anidados. La taxonomía biológica es una subdisciplina de la biología sistemática, que estudia las relaciones de parentesco entre los organismos y su historia evolutiva. Actualmente, la taxonomía actúa después de haberse resuelto el árbol filogenético de los organismos estudiados, esto es, una vez que están resueltos los clados, o ramas evolutivas, en función de las relaciones de parentesco entre ellos.[wikipedia, 2021b]*

Si bien esa es una definición aceptada del término, es específica de la taxonomía biológica, que no es la que nos atañe en este proyecto. Para este proyecto una definición más adecuada sería la siguiente:

> *La taxonomía es la ciencia que estudia los principios, métodos y fines de la clasificación.[significados, 2021]*

Y esta definición aplicada al sistema educativo que es de lo que trata el proyecto nos situaría en una **taxonomía en teoría del aprendizaje**.

4.2 Estado del arte

Una vez situados dentro del ámbito educativo, el uso de la taxonomía se ve reflejado dentro del diseño instruccional. Cada proceso educativo tendrá asignada una planificación. Para que esto suceda, se necesita dejar claramente definidos los objetivos de aprendizaje, es decir, lo que el alumno podrá realizar al finalizar dicho proceso.

Una vez concretados los objetivos de aprendizaje, estará establecida la dificultad que tendrá el curso, por lo que se podrá proceder a definir los procedimientos, técnicas e instrumentos que se emplearán durante el proceso del aprendizaje. Por consiguiente, se puede decir que dichos objetivos marcarán la estructura planificada de nuestro curso.

Es necesario hacer una distinción entre estos dos términos ***intención educativa*** *y* ***objetivo de aprendizaje***, el primero hace referencia a la orientación de las acciones que durante el curso se implementarán con el propósito de que el alumnado asimile los conocimientos que se les proporcionan. El segundo se refiere al resultado final que se pretende conseguir mediante el programa educativo, es decir, lo que se espera que el alumnado sea capaz de hacer.

Por todo ello, una taxonomía se utilizará para la comprobación del nivel de ejecución en el que se conseguirán los objetivos de aprendizaje. A causa de que no siempre tenemos las mismas condiciones de aprendizaje, se necesita agrupar los diferentes tipos de resultados que se desean en el proceso de aprendizaje. Se pueden dividir en las categorías siguientes:

- **Habilidades motrices:** habilidad que otorga la ejecución de movimientos mediante actos motrices estructurados reflejando vigor, precisión, rapidez o uniformidad del movimiento corporal.
- **Actitudes:** se componen de estados mentales internos que contribuyen en las diferentes posibilidades de respuesta ante acciones especificas a través de elecciones de actos personales.
- **Estrategias cognoscitivas:** son las cualidades que nos posibilitan la regulación de los procesos internos de la persona, permitiéndole conducir su propio pensamiento, memorización y atención. La capacidad para resolver problemas, el uso correcto de reglas y conceptos se hallan en esta categoría.

Teniendo en cuenta esto podemos enfocarnos en ***cuatro taxonomías*****[**crec, 2021**]**que suelen utilizarse frecuentemente dentro del ámbito educativo, las cuales se van a explicar de forma concisa.

Taxonomía de Bloom

La taxonomía de objetivos de la educación también conocida como la taxonomía de Bloom, se basa en clasificar una acción educativa por objetivos y se utilizan como punto de partida para el diseño de objetivos de aprendizaje.

El psicólogo educativo Benjamin Bloom 1913-1999 [Bloom. B, 1971] propuso esta taxonomía que se clasifica en función de 3 dimensiones para alcanzar los objetivos.

Dominio cognitivo: Los objetivos de este nivel se centran en torno ala comprensión y el conocimiento de cualquier tema dado, la habilidad de pensar, dividiéndose en seis niveles "conocer, comprender, aplicar, analizar, sintetizar, evaluar".

Dominio afectivo: Los objetivos de este nivel se enfocan en la conciencia y crecimiento en emoción, sentimientos y actitud, habilidades de sentir dolor o alegría por otro ser, dividiéndose en cinco niveles "recepción, respuesta, valoración, organización, caracterización".

Dominio psicomotor: Los objetivos de este nivel se enfocan en el cambio desarrollado en las habilidades o en la conducta, pericia en el manejo físico de herramientas, dividiéndose en seis niveles "percepción, disposición, mecanismo, respuesta compleja, adaptación, creación".

La taxonomía de objetivos de la educación es de las más conocidas y de las más utilizadas desde su creación en el mundo de la educación. Es una herramienta principal orientada a la creación de objetivos dentro del modelo de enseñanza-aprendizaje y está fundamentada en la idea de que no todos los resultados deben ser iguales. Por ello, se ordenan y se categorizan las habilidades del pensamiento en función del aprendizaje final que se desea conseguir.

En los últimos años se han llevado a cabo grandes progresos tecnológicos en la investigación del cerebro, y la aparición de la neurociencia cognitiva ha hecho que se conozcan nuevos datos y por lo tanto que esta taxonomía quede obsoleta conforme fue fundamentada.

Taxonomía de Marzano

La nueva taxonomía de los Objetivos Educativos o la taxonomía de Marzano elaborada por John S.Kendall y Robert J. Marzano[Marzano, R.J. & Kendall, J.S, 2007] toma su base de la famosa taxonomía de objetivos de la educación o taxonomía de Bloom.

En el planteamiento de Kendall y Marzano se realizo una revisión del modelo clásico, actualizándolo e incorporando nuevas ideas sobre cómo es el procesamiento de la información en el ser humano.

Esta taxonomía fue elaborada entre los años 2007 y 2008.Debido al paso del tiempo y los avances tras décadas de investigación en el área y mejorando el conocimiento sobre cómo funciona el aprendizaje en los seres humanos, el modelo de Bloom se quedo obsoleto y se necesitó una reformulación.

Aunque Bloom pretendía crear un sistema práctico de clasificación de objetivos educativos, acabó siendo más bien teórico, teniendo un impacto reducido sobre el currículo escolar y en cómo debería ser la elaboración de este. Por esta razón, Marzano y Kendell decidieron construir un modelo más práctico, centrado en mejorar el sistema educativo. Su método de clasificación es más aprovechable, permitiendo a los docentes adaptar sus enseñanzas a las demandas y necesidades de su alumnado.

La taxonomía está distribuida en dos dimensiones, las cuales interactúan entre sí:

Niveles de procesamiento:

1. **Nivel cognitivo:** Se encuentra la información adquirida recientemente, la que todavía tenemos en un plano consciente, se puede dividir en 4 grupos: "recuperación, comprensión, análisis y uso del conocimiento".
2. **Nivel meta cognitivo:** Se aplican los conocimientos adquiridos recientemente para regular los procesos mentales propios, es decir, el pensamiento en base a lo que se está aprendiendo y comprender de qué manera se dirige el proceso de aprendizaje.
3. **Nivel interno:** Sucede cuando las creencias del individuo son afectadas cambiando sus conocimientos previos o haciéndole reflexionar, ya sea sembrando la duda o expandiéndolos mediante la adquisición de un nuevo conocimiento.

Dominios del conocimiento:

1. **Información:** Es la adquisición de datos puros. La memorización de datos sin necesidad de ningún razonamiento.
2. **Procedimientos mentales:** Todos aquellos conocimientos que necesiten el pensamiento, es decir, que para alcanzar un objetivo sea necesario el seguimiento de una serie de pasos.
3. **Procedimientos psicomotores:** Todo dominio y coordinación física mediante el uso del cuerpo. Por ejemplo actividad deportiva, o aprendizajes de tipo manual como tocar un instrumento o escribir.

Los dominios del conocimiento y los niveles de procesamiento interactúan entre sí, ya que para todo conocimiento nuevo independientemente del dominio por el que llegue, debe interactuar con el nivel cognitivo puesto que en algún momento del aprendizaje del individuo ese conocimiento es algo recién adquirido.

Habrá otro momento en el que el individuo regule su comportamiento para perfeccionarlo o se marque unos objetivos, pasando al plano meta cognitivo.

Ya sea realizar ejercicio físico, aplicar formulas teóricas o aprender un idioma, todo conocimiento influye sobre el sistema de creencias del individuo, pasando al nivel interno.

De esta taxonomía obtenemos la importancia que adquiere el autoconocimiento para la correcta apropiación de los conocimientos y su directriz al aprendizaje visto como una posibilidad de cambio.

Taxonomía de Guilford

La inteligencia ha sido conceptualizada de diferentes formas a lo largo de la historia.

Se han generado una enorme cantidad de modelos y teorías que tratan de explicar el concepto de inteligencia, cómo actúa y cómo se estructura. Desde tratarla como una capacidad única y unitaria, considerarla una agrupación de capacidades independientes entre sí o estipular la existencia de conjuntos de capacidades organizadas jerárquicamente son algunos de los ejemplos que han surgido.

A modo genérico, podemos establecer que se considera de la habilidad o conjunto de habilidades mentales que posibilitan nuestra adaptación, concediéndonos la gestión de nuestros recursos cognitivos de la manera más competente a la hora de afrontar diversas situaciones.

Mediante ella, tenemos la habilidad de captar y analizar de forma correcta la información, establecer estrategias, planificar que conducta tomar y ponerla a prueba.

Un psicólogo estadounidense llamado Joy Paul Guilford[Guilford, JP, 1967]tuvo grandes aportaciones al estudio de la inteligencia. Todas sus aportaciones llegaron a confluir en una teoría de la inteligencia que supuso uno de los modelos más relevantes en cuanto al entendimiento de la inteligencia como una agrupación de habilidades.

En su taxonomía, Guildford, consideró la inteligencia como el método mediante el cual el ser humano era capaz de procesar la información del medio y conducirla a contenidos mentales, proporcionándole una visión operativista de la información.

Considera las distintas aptitudes como no jerarquizadas, es decir, considera las diferentes capacidades independientes. La concepción de inteligencia sería un conjunto de capacidades independientes entre sí que permiten adaptarnos al medio.

Su modelo suele representarse de manera cubica donde las interacciones se interrelacionan entre las tres dimensiones dándole forma hasta 120 factores distintos.

Contenidos: Conjuntos de informaciones y datos recabados, sin que se hayan realizado ningún tipo de trabajo sobre ellos.

- Figural → datos visuales que percibimos, imágenes
- Conductal → información mediante interacciones con otros individuos, gestos, expresiones, intenciones.
- Simbólica → información mediante signos indicativos que por sí mismos no tienen significado, lenguaje de signos
- Semántica → obtención de significados y su relación con los símbolos representados principalmente en el lenguaje verbal

Operaciones Mentales: La transformación de la información que se está recibiendo mediante un proceso intelectual en un producto de salida, ya sea en forma física o mental.

- Cognición → Entendimiento de la información.

- Memoria → Almacenamiento de información con el fin de utilizarla en un momento posterior.
- Producción convergente → Crear alternativas a través de la información recopilada.
- Producción divergente → Crear distintas alternativas a las habituales y almacenarlas en la memoria
- Evaluación → Comparaciones entre los distintos conceptos que permiten establecer relaciones

Productos: Resultado de las transformaciones realizadas mediante las operaciones a los contenidos. Expresión o respuesta generadas ya sean bien mental o conductual.

- Unidades → respuestas simples y básicas
- Clases → conceptualizaciones u organizaciones de unidades semejantes
- Relaciones → idea de una conexión entre las distintas informaciones
- Implicaciones → conexión entre informaciones sugeridas por algún elemento concreto sin que la conexión figure específicamente como información
- Transformaciones → cualquier alteración ocasionada en base a la información captada
- Sistemas → interacción entre sí de organizaciones de diferentes informaciones

La incorporación del pensamiento divergente como elemento distintivo fue una de las principales aportaciones de Guilford a la concepción de la inteligencia. Anteriormente, no se tenía en cuenta la creación de alternativas fuera indicio de inteligencia.

Mediante este modelo propuesto por Guilford se empezó a enfocarse en el estudio de la creatividad como aspecto de la capacidad intelectual.

Taxonomía de Camperos

La taxonomía de Camperos fue propuesta por Mercedes Camperos [Camperos. M, 1992].Su modelo consta de tres métodos de aprendizaje:

Aprendizajes Reproductivos: Está compuesto por las preguntas o requerimientos de los instrumentos evaluativos y de los objetivos instruccionales que demandan al estudiante reconocer y recordar el conocimiento que ha sido memorizado y adquirido. Se demanda que reproduzca o repita el aprendizaje no que aporte nuevas alternativas.

Se establecen 2 subcategorías:

- **Evocación o reproducción de la información:** aprendizajes evaluados y objetivos instruccionales, se pide al estudiante demostrar la memorización de estructuras metodológicas o conceptuales e información en general.
- **Reconocimiento de información:** aprendizajes que exigen que el estudiante identifique el conocimiento proporcionado, es decir, la aplicación de las condiciones propuestas para las operaciones de memoria.

Aprendizajes Productivos: Está compuesto por los requerimientos formulados en los eventos evaluativos y los objetivos instruccionales, que solicitan del estudiante completos procesos de recombinación, integración de información y contenidos para la creación de un comportamiento que pudiese o no estar ligado a principios, reglas, estructuras y generalizaciones universales, pudiendo o no haberse aprendido anteriormente.

También supone la propuesta activa del estudiante, pues estos aprendizajes necesitan que el estudiante muestre y ponga en evidencia parte de su inventiva.

En este modelo se conjugan las operaciones correspondientes a las producciones divergentes y convergentes del modelo de Guilford y la categorización de las capacidades intelectuales de Bloom. Se delimitaron 2 subcategorías:

- **Producciones convergentes:** aprendizajes que necesitan crear un producto o información vinculada coherentemente en forma imperativa a una información o situación preexistente. Ello supone la creación inventiva y activa del estudiante bajo cánones preestablecidos adquiridos en la enseñanza.
- **Producciones divergentes:** requerimientos y objetivos evaluativos, que demandan del estudiante la generación de información a partir de una información proporcionada, creación no ligada a patrones o pautas de imperativo lógico, si no que exige variedad y amplitud de respuestas. Requiere la utilización del aprendizaje memorizado, capacidad de transferencia para reorganizar y estructurar nuevas informaciones creando opciones reales. Este aprendizaje no se basa en pautas que se han definido por patrones rígidos. El creador deja elementos distintivos a modo de sello en cada producto divergente.

Producciones evaluativas: Son aquellos aprendizajes mediante los cuales derivan o producen una información valorativa que conduce a un juicio crítico y el

asentamiento de una decisión relativa al criterio empleado para emitir y juzgar el mismo.

Implica la presencia de dos informaciones:

- La información que podrá ser valorada y sometida a juicio.
- La información juzgada en base a los criterios.

La producción evaluativa se genera mediante la relación latente entre estas dos informaciones, es decir, la generación de nueva información que es lo que finalmente consideramos como producciones evaluativas.

En esta categoría se encuentran ubicados los objetivos instruccionales y las exigencias de las herramientas de evaluación que solicitan al estudiante un juico de merito, calidad, valor, conveniencia, pertinencia, adecuación, etc. de situaciones determinadas en base a criterios que pudiesen ser implícitos o explicitaos y cuyo producto suponga la generación de juicios decisorios sobre dicha situación.

4.3 Taxonomía orientada a la programación en Java

El modelo taxonómico que se propone se basa en los conceptos recogidos de las taxonomías expuestas en el estado del arte, pero de una forma más especializada. No se busca clasificar el aprendizaje general si no que nos centramos en la disciplina de la programación en Java.

Todos los modelos expuestos anteriormente están planteados de una forma genérica donde, siguiendo sus respectivos métodos, puedes desarrollar un programa docente orientado al aprendizaje del alumnado.

En este modelo no desarrollaremos un programa docente puesto que el proyecto está orientado a compaginarse con el impartido en la Universidad Politécnica de Valencia. Cabe recalcar que este proyecto ha sido escrito en 2021 por lo que se tiene en cuenta el programa docente impartido en la UPV durante los años previos, por lo tanto, si este programa sufriera cambios en un futuro habría que ajustar el modelo para que se pudiera complementar.

El modelo está enfocado a la parte práctica, es decir, a poner a prueba la parte teórica aprendida en clase. Se ha diseñado este progreso tanto de características del lenguaje como de Java situándonos en la posición de un alumno que no ha tenido contacto alguno anteriormente con ningún aspecto de la programación.

Si bien es cierto que en el primer año de universidad se pueden encontrar alumnos que ya han experimentado con la programación, ya sea por haber cursado otra carrera y haber dado los fundamentos básicos, por haber realizado algún curso o por haber investigado por iniciativa propia; la gran mayoría son alumnos que han accedido a la universidad por primera vez sin ningún conocimiento.

Los ejercicios son de conocimientos progresivos, es decir, el primer ejercicio será el más fácil y el último contendrá las características más avanzadas. Se establece que para la realización del ejercicio 20 ya se ha adquirido el conocimiento de los 19 previos. Es por tanto una metodología incremental. Los ejercicios se han diseñado con el objetivo de asentar los conocimientos, practicándolos desde diferentes perspectivas. Por ejemplo, se pretende enseñar el uso de los bucles desde su concepto más primitivo como sería recorrer con un "*for*" un vector de números enteros, así como ir avanzando con el recorrido de diferentes tipos, inclusive las listas o colecciones de clases propias e incluso tipos/clases genéricos; a su vez intentar que el alumno consiga practicar los diferentes tipos de bucles (*for, foreach, while, do-while*).

Mediante este enfoque de orientar varios ejercicios para un mismo concepto desde diferentes perspectivas, permite asentar los conocimientos y comprender las utilidades de la característica practicada.

Como punto más significativo de este modelo tenemos el autoaprendizaje. En otros modelos, en general, se suele necesitar a un instructor que ayude en los procesos. Aunque se practique lo expuesto en clase, se necesita que alguien compruebe que lo que se ha realizado sea correcto. Con este método de autocorrección no se depende de nadie; se pueden afrontar los ejercicios de forma autónoma y comprobar con la

herramienta en qué se ha fallado y cuál es la forma correcta de resolverlo. Por lo tanto puedes volver a enfrentarte a ellos y ver si has afianzado los conocimientos.

En las siguientes tablas se observan las características del lenguaje que evalúa cada ejercicio así como su dificultad. La dificultad está catalogada de más fácil a más difícil, siendo un 1 la más sencilla y un 5 la más complicada. En cada ejercicio se evaluarán diferentes características y cada característica tendrá una valoración de dificultad pudiendo así un mismo ejercicio siendo de una dificultad de 2 para condicionales y de 3 para bucles.

La dificultad ha sido establecida en la posición de un alumno nuevo, de forma que si por ejemplo un alumno de último curso accediera a realizar estos ejercicios debería encontrar la gran mayoría sencillos de resolver.

Modelo

Se ha realizado un estudio basado en el programa docente de la UPV donde se han clasificado las principales características del lenguaje de Java[Riley D, 1951;Natividad P et al., 2012; Weiss, M.A, 2013; Sharan K , 2014].

Clases	If/Switch	Interfaces
Objetos	For/While	Genericidad
Constructores	For each	Listas
Parámetros	Enum	ArrayList
Métodos	Array	Pilas
Librerías	Excepciones	Colas
Attr./Variable	Recursividad	Árboles
Constantes	Herencia	HashMap
Modificadores	Polimorfismo	Threads
E/S Datos	Cl. Abstractas	

Tabla 1 - Características Java del modelo

Una vez analizadas las características del lenguaje se han determinado diferentes niveles que facilitan el aprendizaje. Como ejemplificación, no se debe intentar enseñar el uso de un bucle *'for'* recorriendo un *'array'*, si no se ha explicado lo que es un '*array*', tampoco puede enseñarse la utilización de un condicional '*if*' si no se ha explicado lo que son los atributos ni las variables. Se debe seguir un proceso lineal, primero se deben explicar qué son los atributos, qué son las variables, cómo se construye un constructor, métodos y una vez se tiene conocimiento de esas características, se puede enseñar a realizar condicionales en la lógica de un método.

En la Ilustración10 se pueden ver las dependencias de las características en el modelo propuesto.

Hasta el nodo 1, observamos cómo en primer lugar se debe tener conocimientos de creaciones de clases y objetos, una vez sabemos lo que son y cómo crear clases, podemos aprender los conceptos de atributos y variables. Ligado a estos conceptos introducimos los modificadores y las constantes.

Una vez el alumno entiende los conceptos del nodo 1,se introducen los constructores, getter/setter, métodos y librerías. Para la utilización de cualquiera de estos conceptos es necesaria la asimilación de los anteriores.

Desde el nodo 2 al 3 se practican los conceptos de *'if', 'switch', 'enum', 'operaciones E/S', 'excepciones', 'recursividad'* y también nos introducen los conceptos de 'for' *y 'while'* en dependencia con *'array'*, ya que en los ejercicios planteados se recorren *'arrays'* con los *'for'*, por lo que es necesario haber aprendido *'array'* anteriormente. También existe una relación de dependencia entre *'foreach' y 'for'*. Primero se comprende cómo funciona un *'for'* estándar y luego se aprende el *'foreach'.*

Hasta el nodo 3 se han practicado los conocimientos básicos de la programación, introducidos en las asignaturas de "Introducción a la informática y a la programación" (1º año, semestre A) y de "Programación" (1ª año, semestre B).

Desde el nodo 3 hasta el final aprenderemos conceptos más avanzados dependientes de los anteriormente expuestos. Algunos dados en la asignatura de "Programación" nuevamente y otros en las de "Lenguajes, Tecnologías y Paradigmas de la Programación" (2ª año, semestre A), "Estructuras de Datos y Algoritmos" (2ª año, semestre B) y "Concurrencia y Sistemas Distribuidos " (2ª año, semestre B).

Podemos visualizar en la siguiente tabla los conceptos Java que se van a practicar y en que asignatura se estudian.

IIP	PRG	LTP	EDA	CSD
Clases	Librerias	Polimorfismo	Recursividad (DyV)	Threads
Objetos	Enum	Genericidad	HashMap	
Atributos	Operaciones E/S		Listas	
Variables	Excepciones		Colas	
Constantes	Recursividad		Pilas	
Modificadores	Foreach		Árboles de búsqueda	
Constructores	Herencia			
Getter/Setter	Interfaz			
Métodos	Listas			
If/Switch	Pilas			
Array	Colas			
For/While	ArrayList			

Tabla 2 - Relación Asignaturas/Conceptos

Como se puede observar, la mayor parte del modelo se centra en el primer año, ya que son los fundamentos básicos de la programación. Posteriormente, también introducimos ejercicios de asignaturas de segundo con una estructura más compleja.

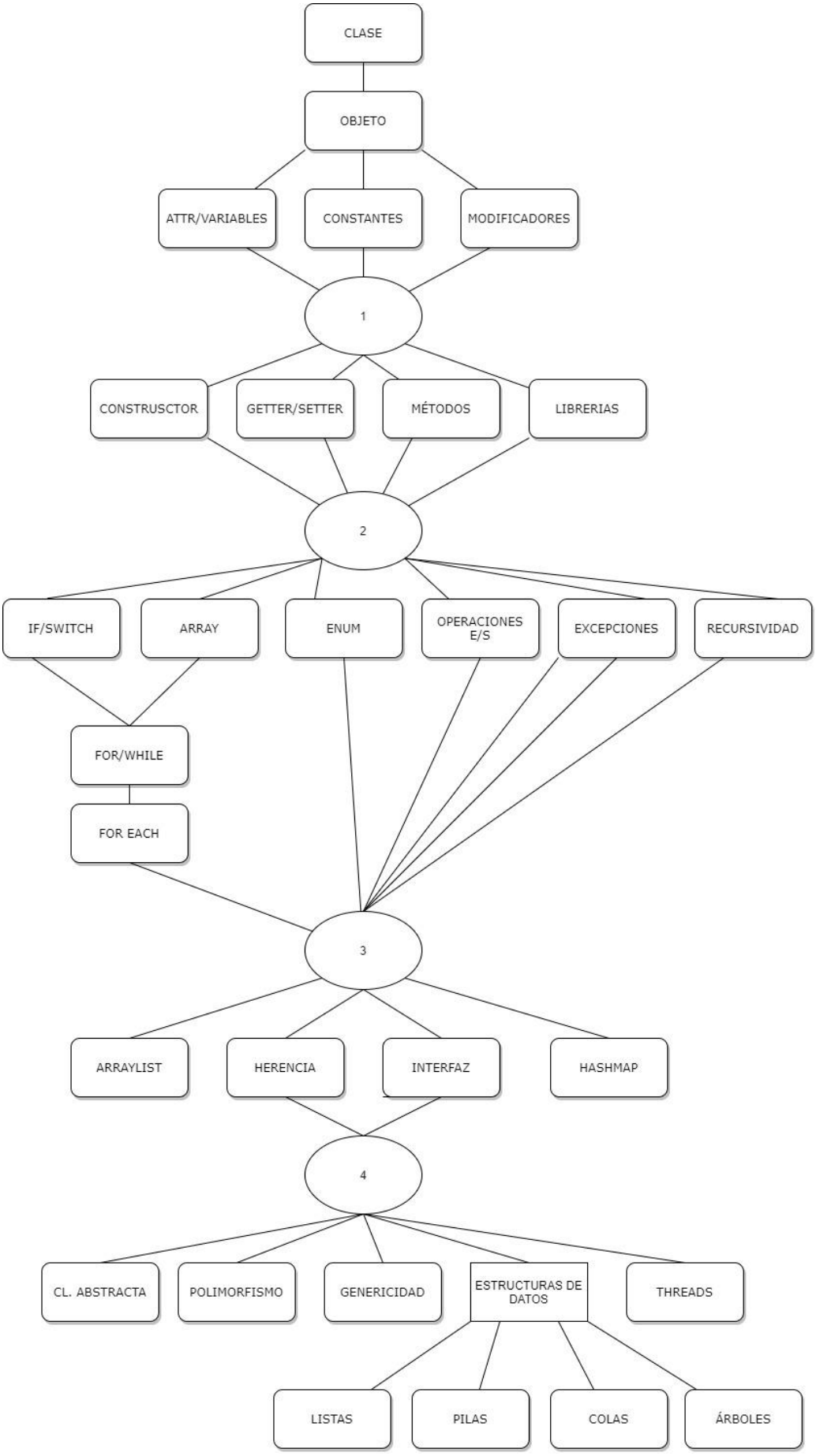
CLASE
OBJETO
ATTR/VARIABLES
CONSTANTES
MODIFICADORES
1
CONSTRUSCTOR
GETTER/SETTER
MÉTODOS
LIBRERIAS
2
IF/SWITCH
ARRAY
ENUM
OPERACIONES E/S
EXCEPCIONES
RECURSIVIDAD
FOR/WHILE
FOR EACH
3
ARRAYLIST
HERENCIA
INTERFAZ
HASHMAP
4
CL. ABSTRACTA
POLIMORFISMO
GENERICIDAD
ESTRUCTURAS DE DATOS
THREADS
LISTAS
PILAS
COLAS
ÁRBOLES

Ilustración 10 - Diagrama de aprendizaje

Ejercicios

En el apartado anterior se han introducido los conceptos a estudiar; en este apartado atribuiremos a cada ejercicio los conceptos que en él van a practicarse.

Como se ha comentado con anterioridad, la dificultad del ejercicio está catalogada con una enumeración del 1 al 5, siendo el 1 la más fácil y el 5 la más difícil.

También se ha comentado que los ejercicios son de modo progresivo. Es decir si 3 ejercicios evalúan la misma característica, su dificultad ira progresando. Como ejemplificación mostraré 3 ejemplos de dificultad con un *"for".*

- Como dificultad 1: Un ejemplo sería una iteración de "X" veces fijas (Iterar algo 100 veces)
- Como dificultad 2/3: Un ejemplo sería recorrer una estructura dinámica, como puede ser un *array, arrayList, list*...
- Como dificultad 4/5: Un ejemplo sería recorrer un sistema matricial con bucles anidados.

El sistema de puntuación se establece en relación a los conceptos que se van a practicar, con CPS podemos enumerar la puntuación que va a recibir cada declaración, cada nombre de los atributos, variables, métodos, constructores...; también podemos puntuar la declaración de clases, herencias, imports de librerías, interfaces... en definitiva todos los recursos sintácticos del código.

Por otro lado, tenemos los test funcionales, en CPS podemos determinar que un método devuelva un integer, con los test funcionales podemos llevar a cabo una prueba del funcionamiento y comprobar que un método que debe devolver el número 3 lo devuelve y por lo tanto tiene una lógica correcta.

Cada ejercicio tienes valores diferentes dependiendo de lo que se evalúa y de su enunciado, pero en general los test funcionales suelen comprender del 2 a 5 puntos de la nota.

En la tabla siguiente se observan las características a evaluar por ejercicio y su dificultad. Los primeros 20 ejercicios están centrados en los fundamentos básicos, la mayoría introducidos en la asignatura IIP. Los 30 restantes practican conceptos más avanzados que requieren haber asimilado los fundamentos básicos.

Al finalizar los 50 ejercicios se puede considerar que el alumno ha adquirido soltura y una base consistente en programación.

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Ejercicio	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Clases	1																																																		
Objetos			1	3																																														
Constructores		1	3	4																																														
Parámetros			2	3																																														
Métodos				1	3													2		3																																
Librerías					1	2									3	3		2																																	
Attr./Variable		1	3	4	3		2			2										3																														
Constantes					1																																														
Modificadores					2						4																																							
E/S Datos					2	3			3		3				3	3				4																														
If/Switch					1	3		3	2									3		3																														
For/While							2	3	3	3	4							3	4	3																														
Foreach										3	4																																						
Enum									2																																									
Array							2			3	4			2				3	4	3																														
Excepciones													2	3	3	4	5																																	

Recursividad												3							3																									4						
Herencia																					2	3	3	4		4				3																		3		
Polimorfismo																										4																			3					
Cl. Abstractas																													2	3	4																	3		
Interfaces																											2	4			4														3					
Genericidad																																2	4																4	
Listas																																		3	4	3		3												
ArrayList																									3	3							3	4																
Pilas																																				3	4													
Colas																																						3	4											
Árboles																																								3	4									
HashMap																																										3	3						4	
Threads																																														3	3			5

Tabla 3 - Relación Ejercicios/Conceptos

4.3.2 Mejoras

Existen muchas áreas de mejoras en el proyecto. Se podrían subdividir en 2 tipos:

- Contenido
- Funcionalidad

Contenido: En lo que al contenido se refiere, se podría ampliar enormemente, ya no solo con más ejercicios de cada aspecto, eso sería una mejora leve, pero sí lo que es introducir características de los cursos posteriores.

Tenemos que tener en cuenta que nos encontramos en un ámbito académico, no obstante sería beneficioso contar con un apartado más profesional para trabajar con las especializaciones de cada rama, así como en el uso de los frameworks más populares.

Una vez el alumno termine su etapa académica se enfrentará al mundo laboral. En el mundo laboral tendrá que convivir con varias tecnologías de forma simultánea, así como con la utilización de diferentes frameworks. Por lo que contar con un apartado profesional para poder practicar estos rasgos en un nivel elemental sería beneficioso.

Otro punto en el contenido sería la detección y evaluación de buenas prácticas, robustez y escalabilidad del código.

Funcionalidad: Respecto a la funcionalidad de la herramienta, estaría bien introducir un apartado de teoría, donde se documenten los temas a tratar en los ejercicios. Bien podría ser información en cada ejercicio de lo que se va a ver o por bloques de ejercicio, si por ejemplo tenemos un bloque que practican los mismos conceptos, tener una documentación genérica para dicho bloque.

A raíz de este proyecto, se podrían crear diferentes herramientas o ampliar la actual para admitir más tecnologías además de Java, para poder facilitar el aprendizaje de Javascript, Haskell, C++, Python o cualquier tecnología relevante que se imparta en la carrera.

5. Diseño de la solución

En esta sección se presenta el diseño que se ha seguido para la realización del repositorio de ejercicios.

En la sección anterior *"Análisis: 4.3 Taxonomía orientada a la programación en Java"*, se ha presentado una tabla de características (*Tabla 1*) seleccionadas de Java que sirven como dimensiones del aprendizaje que queremos evaluar.

Estás características se han escogido basándonos en el estudio de Java (fundamentalmente cursos y colecciones de ejercicios de Java) y en el análisis del plan de estudios del Grado en Informática de la UPV.

Los ejercicios deben estar orientados para alumnos de la universidad, por lo que la progresividad y la dificultad se orientan a un nivel académico, es decir, cada característica se practicará en una serie de ejercicios y, en función de la que se esté practicando, se compaginará con otras características. Esto se hará intentando que el primer ejercicio que se practique corresponda a una dificultad mínima, y que vaya avanzando hasta una dificultad más compleja y que requiera de un análisis mayor del alumno para hallar la solución.

En función de la característica puede llegar a practicarse en un único ejercicio o en varios, lo habitual suele ser 3 o 4 ejercicios.

La estructura de un ejercicio autocorregible se compone de las siguientes partes:

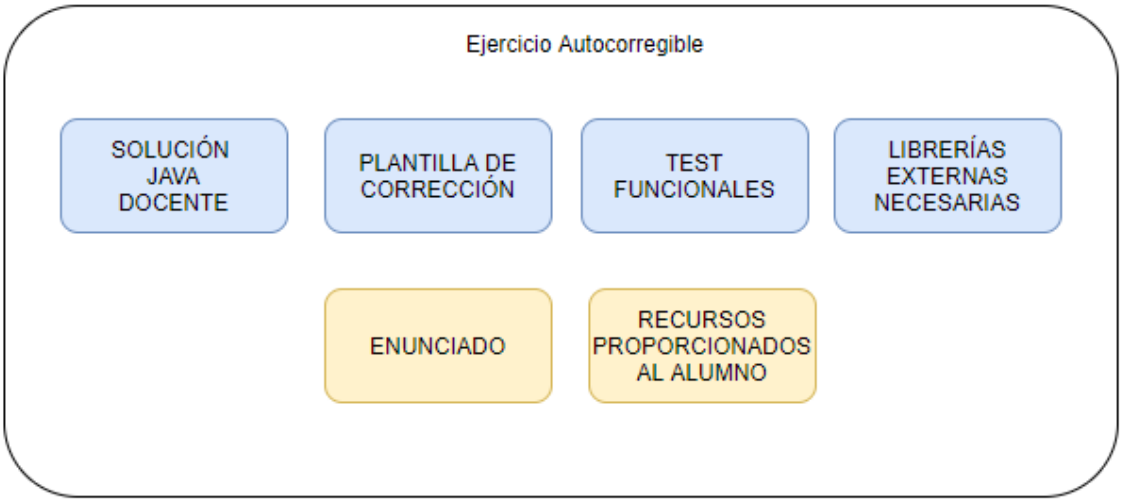

Ilustración 11 - Estructura ejercicio autocorregible

En la ilustración 11, la primera fila de bloques (con fondo azul)contiene los componentes principales para la solución del ejercicio.

En un problema tenemos:

- La solución en el lenguaje Java proporcionado por el docente.

- La plantilla de corrección del problema generado a través de CPS y como base de la solución Java.
- Los tests funcionales para comprobar que la solución del alumno funciona de la forma correcta.
- Archivos de librerías externas (en el caso de que la solución lo requiera)

En el segundo bloque (con fondo amarillo)se nos muestran las herramientas que se le proporcionan al alumno:

- Un fichero PDF con el enunciado del problema
- Y en el caso de requerirlo, un punto de partida o archivos/recursos adicionales que se necesiten para la resolución del problema.

Una vez tienes esta estructura, el ejercicio autocorregible está listo para cargarlo en el sistema ASys y que el alumno pueda trabajar en su resolución.

Concepto y usabilidad

Un ejercicio autocorregible [Insa D, Pérez S, Silva J, Tamarit S, 2020] se compone de la estructura mencionada anteriormente. La función principal es poder tener una corrección inmediata de una solución propuesta por un alumno.

Para la realización de esta corrección se utiliza la herramienta ASys. Uno de los elementos principales para determinar la evaluación del ejercicio es la plantilla de corrección.

Inicialmente, la plantilla se desarrollaba desde la propia herramienta ASys mediante un lenguaje de domino específico (DSL). La manipulación de este proceso directamente puede ser compleja, por lo que se desarrolló una segunda herramienta llamada CPS para la creación de estas plantillas a través de una interfaz gráfica más amigable.

La ejecución de la auto corrección se divide en 3 partes (Ilustración 12) :

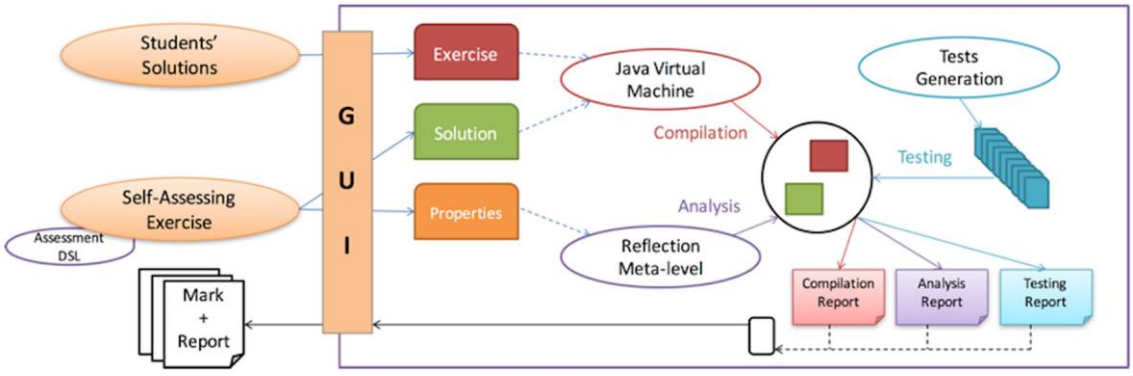

Ilustración 12 - Arquitectura ASys

- Compilación: Para identificar errores de compilación en la solución y el ejercicio del alumno.
- Analisis: El sistema analiza el codigo y verifica que se cumplen las propiedades introducidas en la plantilla de corrección de forma automática. Las propiedades se recorren una por una siguiendo un algoritmo de corrección (ver Ilustración 13).
- Testing: Para comprobar errores de ejecución. Una vez comprobado la compilación y la estructura del codigo, debe comprobarse la funcionalidad. Se implementan tests, se les asigna un valor y se pueden comprobar automaticamente para ver si la solución funciona de la forma esperada.

```
    Input: The student's solution and the template to correct it
    Output: The student's solution corrected
 1: function CORRECTSOLUTION(template, solution)
 2:     properties = getProperties(template)
 3:     code = solution
 4:     repeat
 5:         { property, code } = checkProperties(properties, code)
 6:         if property ≠ ⊥ then
 7:             code = askTeacherToCorrect(property, code)
 8:         end if
 9:     until property == ⊥
10:     return code
11: end function
12: function CHECKPROPERTIES(properties, code)
13:     for all property ∈ properties do
14:         if not check(property, code) or not test(property, code) then
15:             previousCode = code
16:             code = solve(property, code)
17:             if not check(property, code) or not test(property, code) then
18:                 return { property, previousCode }
19:             end if
20:         end if
21:     end for
22:     return { ⊥, code }
23: end function
```

Ilustración 13 - Algoritmo de corrección

Alternativamente a los tests implementados manualmente, la herramienta proporciona la posibilidad de generar casos de prueba. Estos casos de prueba son utilizados para comparar el comportamiento de la solución del profesor y del alumno(ver Ilustración 14).

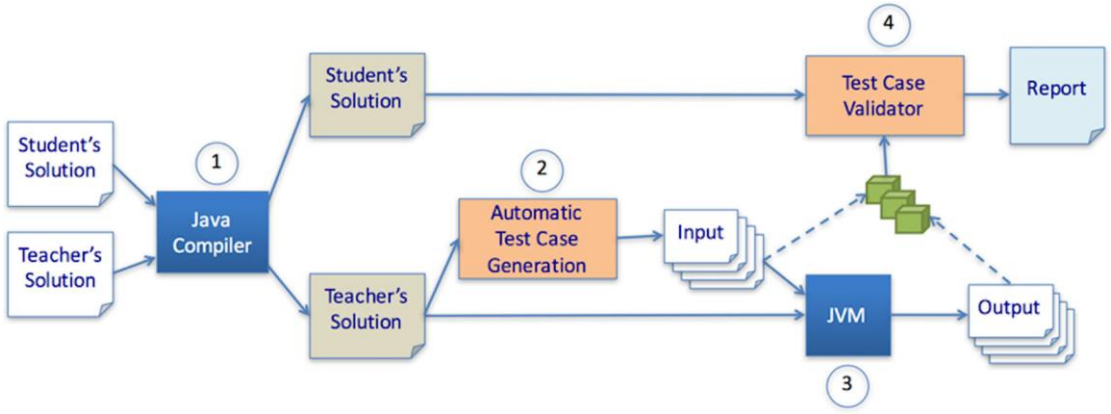

Ilustración 14 - Generación de casos de prueba

El diagrama de decisión (ver Ilustración 15) resume el proceso de autocorrección y señala cómo son construidos los reportes de error de cada paso.

Las cajas rojas (*Teacher'sSolution, StudentsSolutions and SelfAssessmentExercise*) representan los datos introducidos por los usuarios.

La primera caja gris corresponde a la generación de los casos de prueba en función del código del profesor. Las otras 3 cajas representan cada una de las fases anteriormente explicadas.

Cada fase genera un report con la información proporcionada por el profesor. Por esta razón, en cada fase se encuentra una sección "Ask the teacher" con una flecha discontinua que lleva la información a los reports y combinando todos los reports proporcionados se construye la nota final.

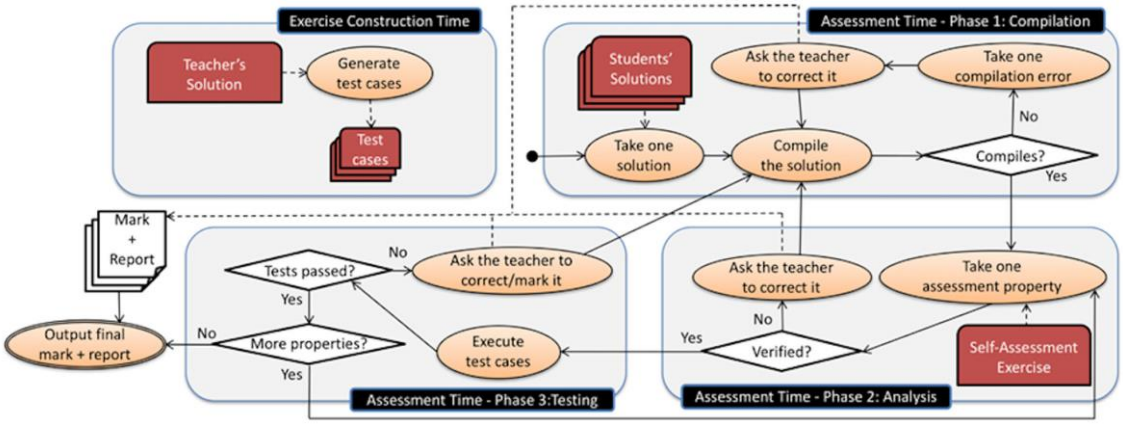

Ilustración 15 - Diagrama de decisión ASys

Las 3 fases se ejecutan de forma secuencial, pero con una particularidad. Se comienza con la fase de compilación pero únicamente avanza hacia la fase de análisis si la primera se ha ejecutado de forma correcta. Si tiene un fallo se muestra al profesor. El profesor puede decidir corregirlo anotando el fallo para tener constancia de la nota que equivale, y posteriormente arregla el fallo y ejecuta de nuevo el test, si

no tiene otro fallo en la fase de compilación pasara a la siguiente. Este proceso se repite para la fase de análisis y testing.

Este comportamiento se debe a que si el test se hiciera de una forma secuencial clásica y se ejecutará las 3 fases de una sola vez, un error en una fase podría alterar los resultados de las siguiente, y de esta forma no se realizaría una corrección correcta de cada fase.

Repositorio

Cada ejercicio perteneciente al repositorio que engloba este proyecto tiene una solución basada en la estructura y la usabilidad que se ha descrito en esta sección. De forma que el alumno pueda determinar sus propios errores, saber que nota le corresponde a cada error y corregirlos.

6. Implementación

Una vez efectuado el análisis del proyecto y establecida la tabla taxonómica, se procede a la implementación de los diferentes ejercicios que componen este repositorio.

En esta sección vamos a introducir brevemente los conceptos de la programación (en este caso en el lenguaje Java) que se trabajan, un breve resumen y en los casos necesarios imágenes con una parte o la totalidad del código de la solución.

La estructura de archivos que deben seguir los ejercicios está explicada en el apartado 3.4 de esta memoria y en el apartado 4.3 la tabla taxonómica que se ha seguido para el desarrollo del repositorio.

El repositorio consta de 50 ejercicios autocorregibles, divididos en 2 partes: la primera consta de los 20 primeros ejercicios donde se van introduciendo los conceptos iniciales de la programación. En los 30 restantes se dan por sabidos dichos conocimientos y se evalúan conceptos intermedios equivalentes en su mayoría a un 2º curso del grado en ingeniera informática de la UPV.

Los ejercicios tienen una evolución progresiva, siendo el ejercicio número 1 el más sencillo y el ejercicio 50 el más avanzado, durante la progresión de los ejercicios se dan por sentados los conocimientos de problemas previos. Es decir, el ejercicio 7 puede contener o dar por aprendidos conocimientos dados en los 6 ejercicios anteriores.

Las siguientes secciones explicarán detalladamente en qué consisten los ejercicios. Todos ellos seguirán una estructura común. Cada sección equivale a 1 ejercicio. Como se ha comentado previamente, los ejercicios parten de la base de que el alumno no tiene conocimientos iniciales sobre la materia. Estructura:

- Conceptos a practicar
- Dificultad (aquí encontraremos 5 categorías: muy fácil, fácil, medio, difícil y muy difícil. Teniendo en cuenta que el nivel de este proyecto abarca los primeros 2 años de grado en ingeniería informática)
- Descripción del problema (un breve resumen genérico de lo que se espera, para el detalle del problema debe consultarse la implementación y su correspondiente hoja de instrucciones). A continuación se adjunta un ejemplo completo a modo de ejemplo.

Ejemplo de ejercicio completo

Se muestra un ejemplo completo de un ejercicio real con todos sus componentes. Instrucciones:

Enunciado:

Se proporciona una clase "*Alumno*".

Se debe completar con las siguientes declaraciones:

- Un atributo llamado "*nombre*" de tipo cadena de caracteres, privado y otro llamado " *notaMedia*" entero de doble precisión. .(1 punto)
- Se debe crear un constructor con un parámetro "*notaMedia*" y un parámetro "*nombre*", donde inicializaremos los atributos con esos valores. (2 puntos)
- Se debe crear otro constructor con un único parámetro "*nombre*", que solo inicializará el atributo "*nombre*". (2 puntos)
- Se debe crear un método "*calcularNotaMedia*" con 3 parámetros llamados ("*nota1*", "*nota2*", "*nota3*"). A partir de esos parámetros, se debe asignar a la variable *notaMedia* el promedio de las notas dadas. (3 puntos)
- Se debe crear un método "*devolverNotaMedia*", que devuelve el valor de la variable *notaMedia*. (1 punto)
- Se debe crear los métodos *set* y *get* del atributo *nombre*. (1 punto)

Ilustración 16 - Ejemplo enunciado

Solución:

```
/**
 *      *Se debe declarar dos variables, (notaMedia y nombre) (1p)
          *Se debe construir un constructor con un parámetro nombre
          y un parámetro notaMedia donde iniciaremos las variables con esos valores. (2p)
          *Se debe construir otro constructor con un único parámetro nombre. (2p)
          *Se debe crear un método "calcularNotaMedia" con 3 parámetros llamados (nota1, nota2, nota3).
          Y se debe asignar a la variable notaMedia la media entre las notas dadas. (3p)
          *Se debe crear un método devolverNotaMedia, donde devuelve el valor de la variable notaMedia. (1p)
          *Se debe crear los métodos set y get del atributo nombre. (1p)
 */
public class Alumno
{
    private String nombre;
    private double notaMedia;

    //Crea constructor1 (nombre y notaMedia)
    public Alumno(String nombre, double notaMedia){
        this.nombre = nombre;
        this.notaMedia = notaMedia;
    }

    //Crea constructor2 (nombre)
    public Alumno(String nombre){
        this.nombre = nombre;
    }

    //CalcularNotaMedia con 3 parámetros de notas
    public void calcularNotaMedia(double nota1, double nota2, double nota3){
        this.notaMedia = (nota1 + nota2 + nota3)/3.0;
    }


    //DevolverNotaMedia
    public double devolverNotaMedia(){
        return notaMedia;
    }


    public String getNombre(){
        return nombre;
    }

    public void setNombre(String nombre){
        this.nombre= nombre;
    }


}
```

Ilustración 17 - Ejemplo solución

Tests funcionales:

```java
public class Test
{
    public static boolean test(){
        Alumno a1 = new Alumno("Pepe", 7.5);
        Alumno a2 = new Alumno("Juan");

        // Comprobamos los GETTERS
        if (!a1.getNombre().equals("Pepe") || a1.devolverNotaMedia() != 7.5
        || !a2.getNombre().equals("Juan") || a2.devolverNotaMedia() != 0.0)
        {
            return false;
        }

        // Usamos los SETTERS
        a1.setNombre("Juan2");
        a2.setNombre("Juan3");
        a1.calcularNotaMedia(7.0, 5.5, 8.5);
        a2.calcularNotaMedia(7.0, 5.5, 8.5);


        // Comprobamos los GETTERS
        if (!a1.getNombre().equals("Juan2") || a1.devolverNotaMedia() != 7.0
        || !a2.getNombre().equals("Juan3") || a2.devolverNotaMedia() != 7.0)
        {
            return false;
        }

        return true;
    }

    public static boolean test2() throws Throwable
    {
        String nombre = "Alan";
        int asignaturas = 5;
        double notaMedia = 7.4;

        final Class<?> alumnoClass = Alumno.class;

        final Field nombreField = alumnoClass.getDeclaredField("nombre");
        final Field notaMediaField = alumnoClass.getDeclaredField("notaMedia");

        nombreField.setAccessible(true);
        notaMediaField.setAccessible(true);

        final Alumno alumno = new Alumno(nombre, notaMedia);
        final String nombreValue = (String) nombreField.get(alumno);
        final double notaMediaValue = (Double) notaMediaField.get(alumno);

        final Alumno alumno2 = new Alumno(nombre);
        final String nombreValue2 = (String) nombreField.get(alumno2);
        final double notaMediaValue2 = (Double) notaMediaField.get(alumno2);

        return nombreValue == nombre && notaMediaValue == notaMedia
        && nombreValue2 == nombre && notaMediaValue2 == 0.0;
    }
}
```

Ilustración 18 - Ejemplo test funcional

Trozo de la plantilla de corrección generada mediante CPS:

```
import java.util.*;
import codeassess.javassessment.Assessor;
import codeassess.javatemplate.ast.ASTexplorer;

public class Template extends Assessor {

    ASTexplorer explorer;

    // BUILDER
    public Template() {
        explorer = new ASTexplorer();
    }

    // CONFIGURATION
    public Map<String, String> getConfig() {
        final Map<String, String> config = new Hashtable<String, String>();
        config.put("Language", "Java");
        config.put("Extensions", ".java");
        config.put("AssessmentMode", "SemiAutomatic");
        return config;
    }

    // PROPERTY SECTION
    public List<Object[]> getProperties() {
        final List<Object[]> properties = new LinkedList<Object[]>();
        properties.add(new Object[] { "ASTexplorer", "Functionality", null, new String[] {}, "ASTexplorer()", "ASTexplorer()", "ASTexplorer()", 0.0 });
            properties.add(new Object[]{ "P0","Functionality",null,new String[]{},"Alumno.java","Revisar que tiene paquete y que tiene mismo nombre         ","PackageDeclaration (line 1): La declaración del paquete es incorrecta",0.0});
        properties.add(new Object[]{ "P1","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 10): el modificador es incorrecto",0.0});
        properties.add(new Object[]{ "P2","Functionality",null,new String[]{},"Alumno.java","Revisar que la clase [sea]/[no sea]  una interface","IsInterface (line 10): la declaración de clase/interface es incorrecta    ",0.0});
        properties.add(new Object[]{ "P3","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 10): el nombre es incorrecto",0.0});
        properties.add(new Object[]{ "P4","Functionality",null,new String[]{},"Alumno.java","Revisar Size,TypeParametersList","TypeParameters (line 10): La lista de parámetros tipificados es incorrecta",0.0});
        properties.add(new Object[]{ "P5","Functionality",null,new String[]{},"Alumno.java","Revisar Size,ExtendedTypesList","ExtendedTypes (line 10): La lista de extends es incorrecta",0.0});
        properties.add(new Object[]{ "P6","Functionality",null,new String[]{},"Alumno.java","Revisar Size,ImplementedTypesList","ImplementedTypes (line 10): La lista de implements es incorrecta",0.0});
        properties.add(new Object[]{ "P7","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 12): el modificador es incorrecto",0.2});
        properties.add(new Object[]{ "P8","Functionality",null,new String[]{},"Alumno.java","Revisar Name,TypeArguments,SuperType","ClassOrInterfaceType (line 12): El tipo de clase o interface es incorrecta",0.15});
        properties.add(new Object[]{ "P9","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 12): el nombre es incorrecto",0.15});
        properties.add(new Object[]{ "P10","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 13): el modificador es incorrecto",0.2});
        properties.add(new Object[]{ "P11","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo sea primitivo","PrimitiveType (line 13): el tipo primitivo es incorrecto",0.15});
        properties.add(new Object[]{ "P12","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 13): el nombre es incorrecto",0.15});
        properties.add(new Object[]{ "P13","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 16): el modificador es incorrecto",0.15});
        properties.add(new Object[]{ "P14","Functionality",null,new String[]{},"Alumno.java","Revisar que el tamaño sea el mismo","Size (line 16): el tamaño es incorrecto",0.15});
        properties.add(new Object[]{ "P15","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 16): la posición del elemento es incorrecta",0.15});
        properties.add(new Object[]{ "P16","Functionality",null,new String[]{},"Alumno.java","Revisar Name,TypeArguments,SuperType","ClassOrInterfaceType (line 16): El tipo de clase o interface es incorrecta",0.15});
        properties.add(new Object[]{ "P17","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 16): el nombre es incorrecto",0.15});
        properties.add(new Object[]{ "P18","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 16): la posición del elemento es incorrecta",0.15});
        properties.add(new Object[]{ "P19","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo sea primitivo","PrimitiveType (line 16): el tipo primitivo es incorrecto",0.15});
        properties.add(new Object[]{ "P20","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 16): el nombre es incorrecto",0.15});
        properties.add(new Object[]{ "P21","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 22): el modificador es incorrecto",0.1});
        properties.add(new Object[]{ "P22","Functionality",null,new String[]{},"Alumno.java","Revisar que el tamaño sea el mismo","Size (line 22): el tamaño es incorrecto",0.15});
        properties.add(new Object[]{ "P23","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 22): la posición del elemento es incorrecta",0.15});
        properties.add(new Object[]{ "P24","Functionality",null,new String[]{},"Alumno.java","Revisar Name,TypeArguments,SuperType","ClassOrInterfaceType (line 22): El tipo de clase o interface es incorrecta",0.15});
        properties.add(new Object[]{ "P25","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 22): el nombre es incorrecto",0.15});
        properties.add(new Object[]{ "P26","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 27): el modificador es incorrecto",0.1});
        properties.add(new Object[]{ "P27","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo es vacio (void)","VoidType (line 27): el tipo vacio (void) es incorrecto",0.1});
        properties.add(new Object[]{ "P28","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 27): el nombre es incorrecto",0.1});
        properties.add(new Object[]{ "P29","Functionality",null,new String[]{},"Alumno.java","Revisar que el tamaño sea el mismo","Size (line 27): el tamaño es incorrecto",0.1});
```

Ilustración 19 - Ejemplo plantilla corrección 1

```
57          properties.add(new Object[]{ "P30","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 27): la posición del elemento es incorrecta",0.1});
58          properties.add(new Object[]{ "P31","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo sea primitivo","PrimitiveType (line 27): el tipo primitivo es incorrecto",0.1});
59          properties.add(new Object[]{ "P32","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 27): el nombre es incorrecto",0.1});
60          properties.add(new Object[]{ "P33","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 27): la posición del elemento es incorrecta",0.1});
61          properties.add(new Object[]{ "P34","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo sea primitivo","PrimitiveType (line 27): el tipo primitivo es incorrecto",0.1});
62          properties.add(new Object[]{ "P35","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 27): el nombre es incorrecto",0.1});
63          properties.add(new Object[]{ "P36","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 27): la posición del elemento es incorrecta",0.1});
64          properties.add(new Object[]{ "P37","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo sea primitivo","PrimitiveType (line 27): el tipo primitivo es incorrecto",0.1});
65          properties.add(new Object[]{ "P38","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 27): el nombre es incorrecto",0.1});
66          properties.add(new Object[]{ "P39","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 33): el modificador es incorrecto",0.1});
67          properties.add(new Object[]{ "P40","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo sea primitivo","PrimitiveType (line 33): el tipo primitivo es incorrecto",0.1});
68          properties.add(new Object[]{ "P41","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 33): el nombre es incorrecto",0.1});
69          properties.add(new Object[]{ "P42","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 38): el modificador es incorrecto",0.1});
70          properties.add(new Object[]{ "P43","Functionality",null,new String[]{},"Alumno.java","Revisar Name,TypeArguments,SuperType","ClassOrInterfaceType (line 38): El tipo de clase o interface es incorrecta",0.1});
71          properties.add(new Object[]{ "P44","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 38): el nombre es incorrecto",0.1});
72          properties.add(new Object[]{ "P45","Functionality",null,new String[]{},"Alumno.java","Revisar modificador","Modifier (line 42): el modificador es incorrecto",0.1});
73          properties.add(new Object[]{ "P46","Functionality",null,new String[]{},"Alumno.java","Revisar que el tipo es vacio (void)","VoidType (line 42): el tipo vacio (void) es incorrecto",0.1});
74          properties.add(new Object[]{ "P47","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 42): el nombre es incorrecto",0.1});
75          properties.add(new Object[]{ "P48","Functionality",null,new String[]{},"Alumno.java","Revisar que el tamaño sea el mismo","Size (line 42): el tamaño es incorrecto",0.1});
76          properties.add(new Object[]{ "P49","Functionality",null,new String[]{},"Alumno.java","Revisar misma posición","Index (line 42): la posición del elemento es incorrecta",0.1});
77          properties.add(new Object[]{ "P50","Functionality",null,new String[]{},"Alumno.java","Revisar Name,TypeArguments,SuperType","ClassOrInterfaceType (line 42): El tipo de clase o interface es incorrecta",0.1});
78          properties.add(new Object[]{ "P51","Functionality",null,new String[]{},"Alumno.java","Revisar el nombre","SimpleName (line 42): el nombre es incorrecto",0.1});
79          properties.add(new Object[]{ "P52","Functionality",null,new String[]{},"Alumno.java","Test Functional","Test Case",2.5});
80          properties.add(new Object[]{ "P53","Functionality",null,new String[]{},"Alumno.java","Test Functional","Test Case",2.0});
81
82          return properties;
83      }
84
85      /*****************PROPERTIES TEST SECTION ************************************/
86
87      public boolean checkP52(){
88      return true;
89  }
90
91      public boolean solveP52(){
92      return false;
93  }
94
95  public String[] testP52(){
96      final String testCaseName = "Test";
97          final String testMethodName = "test";
98          final Boolean expectedResult = true;
99          final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName);
100
101         if (expectedResult.equals(testCaseResult)){
102             return null;
103         }else{
104             return new String[] { testCaseName, testMethodName };
105         }
106 }
107
108     public boolean checkP53(){
109     return true;
110 }
111
112     public boolean solveP53(){
```

Ilustración 20 - Ejemplo plantilla corrección 2

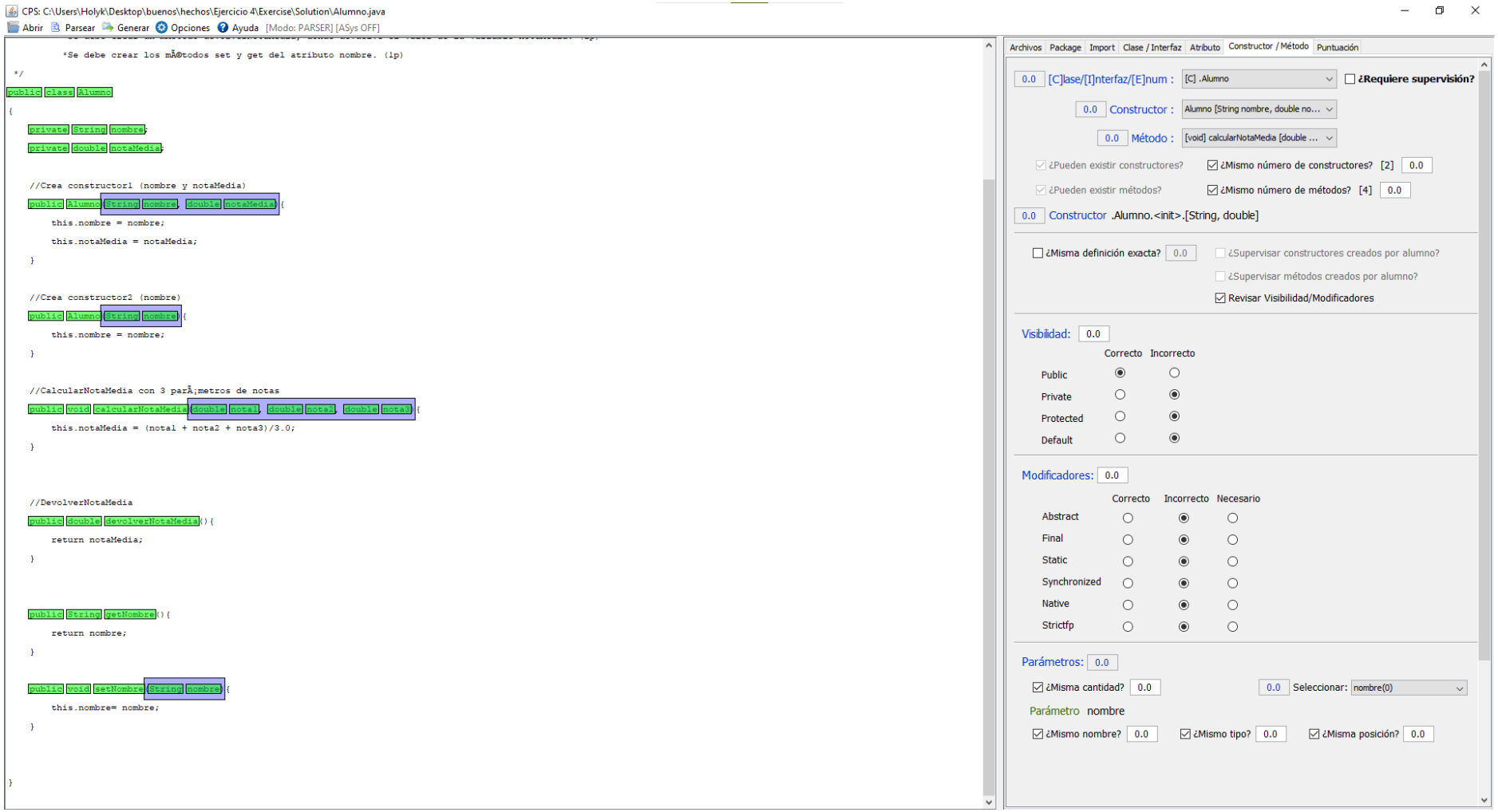

Ilustración 21 - Ejemplo CPS

6.1 Ejercicio 1

Concepto

En este ejercicio se practica el concepto de clase en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como muy fácil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase llamada "X".

6.2 Ejercicio 2

Concepto

En este ejercicio se practican los conceptos de constructor y atributo de una clase en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como muy fácil.

Descripción

Se espera conseguir que el alumno sea capaz de, dada una clase y sus correspondientes métodos *get* y *set*, declarar un constructor sin parámetros que inicialice los atributos.

6.3 Ejercicio 3

Concepto

En este ejercicio se practican los conceptos de variables globales, constructores, getters y setters en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como muy fácil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar variables de clase, un constructor con parámetros que las inicialice y sus correspondientes getters y setters.

6.4 Ejercicio 4

Concepto

En este ejercicio se practican los conceptos de variables globales, tipos de objeto, modificadores, constructores, métodos, getters y setters en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar variables con diferentes modificadores y de diferentes tipos, que cree un constructor para inicializarlas, sus correspondientes getters y setters, y un método de lógica simple.

6.5 Ejercicio 5

Concepto

En este ejercicio se practican los conceptos de variables globales, tipos de objeto, modificadores, constructores, métodos, getters y setters en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar variables con diferentes modificadores y de diferentes tipos, que cree un constructor para inicializarlas, sus correspondientes getters y setters, y métodos de lógica simple.

6.6 Ejercicio 6

Concepto

En este ejercicio se practican los conceptos de librería, métodos y condicionales en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de utilizar una librería externa y crear diferentes métodos con cierta lógica usando diferentes utilidades de dicha librería.

6.7 Ejercicio 7

Concepto

En este ejercicio se practican los conceptos de atributo, método, bucle y array en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar atributos e inicializarlos en el constructor y métodos funcionales usando bucles y arrays.

6.8 Ejercicio 8

Concepto

En este ejercicio se practican los conceptos de método, bucle y estructura condicional en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos funcionales usando bucles y condicionales.

6.9 Ejercicio 9

Concepto

En este ejercicio se practican los conceptos de enumeración, método, bucle, estructura condicional y array en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar enumeraciones y métodos funcionales usando bucles, condicionales y arrays.

6.10 Ejercicio 10

Concepto

En este ejercicio se practican los conceptos de método, bucle y array en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declara rmétodos funcionales usando bucles y arrays.

6.11 Ejercicio 11

Concepto

En este ejercicio se practican los conceptos de atributo, modificador, tipo de objeto, método, bucle y array usando diferentes clases en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de usar, en diferentes clases, métodos, bucles, arrays y atributos usando diferentes modificadores y tipos de objetos.

6.12 Ejercicio 12

Concepto

En este ejercicio se va a practicar el concepto de recursividad en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos recursivos.

6.13 Ejercicio 13

Concepto

En este ejercicio se practican los conceptos de bucle, array y excepción en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de utilizar las excepciones creando métodos usando bucles y arrays.

6.14 Ejercicio 14

Concepto

En este ejercicio se practican los conceptos de bucle, array y excepción en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de utilizar las excepciones creando métodos usando bucles y arrays.

6.15 Ejercicio 15

Concepto

En este ejercicio se practican los conceptos de librería, excepción y E/S de ficheros en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos de entrada y salida de ficheros y hacer uso de las excepciones.

6.16 Ejercicio 16

Concepto

En este ejercicio se practican los conceptos de librería, excepción y E/S de ficheros en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos de entrada y salida de ficheros y hacer uso de las excepciones.

6.17 Ejercicio 17

Concepto

En este ejercicio se practican los conceptos de excepción y E/S de datos en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como muy difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos de entrada y salida de datos, y crear y hacer uso de excepciones propias.

6.18 Ejercicio 18

Concepto

En este ejercicio se practican los conceptos de método, librería, estructura condicional, bucle y array en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos importando librerías y usando condicionales, bucles y arrays en diferentes clases.

6.19 Ejercicio 19

Concepto

En este ejercicio se practican los conceptos de método, bucle y array en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos usando bucles y arrays.

6.20 Ejercicio 20

Concepto

En este ejercicio se practican los conceptos de método, E/S de ficheros, estructura condicional, bucle y array en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar métodos con E/S de ficheros usando condicionales, bucles y arrays en diferentes clases.

6.21 Ejercicio 21

Concepto

En este ejercicio se practican los conceptos de herencia y constructor en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de crear una clase hija usando la herencia y crear un constructor para inicializarla.

6.22 Ejercicio 22

Concepto

En este ejercicio se practican los conceptos de herencia, modificador y constructor en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear una clase hija usando la herencia y crear un constructor para inicializarla.

6.23 Ejercicio 23

Concepto

En este ejercicio se practican los conceptos de herencia, constructor y método en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear una clase hija usando la herencia y crear un constructor para inicializarla y sobrescribir métodos de la clase padre.

6.24 Ejercicio 24

Concepto

En este ejercicio se practican los conceptos de herencia, constructor y método en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz crear diferentes clases hijas usando la herencia y crear constructores para inicializarlas y sobrescribir métodos de la clase padre.

6.25 Ejercicio 25

Concepto

En este ejercicio se va a practicar el uso de la clase Arraylist en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar un array y crear diferentes métodos para hacer diferentes lógicas usando *Arraylist*.

6.26 Ejercicio 26

Concepto

En este ejercicio se practican los conceptos de herencia, *Arraylist* y polimorfismo en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de usar la herencia en diferentes clases usando *Arraylist* y polimorfismo.

6.27 Ejercicio 27

Concepto

En este ejercicio se practica el concepto de interfaz en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de usar una interfaz y crear una clase completa.

6.28 Ejercicio 28

Concepto

En este ejercicio se practica el concepto de interfaz en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de usar una interfaz y crear una clase completa.

6.29 Ejercicio 29

Concepto

En este ejercicio se practica el concepto de clase abstracta en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de usar una clase abstracta y crear una clase completa.

6.30 Ejercicio 30

Concepto

En este ejercicio se practican los conceptos de clase abstracta y herencia en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de usar el concepto de clases abstractas y herencia de forma más compleja intercalando varios niveles.

6.31 Ejercicio 31

Concepto

En este ejercicio se practican los conceptos de interfaz y clase abstracta en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase extendiendo de una clase abstracta e implementando una interfaz.

6.32 Ejercicio 32

Concepto

En este ejercicio se va a practicar el concepto de genericidad en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como fácil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase genérica.

6.33 Ejercicio 33

Concepto

En este ejercicio se va a practicar el concepto de genericidad en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase genérica y usarla en otra clase mediante arraylist.

6.34 Ejercicio 34

Concepto

En este ejercicio se va a practicar el concepto de lista y *Arraylist* en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "lista" utilizando arraylist.

6.35 Ejercicio 35

Concepto

En este ejercicio se va a practicar el concepto de lista en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "lista" sin utilizar *Arraylist*, con nodos y posiciones.

6.36 Ejercicio 36

Concepto

En este ejercicio se va a practicar el concepto de lista y pila en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "pila" haciendo uso de la estructura de una lista.

6.37 Ejercicio 37

Concepto

En este ejercicio se va a practicar el concepto de pila en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "pila" sin hacer uso de la estructura de una lista.

6.38 Ejercicio 38

Concepto

En este ejercicio se va a practicar los conceptos de lista y cola en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "cola" haciendo uso de la estructura de una lista.

6.39 Ejercicio 39

Concepto

En este ejercicio se va a practicar el concepto de cola en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "cola" sin hacer uso de la estructura de una lista.

6.40 Ejercicio 40

Concepto

En este ejercicio se va a practicar el concepto de árbol binario en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "árbol binario ordenado". Donde se podrán insertar elementos y mostrarlo ordenado.

6.41 Ejercicio 41

Concepto

En este ejercicio se va a practicar el concepto de árbol binario en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de crear la estructura de datos "árbol binario ordenado". Donde se podrán insertar elementos y poseerá métodos para consultar datos del árbol; si tiene hijos, la altura del árbol, cantidad de nodos Hoja, búsqueda de un determinado nodo, posibilidad de borrar nodos...

6.42 Ejercicio 42

Concepto

En este ejercicio se va a practicar el concepto de *HashMap* en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar un *HashMap* y crear nodos para insertar elementos, consultar elementos del *Map* y devolver el mapa.

6.43 Ejercicio 43

Concepto

En este ejercicio se va a practicar el concepto de *HashMap* en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear un "juego" utilizando *Hashmap* para crear una recreación simple de "piedra, papel o tijera" mediante la creación de valores aleatorios haciendo comparaciones y mostrando el resultado.

6.44 Ejercicio 44

Concepto

En este ejercicio se va a practicar el concepto de recursividad en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase con métodos recursivos siguiendo la estrategia de divide y vencerás.

6.45 Ejercicio 45

Concepto

En este ejercicio se practican los conceptos de interfaz y polimorfismo en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear una interfaz con dos métodos y tres clases que implementen dicha interfaz; más una clase resultante llamada Prueba, para crear diferentes instancias de dichas clases y crear un método padre con un *ArrayList* de la interfaz donde se puedan almacenar todas las diferentes instancias.

6.46 Ejercicio 46

Concepto

En este ejercicio se va a practicar el concepto de hilos en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear 2 clases que hereden de *Thread* y donde el método *run* imprima cada clase un carácter diferente un mínimo de 1000 veces. Para probar si se ha realizado correctamente, se puede crear una clase *Test* donde se ejecuten 2 instancias, 1 de cada hilo, y visualizando lo que se imprime por pantalla se podrá ver que se intercalan los caracteres y que por lo tanto se han ejecutado ambos hilos.

6.47 Ejercicio 47

Concepto

En este ejercicio se va a practicar el concepto de hilo en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de crear 2 clases que implementen la clase *Runnable* y donde el método *run* imprima cada clase un carácter diferente un mínimo de 1000 veces. Para probar si se ha realizado correctamente, se puede crear una clase *Test* donde se ejecuten 2 instancias, 1 de cada hilo, y visualizando lo que se imprime por pantalla se podrá ver que se intercalan los caracteres y que por lo tanto se han ejecutado ambos hilos.

6.48 Ejercicio 48

Concepto

En este ejercicio se van a practicar los conceptos de clase abstracta y herencia en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como de dificultad media.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase abstracta con atributos, getter, setters y un método abstracto. Dos clases que extiendan de la clase abstracta y que den ejecuciones diferentes al método abstracto.

6.49 Ejercicio 49

Concepto

En este ejercicio se va a practicar el concepto de genericidad en la programación orientada a objetos.

Dificultad

Este ejercicio está catalogado como difícil.

Descripción

Se espera conseguir que el alumno sea capaz de declarar una clase genérica y usarla en otra clase mediante Maps.

6.50 Ejercicio 50

Concepto

En este ejercicio se va a practicar el concepto de hilo en la programación orientada a objetos

Dificultad

Este ejercicio está catalogado como muy difícil.

Descripción

Se espera conseguir que el alumno sea capaz de crear una clase que extienda la clase *Thread* y posea métodos con lógica para dar información de su hilo. Crear otra clase para declarar varios hilos y que usen sus métodos para obtener información de forma síncrona.

7. Pruebas

Todos los ejercicios se han implementado y probado tanto de manera independiente (usando la máquina virtual de Java) como con ASys. Por un lado se han desarrollado las soluciones y los test funcionales, mediante CPS y la solución del ejercicio se han generado las plantillas de corrección necesarias en ASys para su autocorrección.

Una vez especificadas las diferentes características en CPS (ilustración 21) a evaluar mediante ASys se ha añadido en las plantillas generadas las propiedades de los test funcionales para que los autoevaluara ASys.

De esta forma se ha generado la estructura de ejercicio necesaria para cargarla en ASys y poder implementar una propuesta de solución al ejercicio para su posterior autocorrección.

Se han realizado clases de prueba para verificar la funcionalidad de los ejercicios mediante el uso de las herramientas.

Se carga el código a evaluar en ASys y veremos la siguiente pantalla:

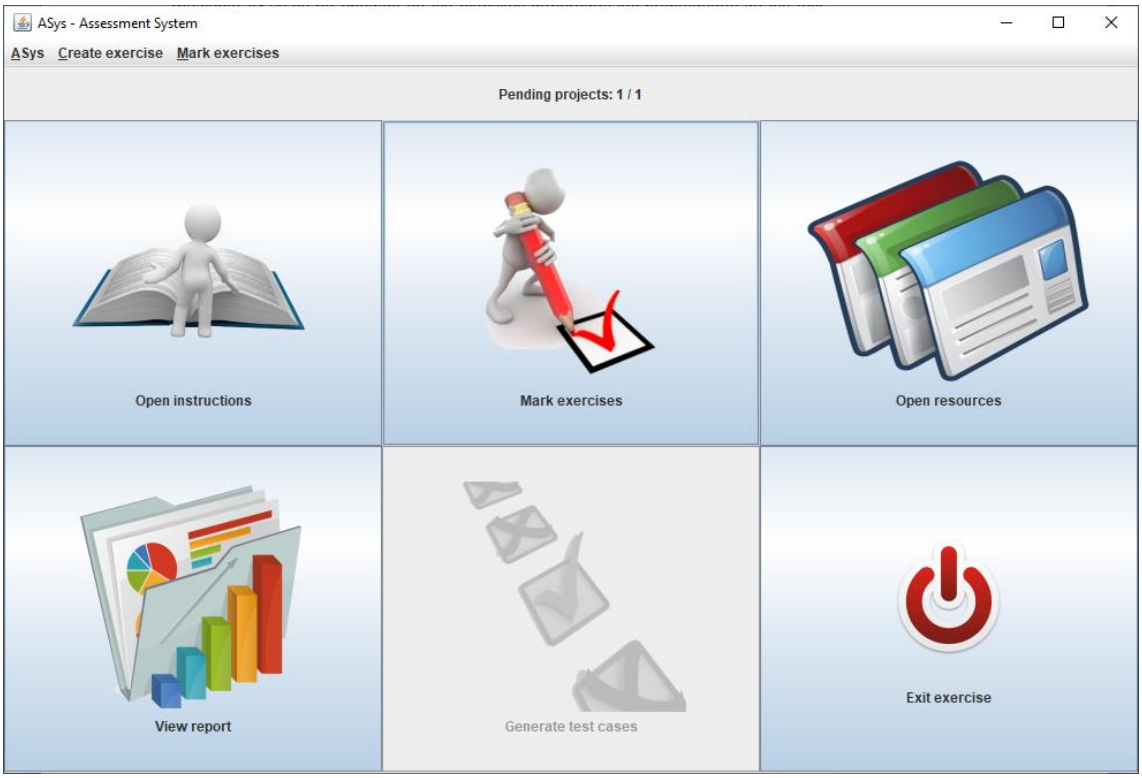

Ilustración 22 - Asys tarea pendiente

En la imagen se puede ver como hay un proyecto pendiente de evaluar, si pulsas en *"Mark exercises"* se corregirá.

En una primera prueba hemos introducido la propia solución del docente como código a evaluar para obtener el 100% de la nota y de esa forma demostrar que la

funcionalidad básica la cubre. Al no tener errores no se visualiza ninguna pantalla de comparación de código y pulsando en "*View report"* puedes ver el resultado.

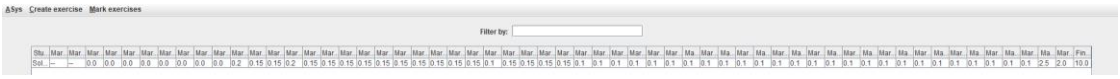

Ilustración 23 - Resultado ASys

Además de la prueba "perfecta" se han llevado a cabo otra serie de pruebas para verificar que funcionan correctamente:

1. Se ha llevado a cabo una prueba donde se genera un error de compilación y, por lo tanto, no puede evaluarlo. En la prueba se puede observar como en un método donde se espera pasar como parámetro un tipo *double*, se le mete un *String* provocando así un error de cast.

2. En una segunda prueba se ha modificado un modificador de los atributos para ver cómo se muestra una leve falta en la solución, sin afectar a la ejecución y la funcionalidad. En esta prueba se puede observar como al haber introducido los 2 atributos como *public* en lugar de *private*, se ha descontado 0,2 por cada fallo.

3. En una tercera prueba se ha cambiado la funcionalidad de un método para ejecutar un fallo funcional y ver como el test funcional falla. En esta prueba se puede observar cómo se ha modificado un método donde se calcula la nota media de 3 notas para que devuelva siempre un 777.0.De esta forma, ha fallado un test funcional con un valor de 2,5 puntos.

En las Ilustraciones 24, 25 y 26 se puede ver la visualización de cada prueba y su comparación con la solución del docente en ASys. De esta forma el alumno puede ver en que ha fallado y la puntuación de ese fallo.

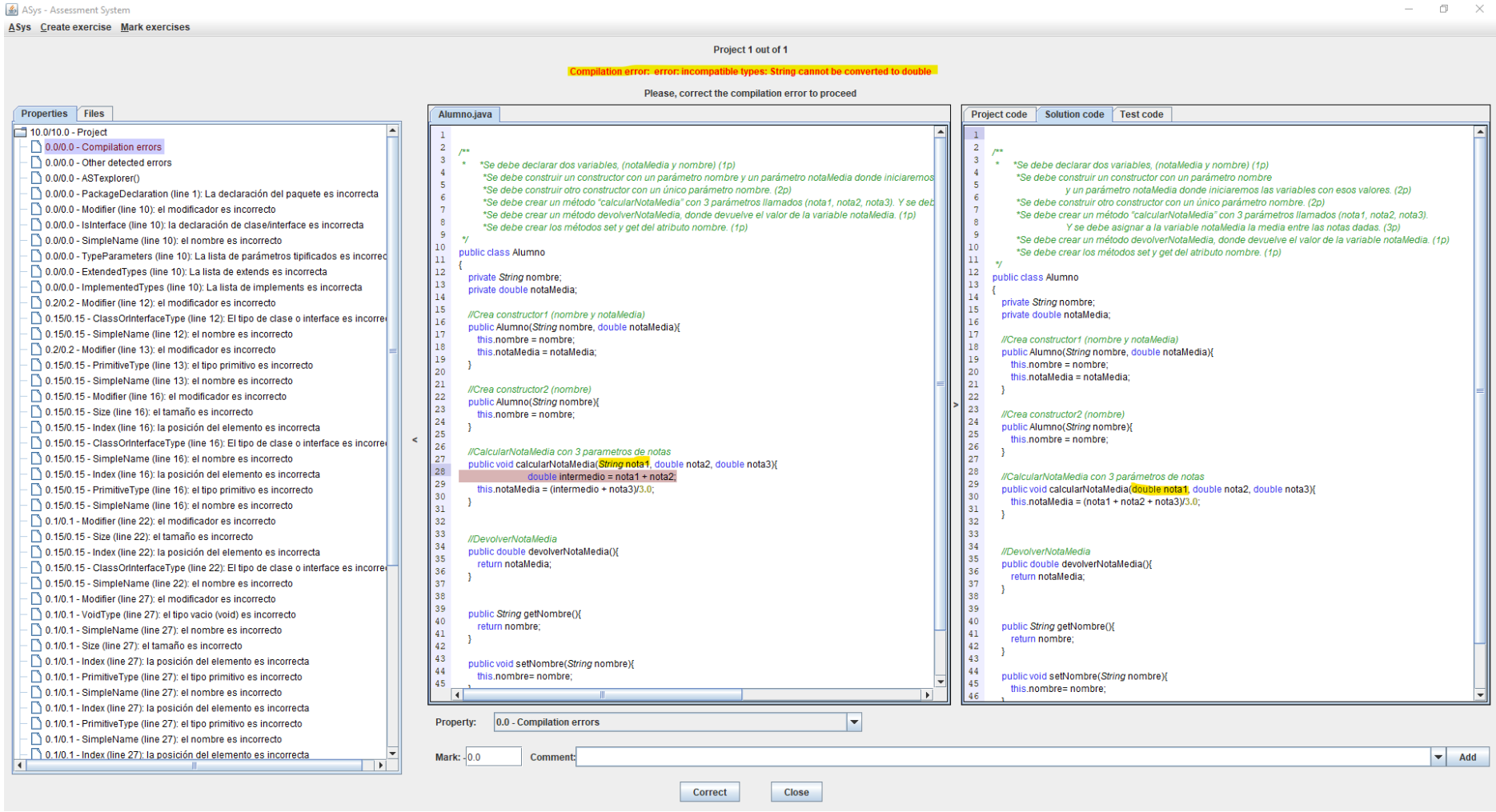

Ilustración 24 - ASys corrección fallo compilación

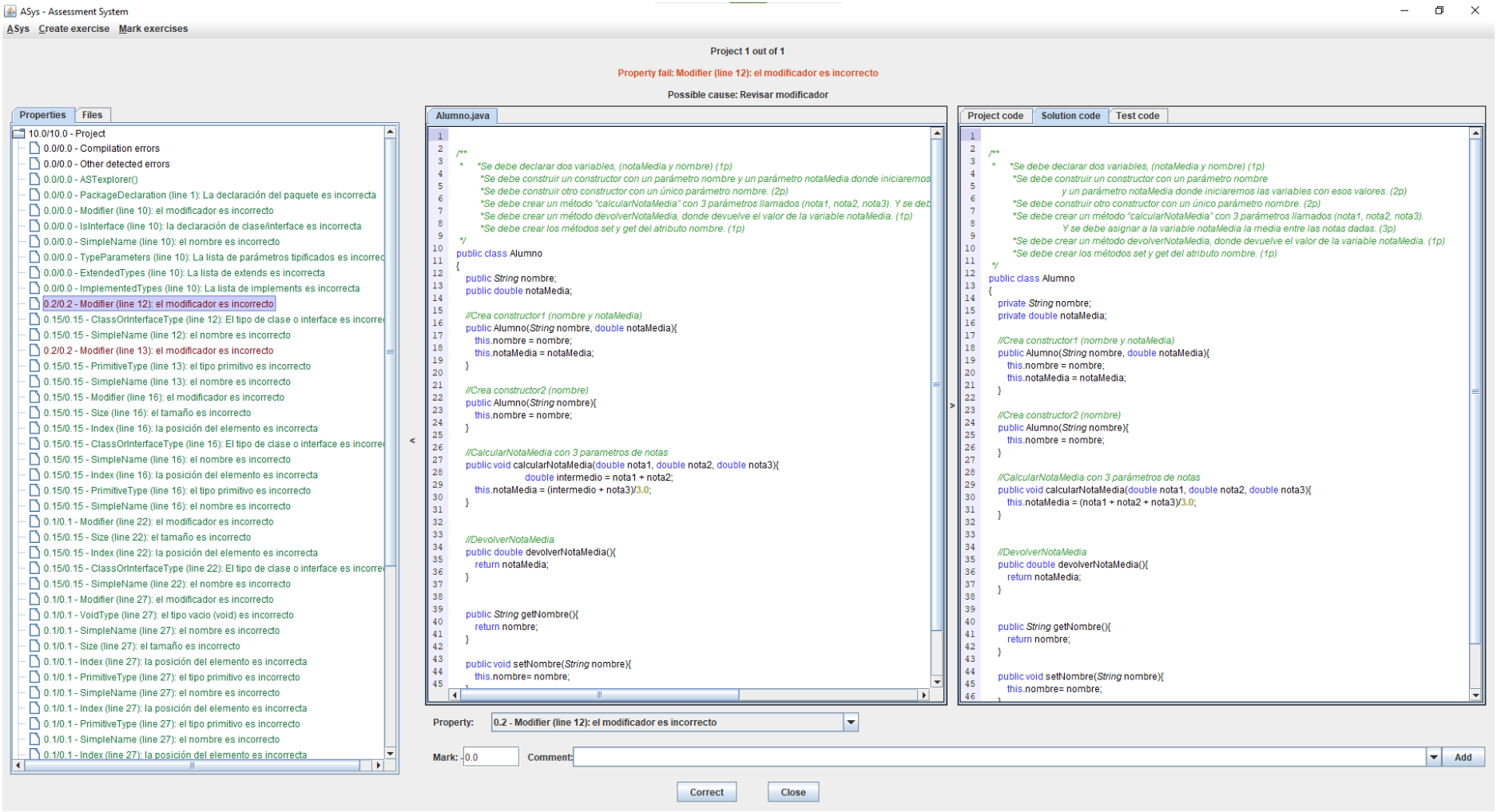

Ilustración 25 - ASys corrección fallo de modificadores

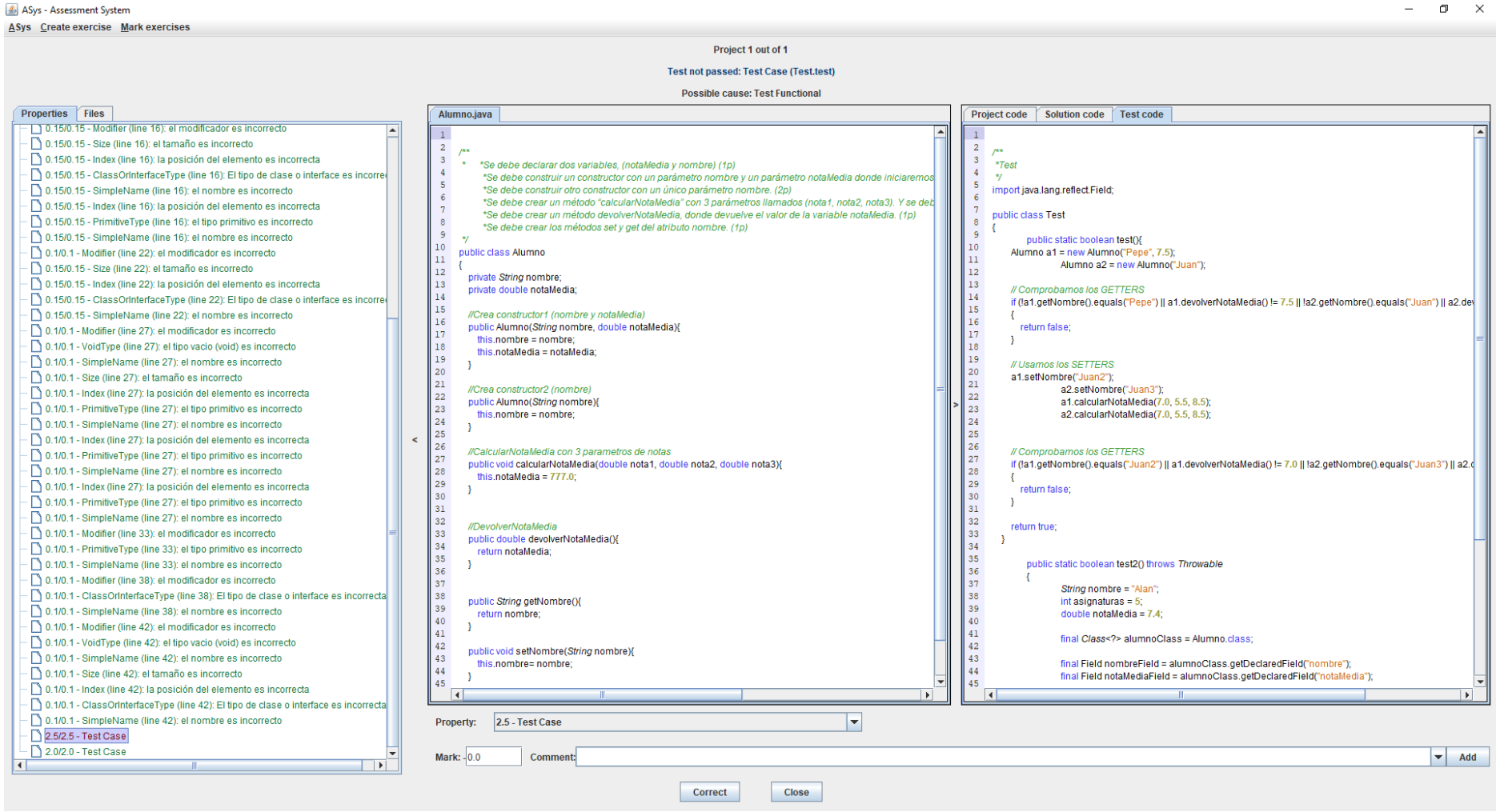

Ilustración 26 - ASys corrección fallo funcional

8. Conclusión

El origen de este proyecto fue el de dotar a los alumnos de un repositorio de ejercicios vinculantes a los temarios impartido en clase para terminar de afianzar de forma autónoma los conocimientos adquiridos. Este repositorio requería explorar todas las características impartidas de una forma amigable y progresiva para familiarizar a los alumnos con la programación y que los docentes pudieran usar estas herramientas como una ayuda a sus asignaturas.

La realización del repositorio se ha llevado a cabo en el lenguaje de programación Java ya que es el lenguaje principal usado en las asignaturas de introducción a la programación en la UPV. Por otro lado, se han debido adaptar a la estructura que requerían las herramientas externas para su utilización ASys y CPS.

Durante la realización de los ejercicios han surgido múltiples problemas; entre ellos, el situarse en la piel de un docente e intentar formular los ejercicios de la forma más adecuada para exponer el enunciado sin relevar la solución. Por otro lado el sistema de puntuación de cada ejercicio, el cómo dividir que esta sección valga X y la otra Y.

Por otro lado ha requerido el aprendizaje del uso de las herramientas externas ASys y CPS para la verificación de los ejercicios en el caso de ASys y para la creación de las plantillas de corrección con CPS.

Una vez teníamos creado los ejercicios, las plantillas de corrección y los test funcionales, surgieron varios errores cuando tuvimos que juntar las plantillas de corrección con los test funcionales. En algunos casos no cogía la puntuación adecuada y en otros casos las plantillas de corrección estaban mal planteadas ya que en función de qué opción elegías en el CPS requería que el alumno hiciera una solución exactamente idéntica a la ofrecida y esto imposibilitaba otras formas de llevar a una solución totalmente válida.

Probando la extensa funcionalidad de las herramientas pudimos deducir el flujo correcto para crear las plantillas de corrección y se hicieron pruebas verificando su funcionamiento.

Como resultado de este TFG he podido adquirir una comprensión de las dificultades en las que se encuentran los docentes a la hora de elaborar los ejercicios adecuados que muestren que el alumno ha asimilado el temario, no es tarea fácil el poder determinar un corte para otorgar cierto valor al ejercicio ni mucho menos indicar que si eres capaz de hacer este ejercicio has entendido este concepto, ya que todo concepto de programación puede tener diversas utilidades que no son demostrables en un único ejercicio.

Por otro lado, al haber tenido que pensar y planear unos ejercicios didácticos desde el otro lado, la parte docente, he tenido que realizar una investigación profunda de todas las características impartidas de la programación orientada a objetos sentando así unos conocimientos en la materia más profundos.

Como resultado de este proyecto se dispone de 50 objetos de aprendizaje. Cada uno de ellos es un ejercicio tradicional de Java con el enunciado, el código de la solución y casos de prueba. Además, cada uno de ellos ha sido reprogramado en el sistema CPS para convertirlo en un ejercicio auto-corregible. Los 50 objetos de aprendizaje forman parte de una taxonomía para el aprendizaje de la programación en Java.

El trabajo futuro inmediato consiste en introducirla taxonomía en Poliformat, de tal forma que se disponga desde todas las asignaturas de la carrera de un recurso de aprendizaje común y transversal a las asignaturas de programación. De esta forma, los alumnos tendrán una guía de ejercicios paralela a las asignaturas y transversal a las mismas. Además, los profesores de diferentes asignaturas podrán conocer el avance de los alumnos una vez llegan a su asignatura y producir una enseñanza personalizada y colaborativa.

Otras líneas de trabajo futuro pueden proporcionar muchas mejoras:

- Un aumento del repositorio con más ejercicios para dotar a los alumnos de una mayor variedad.
- Nuevos repositorios de ejercicios más avanzados orientados a las ramas de especialización de la universidad.
- Nuevos repositorios donde se pueda introducir el uso de los frameworks y tecnologías más demandados en la actualidad ya sea en entornos laborales o de investigación.
- Una interfaz más intuitiva y una mejora de la herramienta CPS para poder generar unas plantillas más editables. Sobre todo en el ámbito funcional.
- Dotar a ASys y a CPS para además de Java, soportar la corrección de otros lenguajes de programación impartidos en la carrera.
- Detección de buenas prácticas, robustez, escalabilidad en el código, duplicidad.

Bibliografía

1. Bloom, B (1971).*Taxonomía de los objetivos de la educación: la clasificación de las metas educacionales*. El Ateneo

2. Camperos, M (1992).*De los fines educativos a los objetivos educacionales: una taxonomía para la planificación y evaluación del aprendizaje*. Universidad Central de Venezuela, Consejo de Desarrollo Científico y Humanístico

3. *Crec (2021). Cuatro taxonomías educativas para crear e-learning. C*rec.mx. Recuperado en mayo del 2020 de http://www.crec.mx/ispring/2016/07/19/cuatro-taxonomias-educativas-crear-e-learning/

4. DocPlayer (2021). *Taxonomía de aprendizaje de Bloom y Camperos.* DocPlayer. Recuperado en mayo del 2020 de https://docplayer.es/9679827-Taxonomia-de-aprendizaje-de-bloom.html

5. Eclipse (2021). Herramienta de trabajo para la implementación de los ejercicios https://www.eclipse.org/

6. *Eduteka (2021).La taxonomía de Bloom y sus actualizaciones*. Eduteka.com. Recuperado en mayo del 2020 de http://eduteka.icesi.edu.co/articulos/TaxonomiaBloomCuadro

7. Guilford, JP(1967).*The Nature of Human Intelligence*. McGraw-Hill Inc

8. InsaD.,Silva J., (2018).Automatic Assessment of Java Code. Computer Languages Systems & Structures, 53: 59-72, ISSN 1477-8424. Doi: https://doi.org/10.1016/j.cl.2018.01.004

9. Insa D, Pérez S, Silva J, Tamarit S (2020). Semiautomatic generation and assessment of Java exercises in engineering education. ComputApplEngEduc. 2020;1–17. https://doi.org/10.1002/cae.22356

10. *Javadesdecero (2021). Java Básico.*Javadesdecero.com. Recuperado en septiembre del 2020 de https://javadesdecero.es/basico/

11. *Javaya (2021).*Lecciones programación Java. Tutorialesprogramacionya.com. Recuperado en septiembre del 2020 de https://www.tutorialesprogramacionya.com/javaya/index.php?inicio=0

12. Marzano, R.J. y Kendall, J.S.(2007).*The new taxonomy of educational objectives.* California. Corwin Press

13. Marzano, R.J. y Kendall, J.S. (2007). *Designing and assessing educational objectives: Applying the new taxonomy.* Corwin Press

14. MediaUpv (2021). Silva, J. *Upv Media*; Recuperado en mayo del 2020 https://media.upv.es/#/portal/search?q=ASys

15. *Mkyong (2021). How to create XML file in Java.* Mkyong.com. Recuperado en mayo del 2020 de https://mkyong.com/java/how-to-create-xml-file-in-java-dom/

16. Morales, J (2016). *XML-DOM.* Recuperado en mayo del 2020 de http://jmoral.es/blog/xml-dom

17. Natividad Pet al., (2012). *Empezar a programar usando Java.* Universitat Politècnica de València

18. OrientacionAnduja *(2021). Taxonomía de Robert Marzano verbos recomendados para indicadores y niveles cognitivos.* OrientacionAnduja. Recuperado en septiembre del 2020 de https://www.orientacionandujar.es/2016/11/06/taxonomia-robert-marzano-verbos-recomendados-indicadores-niveles-cognitivos/

19. *Programarya(2021). Curso Java*. Programarya.com. Recuperado en mayo del 2020 de https://www.programarya.com/Cursos/Java

20. Psicologiaymente*(2021). La teoría de la Inteligencia de Guilford.* Psicologiaymente.com. Recuperado en mayo del 2020 de https://psicologiaymente.com/inteligencia/teoria-inteligencia-guilford

21. Psicologiaymente*(2021b). Taxonomía de Marzano: qué es, objetivos, y qué partes tiene.* PsicologiayMente. Recuperado en septiembre del 2020 de https://psicologiaymente.com/desarrollo/taxonomia-marzano

22. RileyD (1951). *The object of Java : introduction to programming using software engineering principles*. Addison-Wesley

23. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 3, Data Types*, pp 61-98. Appress

24. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 4, Operators,* pp 99-138. Appress

25. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 5, Statements,* pp 139-164. Appress

26. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 6, Classes and objects,* pp 165-280. Appress

27. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 7, The object and objects classes,* pp 281-316. Appress

28. Sharan K (2014). *Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 9, Exceptions Handling,* pp 335-378. Appress

29. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 11, String,* pp 387-410. Appress

30. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 15, Arrays,* pp 543-582. Appress

31. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 16, Inheritance,* pp 583-642. Appress

32. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 17, Interfaces,* pp 643-704. Appress

33. Sharan K (2014).*Beginning Java 8 Fundamentals, Language Syntax, Arrays, Data Types, Objects, and Regular Expressions. Chapter 18, Enum Types,* pp 705-726. Appress

34. *Significados (2021). Taxonomía*. Significados. Recuperado en mayo del 2020 de https://www.significados.com/taxonomia/

35. Stackoverflow (2021). Plataforma usada para consultas básicas sobre la implementación; http://stackoverflow.com

36. *TaxonomiaMarzano (2021). Comprendiendo la taxonomía. T*axonomiamarzano.weebly.com. Recuperado en mayo del 2020 de https://taxonomiamarzano.weebly.com/comprendiendo-la-taxonomiacutea.html

37. *UCM (2021). IEEE 830*. Fdi.ucm.es**.** Recuperado en septiembre del 2020 de https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

38. Weiss, M.A, (2013). *Estructuras de datos en Java.* Pearson Educación.

39. *Wikipedia (2021). Joy Paul Guilford*. Wikipedia.com. Recuperado en septiembre del 2020 de https://es.wikipedia.org/wiki/Joy_Paul_Guilford

40. *Wikipedia (2021b). Definición de taxonomía*. Wikipedia.com. Recuperado en mayo del 2020 de https://es.wikipedia.org/wiki/Taxonom%C3%ADa

41. *Wikipedia (2021c). Taxonomía de objetivos de la educación*. Wikipedia.com. Recuperado en mayo del 2020 de https://es.wikipedia.org/wiki/Taxonomía_objetivos_de_la_educación