EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Orizaba

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OPCIÓN I.- TESIS

TRABAJO PROFESIONAL

"DESARROLLO DE UN MÓDULO DE SOFTWARE PARA APOYAR EL APRENDIZAJE DEL PROCESO DE CONSTRUCCIÓN E INICIALIZACIÓN DE OBJETOS EN JAVA"

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:

I.T.I. Leónides Pérez Ortiz

DIRECTOR DE TESIS:

Dr. Ulises Juárez Martínez

CODIRECTOR DE TESIS:

José Luis Sánchez Cervantes

ORIZABA, VERACRUZ, MÉXICO. **OCTUBRE 2023**

Orizaba, Veracruz, 02/octubre/2023
Dependencia: **División de Estudios de Posgrado e Investigación**
Asunto: **Autorización de Impresión**
OPCION: **I**

C. LEÓNIDES PÉREZ ORTIZ
CANDIDATO A GRADO DE MAESTRO EN:
SISTEMAS COMPUTACIONALES
P R E S E N T E.-

De acuerdo con el Reglamento de Titulación vigente de los Centros de Enseñanza Técnica Superior, dependiente de la Dirección General de Institutos Tecnológicos de la Secretaría de Educación Pública y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

" Desarrollo de un módulo de software para apoyar el aprendizaje del proceso de construcción e inicialización de objetos en Java"

comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

Cuauhtémoc Sánchez R.

DR. CUAUHTÉMOC SÁNCHEZ RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA | TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLOGICO DE ORIZABA
DIV. DE EST. DE POSGRADO E INVEST.

OG-13-F06

Orizaba, Veracruz, 19/septiembre/2023
Asunto: **Revisión de trabajo escrito**

C. CUAUHTÉMOC SÁNCHEZ RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
P R E S E N T E.-

Los que suscriben, miembros del jurado, han realizado la revisión de la Tesis del (la) C.

LEÓNIDES PÉREZ ORTIZ

La cual lleva el título de:

Desarrollo de un módulo de software para apoyar el aprendizaje del proceso de construcción e inicialización de objetos en Java

Y concluyen que se acepta.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
CIENCIA – TÉCNICA - CULTURA®

PRESIDENTE:	DR. ULISES JUÁREZ MARTÍNEZ	FIRMA
SECRETARIO:	DR. JOSÉ LUIS SÁNCHEZ CERVANTES	FIRMA
VOCAL:	DRA. LISBETH RODRÍGUEZ MAZAHUA	FIRMA
VOCAL SUP.:	M.C. MARÍA ANTONIETA ABUD FIGUEROA	FIRMA

TA-09-18

CARTA DE CESIÓN DE DERECHOS

En la ciudad de Orizaba, Veracruz, el día 19 del mes de septiembre del año 2023, el (la) que suscribe Pérez Ortiz Leónides, alumno (a) del programa de Maestría en Sistemas Computacionales con número de control M21011179, manifiesta que es autor (a) intelectual del trabajo de tesis bajo la dirección del Dr. Ulises Juárez Martínez y cede los derechos del trabajo intitulado "Desarrollo de un módulo de software para apoyar el aprendizaje del proceso de construcción e inicialización de objetos en Java" al TecNM/Instituto Tecnológico de Orizaba para su difusión, con fines académicos y de investigación.

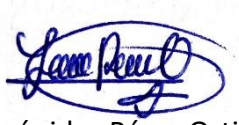
Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y del director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección: msc@orizaba.tecnm.mx . Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Leónides Pérez Ortiz

Nombre y firma

CARTA DE ORIGINALIDAD

En la ciudad de Orizaba, Veracruz, el día 19 del mes de septiembre del año 2023, el (la) que suscribe Pérez Ortiz Leónides, alumno (a) del programa de Maestría en Sistemas Computacionales con número de control M21011179, manifiesta que es autor (a) del trabajo de tesis titulado "Desarrollo de un módulo de software para apoyar el aprendizaje del proceso de construcción e inicialización de objetos en Java" y declara que el trabajo es original, ya que su contenido es producto de su directa contribución intelectual. Todos los datos y las referencias a materiales ya publicados están debidamente identificados con su respectivo crédito e incluidos en las notas bibliográficas y en las citas que se destacan como tal y, en los casos que así lo requieran, se tienen las debidas autorizaciones de quienes poseen los derechos patrimoniales. Por lo tanto, se hace responsable de cualquier litigio o reclamación relacionada con derechos de propiedad intelectual, exonerando de toda responsabilidad al Tecnológico Nacional de México / Instituto Tecnológico de Orizaba.



Leónides Pérez Ortiz

Nombre y firma autógrafa

Agradecimientos

Al Tecnológico Nacional de México por la oportunidad de ingresar a la Maestría en Sistemas Computacionales.

Al Consejo Nacional de Humanidades, Ciencias y Tecnologías, CONAHCyT, por el apoyo económico otorgado durante la realización de mis estudios de maestría.

A mi director de tesis, Dr. Ulises Juárez Martínez, por brindarme sus invaluables conocimientos para llevar a cabo esta investigación, por mostrar empatía en momentos difíciles y por su gran paciencia, comprensión y apoyo.

A la maestra Nancy Aracely Cruz Ramos, por alentarme a ingresar al programa de maestría y por todo el apoyo otorgado de inicio a fin.

A mis compañeros y amigos de la maestría, por compartir conmigo sus conocimientos y por los buenos momentos que hemos compartido desde que nos conocimos.

A mi mejor amiga Adriana, por estar conmigo en cada etapa de mi formación profesional y por alentarme en cada situación a la que tengo que enfrentarme.

A mi familia, es especial a mi madre, por impulsarme siempre a conseguir mis sueños, por cada consejo dado, por su amor incondicional y por motivarme en cada circunstancia de la vida.

Índice general

Índice de tablas

Índice de figuras

Índice de códigos

Abreviaturas

PCIO - Proceso de Construcción e Inicialización de Objetos.

POO - Programación Orientada a Objetos.

OOP - Object-Oriented Programming.

JVM - Java Virtual Machine, Máquina Virtual de Java.

HTTP - Hypertext Transfer Protocol, Protocolo de Transferencia de Hipertexto.

JSON - JavaScript Object Notation, Notación de Objetos de JavaScript.

PYPL - PopularitY of Programming Language Index, Índice de Popularidad del Lenguaje de Programación.

REST - Representational State Transfer, Transferencia de Estado Representacional.

XP - eXtreme Programming, Programación Extrema.

IDE - Integrated Development Environment, Entorno de Desarrollo Integrado.

GUI - Graphical User Interface, Interfaz Gráfica de Usuario.

POST - Power - On Self-Test.

WWW - World Wide Web.

URL - Uniform Resource Locator, Localizador de Recursos Uniforme.

UML - Unified Modeling Language, Lenguaje de Modelado Unificado.

Resumen

Conocer el proceso de construcción e inicialización de objetos es pieza clave para el aprendizaje de cualquier lenguaje de programación orientado a objetos. A medida que los estudiantes y/o programadores identifican los pasos del proceso de construcción, es mucho más fácil codificar programas eficientes porque se trabaja de una manera más consciente de lo que se está programando.

Como alternativa de solución, en el presente trabajo se desarrolló un módulo de aprendizaje como apoyo en el manejo del proceso de construcción e inicialización de objetos en *Java*. El módulo contiene una guía interactiva para verificar que los pasos del proceso de construcción se realicen correctamente.

El principal beneficio que se espera del módulo de aprendizaje es que facilite la comprensión del proceso de construcción, principalmente que reduzca las dificultades al momento de analizar cualquier código que incluya clases y objetos. El área de enfoque de este proyecto se centra en la educación, dirigido primeramente a estudiantes de programación.

Este proyecto implementa *Blockly* como tecnología principal, esta biblioteca se encuentra del lado del cliente (*frontend*) para efectos de arrastrar y soltar bloques de código visuales y generar el código fuente necesario para posteriormente ser enviado al servicio web de validaciones (*backend*).

Abstract

Knowing the process of construction and initialization of objects is a key element for learning any object-oriented programming language. As students and/or programmers identify the steps in the build process, it becomes much easier to code efficient programs because you're working more consciously of what you are programming.

As an alternative solution, in the present work a learning module was developed to support the handling of the process of construction and initialization of objects in Java. The module contains an interactive guide to verify that the steps of the build process are completed correctly.

The main benefit that is expected from the learning module is that it facilitates the understanding of the construction process, mainly that it reduces the difficulties when analyzing any code that includes classes and objects. The focus area of this project is focused on education, aimed primarily at programming students.

This project implements Blockly as the main technology, this library is on the client side (frontend) for the purposes of dragging and dropping blocks of visual code and generate the necessary source code to later be sent to the validation web service (backend).

Introducción

Java, es un lenguaje de propósito general que trata los objetos como entidades que tienen estado (representa los datos o valores) y comportamiento (representa la funcionalidad). Asimismo, la identidad de un objeto se implementa a través de un identificador único, es decir, ese identificador no es visible para usuarios comunes, sin embargo, la Máquina Virtual de Java (JVM) utiliza ese identificador para monitorear un objeto de forma única [1]. Pero ¿por qué es importante comprender todo lo referente a un objeto durante su proceso de construcción e inicialización en un lenguaje de programación como Java?

Según análisis de la compañía Edix Digital Workers® [2] a través de herramientas como el índice TIOBE [3] y el índice PYPL [4], determina que Java, un lenguaje de programación con un ámbito de aplicación amplío, se encuentra dentro de los primeros tres lenguajes de programación más utilizados en los últimos años. Mensualmente el índice TIOBE muestra el ranking de los lenguajes de programación más utilizados, es decir, indica en qué lenguaje se codificó más durante el último mes; por otro lado, el índice PYPL ofrece un ranking basado en *Google Trends* que señala las tendencias de búsqueda de Google en relación con la cantidad de veces que se realizaron búsquedas de tutoriales clasificados por lenguaje de programación, la clasificación indica que, cuanto mayor número de búsquedas tenga un lenguaje determinado, más popular es respecto a otros.

Considerando lo anterior, el módulo de aprendizaje propuesto para el estudio del proceso de construcción e inicialización de objetos en Java, se considera una herramienta

valiosa e importante como apoyo para la formación de estudiantes. La importancia de comprender correctamente el proceso de construcción radica en un desarrollo eficiente, consciente y mejoras en la resolución de problemas y a través del uso de bloques visuales se pretende que el usuario se familiarice con cada parte del código, programe en tiempo de diseño y comprenda la ejecución.

Para ofrecer una mejor visión de esta investigación, el presente documento está constituido por cinco capítulos que se describen a continuación.

El capítulo 1 se enfoca en dar a conocer los conceptos relevantes para entendimiento del proyecto, el planteamiento del problema, el objetivo general, los objetivos específicos y justificación del proyecto.

En el capítulo 2 se describen algunos trabajos que, aunque no tienen relación directa con el tema de interés, la mayoría abarca Programación Orientada a Objetos, lo que incluye clases, objetos y algunos fragmentos del PCIO. Se realiza una comparativa entre todos los trabajos recopilados. Se presenta además la propuesta de solución y las herramientas a utilizar para alcanzar los objetivos de la tesis.

En el capítulo 3 se muestra el desarrollo de la metodología de tesis, se exploran las características y se describen las etapas del modelo en cascada. En el capítulo 4 se presentan los resultados obtenidos tomando como base un caso de estudio académico y se da a conocer una encuesta realizada durante este caso de estudio.

Finalmente, en el capítulo 5 se presentan las conclusiones y recomendaciones para este proyecto.

Capítulo 1

Antecedentes

En este capítulo se presentan diversos conceptos que resultan relevantes para este tema de tesis. También se describe la problemática a resolver, el objetivo general, los objetivos específicos y la justificación del presente trabajo.

1.1. Marco teórico

En esta sección se dan a conocer los conceptos de mayor relevancia que tienen relación con el tema de tesis.

1.1.1. Software

Software [5] es el conjunto de instrucciones o programas que se le asignan a una computadora para ejecutar tareas en particular (esto es, un algoritmo).

1.1.2. Tipos de software

Existen tres tipos de software: de sistema, de aplicación y software de programación; mismos que se indican en la figura 1.1.

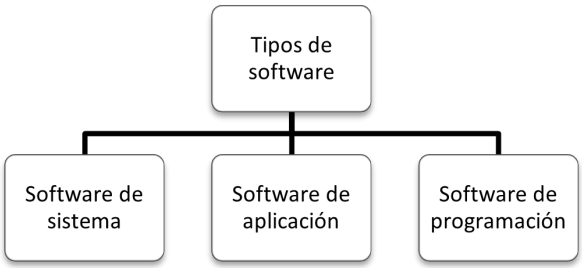


Figura 1.1: Tipos de software.

Software de sistema

El *software* de sistema [5] tiene como propósito ejecutar las partes o dispositivos de *hardware* de la computadora y los programas de aplicación, o bien, es la plataforma que se proporciona al sistema informático para ejecución de otros programas de información. Este tipo de *software* actúa como intermediario entre las aplicaciones de usuario y el *hardware*, ya que permite el uso de este último a través de controladores de dispositivos.

Software de aplicación

El *software* de aplicación [5] ofrece a los usuarios aplicaciones para realizar tareas específicas, dicho de otra forma, ofrece un *software* específico para cada tipo de tarea de acuerdo con la necesidad del usuario. Todo *software* que está diseñado para algún propósito específico se conoce como *software* de aplicación.

Software educativo

Software educativo [6] incluye programas informáticos creados con el propósito específico de utilizarse como herramientas didácticas y facilitar el proceso de enseñanza y aprendizaje. El software educativo motiva la creación de un contexto apto para la construcción y transmisión de conocimiento en el momento que los estudiantes se integran a clases y/o cursos.

Software de programación

Software de programación [5] hace referencia a los programas que utilizan los programadores para escribir código, scripts e instrucciones para ejecutarse por una computadora.

IDE

Un Entorno de Desarrollo Integrado (Integrated Development Environment, por sus siglas en inglés) [7] es un software con interfaz gráfica de usuario donde los programadores codifican instrucciones o algoritmos para un sistema informático en desarrollo. Un IDE proporciona un conjunto de herramientas para acelerar las tareas comunes durante el proceso de desarrollo como compilación, creación o administración de dependencias.

1.1.3. Aprendizaje

Aprendizaje [8] es la adquisición de conocimientos y habilidades a través del estudio de algo.

1.1.4. Tipos de aprendizaje

De acuerdo con [8], existen diferentes tipos de aprendizaje que refieren principalmente a formas diversas de aprender. Es indispensable que los docentes conozcan los estilos de aprendizaje para enriquecer su enseñanza pedagógica.

A continuación, se describen tres tipos de aprendizaje relacionados con la naturaleza del presente tema de investigación.

Aprendizaje explícito

En el aprendizaje explícito [8] el aprendiz muestra interés o tiene la intención de aprender y está consciente de lo que está aprendiendo.

Aprendizaje significativo

En el aprendizaje significativo [8] el individuo recauda información, realiza organización y selección de ésta y la relaciona con conocimiento adquirido previamente, es decir, relaciona nuevos conocimientos con los que ya posee.

Aprendizaje receptivo

En el aprendizaje receptivo [8] el individuo recibe el contenido que necesita alcanzar. En el ámbito educativo ocurre cuando el alumno recibe la explicación del docente, material impreso o información audiovisual, pero es indispensable comprender el contenido para aplicarlo de forma correcta.

1.1.5. Estrategias docentes

De acuerdo con [9] las estrategias docentes se fundamentan en principios psicopedagógicos que reflejan las cuestiones que se plantean los docentes en el proceso educativo para lograr un aprendizaje significativo en los alumnos.

La intención de éstas es aportar criterios que justifiquen la acción didáctica en el aula de instituciones educativas, además de inspirar y guiar al docente en sus actividades de enseñanza para alcanzar los objetivos previstos. Las estrategias requieren ser ajustadas acorde a los esquemas intelectuales del alumnado considerando:

- Los estudiantes necesitan ser orientados a conducir su propio aprendizaje, esto significa pasar de la dependencia a la autonomía.
- Las prácticas de enseñanza-aprendizaje se ocupan más de los procedimientos y las competencias que de los conocimientos estrictos. La enseñanza teórica pierde significado si no se lleva a la práctica.

Por ello, la planificación educativa determina estrategias docentes concretas que constan de los siguientes puntos:

- Tomar como punto de partida la experiencia de los estudiantes, es decir, compensar el aprendizaje de conceptos, procedimientos y actitudes.
- Introducir la interdisciplinariedad.
- Orientar el aprendizaje hacia la solución de los problemas generales por el contexto de los estudiantes más que hacia la adquisición estricta de saberes.

A pesar de lo anterior descrito, existen numerosos condicionantes que suelen llegar a ser producto de anteriores experiencias educativas o de aprendizajes espontáneos e interfieren en el desarrollo personal y académico de los estudiantes. El estudiante inicia el aprendizaje a partir de experiencias previas o de una representación mental que ha ido construyendo a lo largo de su trayectoria educativa, y utiliza ésta como instrumento de interpretación, que condicionan en gran medida el resultado del nuevo aprendizaje.

1.1.6. Lenguaje de programación

Un lenguaje de programación es una notación especial para comunicarse con una computadora, también se define como lenguaje formal o artificial, es decir, un lenguaje con reglas gramaticales muy bien definidas que un programador le proporciona a una computadora en una serie de instrucciones o secuencias con fin de controlar el comportamiento físico o lógico de un sistema informático [10].

Java

Java [11] es un lenguaje de programación y una plataforma de desarrollo que fue lanzada por primera vez por Sun Microsystems® en 1995. Desde su creación ha evolucionado hasta impulsar gran parte del mundo digital, por ser una plataforma confiable sobre la cual se construyen servicios y aplicaciones. Existen aplicaciones e incluso sitios web que no funcionarían sin tener Java instalado. Java es rápido, seguro y confiable.

1.1.7. Proceso de Construcción e Inicialización de Objetos

El proceso de construcción e inicialización de objetos es la serie de pasos que llevan a la creación de un objeto de tipo específico, asignarle un espacio en memoria e inicializarlo a través de un constructor.

Clase y objeto

Una clase es la descripción de un conjunto de objetos, tiene características (elementos de datos) y comportamientos (funcionalidad) idénticos y es un equivalente a un tipo de dato. A partir de la definición de clase, se define objeto como una instancia de clase, tiene estado, comportamiento e identidad. Los términos de objeto e instancia son intercambiables [12].

1.1.8. Modelo de objetos

Booch [13] define el modelo de objetos como una colección de principios que sustentan el diseño orientado a objetos, un paradigma de ingeniería de software que destaca los principios de abstracción, encapsulación, modularidad, jerarquía, tipificación, concurrencia y persistencia.

Elementos mayores

Los elementos mayores comprenden la abstracción, encapsulación, modularidad y jerarquía.

Abstracción

La abstracción [13] denota las características propias de un objeto que lo distinguen de los demás tipos de objetos y, por lo tanto, existen límites conceptuales claramente definidos relacionados con la perspectiva del espectador; en general es el proceso de enfoque de las características esenciales de un objeto de forma particular.

Encapsulación

La encapsulación [13] es el proceso de dividir los elementos de una abstracción que constituyen su estructura y comportamiento; la encapsulación depone la interfaz contractual de una abstracción de su implementación.

Modularidad

La modularidad [13] se define como la división de un programa en varios módulos que se compilan por separado, pero tienen conexiones con más módulos. Por lo tanto, la modularidad es la propiedad de un sistema informático que se desintegra en un conjunto de módulos cohesivos y débilmente acoplados.

Jerarquía

Se define jerarquía [13] como una clasificación u ordenamiento de abstracciones. Las dos jerarquías más relevantes en un sistema complejo son la estructura de clases (la jerarquía "es un") y la estructura de objetos (la jerarquía "parte de").

Elementos menores

Los elementos menores incluyen tipificación, concurrencia y persistencia.

Tipificación

La tipificación [13] es la aplicación de la clase de un objeto, de modo que los objetos de tipos diferentes no se intercambian o, como máximo, se intercambian sólo de formas muy restringidas.

Concurrencia

La concurrencia [13] se define como la propiedad que distingue un objeto activo de uno que no lo está.

Persistencia

La persistencia [13] es la propiedad de un objeto a través de la cual su existencia trasciende el tiempo (el objeto continúa existiendo aún después de que su creador deja de existir) y/o el espacio (la ubicación del objeto se mueve desde el espacio de direcciones en el que se creó).

1.2. Planteamiento del problema

El proceso de construcción e inicialización de objetos en Java es una pieza clave para la comprensión de programas, ejecuciones de escritorio, toma de decisiones sobre errores, comportamiento no deseado de las aplicaciones, mantenimiento de *software* e incluso, facilidad para alcanzar objetivos de certificación, sin embargo, la literatura actual reporta este proceso de forma fragmentada y sólo el libro *Thinking in Java* de Bruce Eckel [12] lo trata en una sola pieza, pero incompleto. Nuevos paradigmas como el orientado a aspectos requieren un conocimiento conciso del proceso de construcción para complementar de forma comprensible el uso de primitivas de corte y facilitar la especificación de patrones de firmas que permitan la coordinación eficiente entre aspectos y objetos.

Con base en lo anterior, es deseable contar con una herramienta de apoyo para el proceso de enseñanza-aprendizaje del proceso de construcción en Java, principalmente en el área académica, por lo que se propone desarrollar un primer módulo de *software* que facilite la comprensión de este proceso considerando temáticas incompletas o no reportadas adecuadamente en la literatura, como son:

- Los propios pasos del PCIO (Proceso de Construcción e Inicialización de Objetos).
- El cambio de orden de inicialización entre campos e inicializadores.
- El efecto del polimorfismo a nivel de objetos ante la presencia de una superclase abstracta y/o implementación de una interfaz.

- La inicialización en las llamadas al súper constructor, conocida en el paradigma de aspectos como pre-inicialización.

1.3. Objetivo general y objetivos específicos

En esta sección se dan a conocer los objetivos tanto general como específicos para resolver el problema planteado.

1.3.1. Objetivo general

Desarrollar un módulo de *software* para apoyar la comprensión de la creación de objetos en Java por medio del Proceso de Construcción e Inicialización de Objetos.

1.3.2. Objetivos específicos

- Analizar los trabajos relacionados con el proyecto para identificar áreas de oportunidad.
- Analizar tecnologías adaptables y óptimas para el desarrollo del proyecto.
- Revisar el proceso de construcción e inicialización de objetos en Java para identificar los detalles no documentados en la literatura.
- Crear un conjunto de heurísticas con base en la documentación del PCIO para identificar los pasos de dicho proceso en una unidad de compilación.
- Desarrollar un módulo de *software* para apoyar la comprensión de la creación de objetos en Java por medio del proceso de construcción e inicialización de objetos.
- Documentar el PCIO en Java de forma completa.
- Llevar a cabo pruebas de implementación para medir el impacto del módulo desarrollado.
- Crear una guía interactiva de estudio basadas en heurísticas y ejemplos del usuario.

1.4. Justificación

Las herramientas que apoyan el proceso de enseñanza-aprendizaje en general permiten propiciar el aprendizaje activo, simplifican las tareas de aprendizaje y permiten una administración más efectiva del tiempo disminuyendo el esfuerzo en preparación de materiales. Particularmente, en la asignatura de Tecnologías de Programación del TecNM, es exigible el manejo adecuado del proceso de construcción e inicialización de objetos para el desempeño adecuado de estudiantes en diversos contextos como su aplicación en otras asignaturas y en la implementación de requerimientos específicos en proyectos y/o tesis. Por otro lado, el curso propedéutico del Campus Orizaba considera el tema de proceso de construcción como fundamental en la preparación de los estudiantes de nuevo ingreso a la maestría. Adicionalmente y en general, el proceso de construcción e inicialización de objetos es altamente relevante para cualquier discente que estudia el lenguaje Java, ya sea por formación académica o por interés en obtener una certificación.

Capítulo 2

Estado de la práctica

A continuación, se presenta un resumen de los trabajos de investigación más importantes que tienen relación directa, indirecta y/o parcial o total con la tesis propuesta.

2.1. Trabajos relacionados

En [14] se analizaron los problemas y soluciones más relevantes que suelen encontrarse en la literatura sobre la enseñanza y el aprendizaje de la POO (Programación Orientada a Objetos), el análisis se basó en estudios terciarios de los repositorios IEEE Xplore®, Scopus®, ACM Digital Library® y Science Direct®. Los problemas y soluciones identificados se clasificaron mediante métodos de decisión multicriterio DEMATEL (*Decision Making Trial and Evaluation Laboratory*) y TOPSIS (*Technique for Order of Preference by Similarity to Ideal Solution*) con el fin de determinar las mejores soluciones a los problemas encontrados y aplicar estos resultados en el contexto académico.

El principal aporte de este estudio fue la clasificación de problemas y soluciones de la POO, así como la propuesta de estrategias para mejorar el problema. Entre los problemas más relevantes se encontró: primero, la dificultad para comprender, enseñar e implementar la orientación a objetos; segundo, las dificultades relacionadas con la comprensión de las clases y tercero, la dificultad para comprender las relaciones orientadas

a objetos. Después de realizar el análisis multicriterio, se encontró que las soluciones más importantes para enfrentar los problemas que se encuentran durante la enseñanza de la POO fueron: 1) uso de técnicas de aprendizaje activo y recompensas intrínsecas y 2) enfatizar en conceptos básicos de programación e introducir la POO en un punto temprano del currículum.

De acuerdo con los resultados obtenidos al aplicar las técnicas multicriterio, el problema "Dificultades relacionadas con la comprensión de las clases" se identificó como causa principal en el método DEMATEL y tuvo un alto valor de ponderación. Esto indica la importancia de enfatizar el tema en clase y generar bases conceptuales adecuadas para los estudiantes de programación. Con base en los resultados del método TOPSIS, se encontró que la solución mejor calificada es "Estrategias para el uso de técnicas de aprendizaje activo y recompensas intrínsecas". Este hallazgo refuerza la necesidad de un cambio en las estrategias de enseñanza de la POO, por tal motivo, el uso de herramientas relacionadas con los juegos de computadora y la búsqueda de nuevas estrategias didácticas que motiven a los estudiantes en su proceso formativo apoyan significativamente el aprendizaje de la POO.

Actualmente, el lenguaje de programación orientado a objetos Java, se utiliza en muchos sistemas prácticos, incluidos servidores empresariales, teléfonos inteligentes y sistemas integrados al proveer alta seguridad y portabilidad. Así, los institutos educativos ofrecen cursos de programación Java para impulsar a los estudiantes aprendices de este lenguaje. Por lo anterior, en [15] se desarrolló el Sistema Asistente de Aprendizaje de Programación Java (JPLAS, *Java Programming Learning Assistant System*) basado en la web, este sistema utiliza el método de desarrollo basado en pruebas (TDD, *Test-Driven Development*), para mejorar los efectos educativos de la programación en Java, ayudar a los estudiantes de la gramática básica de Java a ser autodidactas y al mismo tiempo reducir la carga de trabajo en los docentes.

En JPLAS, como primer paso, el docente registra una tarea de programación Java con una declaración, un código fuente modelo y un código de prueba, posteriormente un

estudiante escribe un código fuente, lee la declaración y el código de prueba de modo que el código fuente se ejecuta automáticamente en el servidor con JUnit, una herramienta para el método TDD. Se comprobó la efectividad de JPLAS a través de aplicaciones experimentales a estudiantes en el Departamento de Ciencias de la Información y la Computación de la Universidad de Osaka, Japón.

Otra herramienta complementaria del sistema desarrollado en [15], se basó en la web como solución al problema de escritura de código que permite a los estudiantes analizar la escritura de códigos fuente para tareas. En un lenguaje de programación orientado a objetos, la encapsulación, la herencia y el polimorfismo son los tres conceptos fundamentales que se requiere que todos los estudiantes dominen. En [16] se propuso el enfoque de código de prueba informativo para el problema de escritura de código, el cual ayuda a los estudiantes a analizarlos y describir la información necesaria para el código. Para las evaluaciones se generaron códigos de prueba informativos para 10 tareas, después se solicitó a 10 estudiantes que escribieran algunos códigos fuente que fueron completados con alta calidad al utilizar los conceptos.

Desafortunadamente, la mayoría de los docentes en las escuelas no están acostumbrados a escribir códigos de prueba. Como complemento a la herramienta que se desarrolló en [15], se propuso otra herramienta de generación de código de prueba [17] que genera automáticamente los casos de prueba a partir del código fuente de referencia. Se extraen salidas para entradas dadas con el uso de funciones en JUnit. Como tareas para estudiantes principiantes de Java, se enfatiza el código que contiene entradas/salidas estándar. Para la evaluación, se recopilaron 97 códigos que contenían entradas/salidas estándar de libros de texto o sitios web de programación Java. El resultado experimental demostró que la herramienta propuesta generó correctamente los códigos de prueba, excepto un código que utilizó un generador aleatorio.

Así también muchos estudiantes que provienen de diferentes áreas de conocimiento encuentran que la programación de software es un tema difícil de aprender y dominar, especialmente en el aprendizaje de conceptos de POO. Los estudiantes son capaces de

modelar cosas físicas en objetos virtualizados y definir relaciones de objetos complejas en sus diseños para interacciones de objetos de una manera muy abstracta, lo cual para algunos es difícil entender. Por ello, en [18] se presentó una plataforma de aprendizaje interactiva para estudiantes en proceso de aprendizaje del lenguaje Java, la cual se diseñó con un conjunto de consideraciones pedagógicas de POO. Se presentó una nueva plataforma y entorno interactivo en línea llamado BlueJ-UML, que ayuda a los estudiantes a aprender y practicar la POO en clase. También se evaluó el resultado de éxito del nuevo método de aprendizaje propuesto a través de un marco de modelo de aceptación de tecnología, seguido de un análisis estadístico integral para evaluar el rendimiento académico de los estudiantes después de utilizar BlueJ-UML. El resultado fue alentador: las habilidades de programación de los estudiantes se elevaron significativamente y se correlacionó positivamente con su percepción general y adopción de esta nueva técnica presentada en clase.

En la actualidad, los compiladores modernos son muy útiles, no solo para detectar errores de programación, sino también para sugerir posibles soluciones para esos errores utilizando técnicas de reparación de errores. La reparación de errores se refiere al proceso de encontrar una reparación para un error dado que ocurre como resultado de la compilación de un código escrito por programadores. Sin embargo, una reparación no siempre es consistente con el propósito del código escrito, esto significa que un compilador sugiere una reparación diferente de la instrucción que el programador quiso escribir. Muchos compiladores no sugieren la reparación correcta cuando se producen errores de programación como resultado de errores ortográficos.

Como consecuencia al problema anterior, el objetivo de [19] fue mejorar el proceso de reparación de errores en compiladores utilizando algoritmos de corrección ortográfica, ya que un compilador regular no proporciona soluciones para los errores de programación de sintaxis más comunes que ocurren como resultado de errores ortográficos. Los errores son fáciles de detectar por los compiladores, pero es difícil sugerir una solución que tenga la forma correcta, por lo tanto, se desarrolló un nuevo mecanismo de corrección de errores que se denomina enfoque CoEdit para en conjunto con los compiladores

sugerir la reparación más adecuada para los errores de programación que se produjeron como resultado de errores de escritura. Se emplearon los algoritmos Four-Way y Editex de corrección ortográfica compatibles con los lenguajes de programación para encontrar reparaciones a los errores de ortografía.

CoEdit es un enfoque genérico para mejorar la reparación de errores en cualquier compilador para cualquier lenguaje de programación porque se enfoca en errores ortográficos del compilador. Este trabajo concluyó que utilizar el algoritmo Editex con CoEdit produjo mejores resultados al acertar o encontrar reparaciones a errores de programación que se produjeron como consecuencia de errores ortográficos de prueba.

En [20] se presentó la herramienta jCAB (*Java Class Auto Builder*) que ayuda a los estudiantes a diseñar clases personalizadas en Java, a comprender los conceptos básicos en el diseño de la clase y a desarrollar ideas orientadas a objetos de manera eficiente. Cuando los estudiantes comienzan a aprender POO, al inicio los conceptos fundamentales son difíciles de aprender e impactan cuando comienzan a diseñar sus plantillas de clase personalizadas.

Las herramientas basadas en UML actuales sirven para fines educativos, pero no son adecuadas para estudiantes o programadores principiantes de Java, esas herramientas generan código de plantilla de clase automáticamente a partir de diagramas UML que representan claramente las estructuras y funcionalidades de clase, lo cual requiere que los usuarios tengan una visión completa de las estructuras de clase, sin embargo, cuando los estudiantes principiantes indagan por primera vez el diseño de clases personalizadas, generalmente no conocen lo necesario para el diseño y/o las relaciones entre las clases (si las hay); por lo tanto, es difícil para ellos desarrollar diagramas UML que representen claramente las plantillas de clase. Dicho lo anterior, es necesaria una herramienta que ayude a los alumnos en esta etapa.

La herramienta jCAB que se presentó fue diseñada para el curso Java CS1 y fue un gran comienzo para que los programadores principiantes aprendan a implementar clases

personalizadas. La herramienta divide el procedimiento de diseño de clase personalizado en varios pasos para mostrar a los estudiantes lo necesario en el diseño de clase de forma clara y muestra cómo hacerlo a detalle. Su Interfaz Gráfica de Usuario interactúa con los estudiantes, guía paso a paso, genera código en cada paso de forma automática para que los estudiantes comprendan la estructura de una clase, muestra los procedimientos de diseño e indica cómo cada pieza de código se relaciona con conceptos de orientación a objetos. De esta forma se comprenden mejor los fundamentos y los procedimientos de diseño de clases personalizadas.

La POO juega un papel importante en el desarrollo de software moderno, sin embargo, a la mayoría de los estudiantes les resulta difícil aprender bien los conceptos de POO porque suelen ser abstractos y difíciles de imaginar, por lo tanto, el problema apremiante es determinar cómo facilitar de manera eficiente el aprendizaje. Se demostró que las herramientas de programación gráfica benefician el aprendizaje de conceptos de programación estructurada, por ejemplo, Scratch y Blockly. En consecuencia, en [21] se diseñó y desarrolló una herramienta de aprendizaje para la POO visualizada basada en la web (VLT-OOP) para facilitar el aprendizaje de conceptos de la POO.

VLT-OOP visualiza los conceptos importantes y abstractos como clase, objeto, método, encapsulación, herencia y polimorfismo para ayudar a los estudiantes a comprender visualmente el significado a través del desarrollo de aplicaciones gráficas 2D basadas en la Web utilizando los objetos visualizados y operaciones. En consecuencia, se espera que la motivación y el rendimiento del aprendizaje de POO superen el enfoque de programación tradicional basado en texto.

Teniendo en cuenta la alta complejidad y la abstracción de los conceptos de POO, se presentó la necesidad de una solución innovadora [22] que sirva como medio para aprender estos conceptos. Se utilizó *Unity*, un motor de juego 3D multiplataforma, se creó la parte interactiva de la aplicación como un juego de arrastrar y soltar en el que el usuario crea una solución para un problema dado utilizando el conocimiento que se le presentó en la introducción de la aplicación. Una vez que el usuario finaliza el desafío,

es indispensable que responda un cuestionario para verificar lo aprendido. En cada nivel hay varias preguntas para conceptos particulares, sobre las cuales el usuario necesita responder, en esta última parte el usuario evalúa el conocimiento adquirido.

La complejidad del código y la POO es un tema relevante porque se aplica en muchos de los diseños y arquitecturas de software hoy en día. En la POO hay conceptos clave de diseño como encapsulación, polimorfismo y herencia que afectan el diseño, la estructura y el estilo de la codificación. El desafío es cómo minimizar la complejidad en la POO y cumplir con los conceptos clave del diseño de esta. En [23] se revisó la literatura sobre las soluciones actuales para la complejidad del código y se propuso un nuevo modelo para la complejidad del código de la POO. El modelo propuesto se basó en los siguientes criterios de selección de atributos: legibilidad, comprensibilidad, mantenibilidad, reutilización, extensibilidad y consistencia del código de programación.

Durante los cursos de introducción a la programación, los estudiantes experimentan una variedad de emociones. Los estudiantes a menudo experimentan ansiedad y frustración cuando encuentran dificultades para escribir programas. La frustración constante desalienta a los estudiantes a continuar sus estudios de ingeniería e informática. Aunque investigaciones anteriores demuestran cómo las emociones afectan la motivación y el aprendizaje de los estudiantes, se sabe poco sobre las emociones de los estudiantes en los cursos de programación.

En [24] se realizó un estudio cualitativo de estudiantes de primer grado de ingeniería que tomaron un curso introductorio de programación, se examinaron las emociones que experimentan estos estudiantes durante las tareas de programación y las razones para experimentar esas emociones. El estudio se basó en la teoría del valor de control de las emociones de logro donde cada participante de la investigación asistió a dos sesiones de laboratorio: una sesión de programación y una sesión retrospectiva de entrevistas de pensamiento en voz alta. En la sesión de programación, cada participante trabajó individualmente en problemas de programación, se recopilaron capturas de pantalla, datos biométricos y respuestas a encuestas. En la sesión de entrevistas, cada participante

vio un video de sus acciones durante la sesión de programación y después de cada dos minutos de visualización, los participantes informaron las emociones que habían experimentado durante este período de dos minutos.

Se realizó un análisis temático de los datos de las entrevistas y los resultados indican que los participantes experimentaron frustración con mayor frecuencia. A veces experimentaban múltiples emociones, por ejemplo, una participante se sintió molesta porque había cometido un error, pero sintió alegría y orgullo cuando corrigió el error. Para promover el aprendizaje de los estudiantes, los educadores necesitan tener en cuenta las emociones de los estudiantes en el diseño del plan de estudios y la pedagogía para cursos introductorios de programación.

En general, estudiantes enfrentan varias dificultades en los cursos de introducción a la programación, lo que a menudo conduce a altas tasas de deserción, desmotivación de los estudiantes y falta de interés. La literatura indica que el uso adecuado de la gamificación mejora el aprendizaje en varias áreas, sin embargo, falta la comprensión de qué (y cómo) factores influyen en el éxito de la gamificación, especialmente para el aprendizaje de programación; por tal motivo, existe una clara necesidad de entender los factores que generan impacto en la importancia de la gamificación. Para abordar este tema, en [25] se realizó una investigación de cómo los factores contextuales y del usuario influyen en el efecto de la gamificación en los estudiantes de CS1 a través de un estudio retrospectivo cuasi experimental (N = 399), basado en un diseño entre sujetos (condiciones: gamificado o no gamificado) en términos de calificación final (rendimiento académico) y el número de asignaciones de programación completadas en un sistema educativo (es decir, cuánto practicaron).

Después, se evaluaron si las características contextuales y del usuario (como la edad, el género, la especialización, la experiencia en programación, la situación laboral, el acceso a Internet y el acceso/uso compartido de computadoras) moderan ese efecto y cómo lo hacen. Los hallazgos indicaron que la gamificación amplificó hasta cierto punto el impacto de la práctica. En general, los estudiantes que practicaban en la versión

gamificada presentaron un mayor rendimiento académico que los que practicaban la misma cantidad en la versión no gamificada. Curiosamente, el grupo de aquellos en la versión gamificada que practicaron mucho más que el promedio mostró logros académicos más bajos que el grupo de aquellos que practicaron cantidades comparables en la versión no gamificada. Además, los resultados revelaron que el género es el único moderador estadísticamente significativo del efecto de la gamificación: en los datos, fue positivo para las mujeres, pero no significativo para los hombres. Estos hallazgos sugieren qué (y cómo) los factores personales y contextuales moderan los efectos de la gamificación, indican la necesidad de comprender y examinar más a fondo el papel del contexto, y muestran que la gamificación tiene que diseñarse con cautela para evitar que los estudiantes jueguen en lugar de aprender.

Por otro lado, la generación de comentarios automatizada para tareas de programación introductorias es útil para el aprendizaje. La mayoría de los trabajos intentan generar retroalimentación para corregir el código fuente (programa) de un estudiante comparando su comportamiento con el programa de referencia de un instructor en pruebas seleccionadas. En [26], el objetivo fue generar reparaciones del programa comprobablemente correctas como retroalimentación a los estudiantes. Un programa enviado por el estudiante se alinea y se compone con una solución de referencia en términos de flujo de control, y las variables de los dos programas se alinean automáticamente a través de predicados que describen la relación entre las variables. Cuando falla el intento de verificación del programa alineado que se obtuvo, se convierte un problema de verificación en un problema de MaxSMT cuya solución conduce a una reparación mínima.

Se realizaron experimentos con tareas de estudiantes seleccionadas a partir de un sistema de tutoría inteligente ampliamente implementado. Los resultados mostraron que es posible generar una reparación verificada sin sacrificar la tasa de reparación general, incluso, la implementación Verifix supera a Clara, una herramienta de última generación, en términos de tasa de reparación. Esto último demuestra la promesa de utilizar la reparación verificada para generar retroalimentación de alta confianza en

entornos de enseñanza de programación.

Finalmente, en [27] se tuvo como principal objetivo estudiar las prácticas actuales de documentación del código y analizar la literatura existente para brindar una perspectiva sobre su preparación para abordar dicho problema y los desafíos que se avecinan. Contexto: la codificación es una actividad incremental en la que un desarrollador necesita comprender un código antes de realizar los cambios adecuados en él. La documentación del código se considera una de las buenas prácticas en el desarrollo de software, pero requiere un esfuerzo significativo por parte de los desarrolladores.

Los avances recientes en el procesamiento del lenguaje natural y el aprendizaje automático han proporcionado suficiente motivación para diseñar enfoques automatizados para la documentación del código fuente en múltiples niveles. Metodología: Se proporcionó una descripción detallada de la literatura en el área de la documentación del código fuente automatizado en diferentes niveles y se analizó críticamente la efectividad de los enfoques propuestos. Esto también permitió inferir brechas y desafíos para abordar el problema en diferentes niveles. Hallazgos: 1) La comunidad de investigación se centró en el resumen a nivel de método. 2) El aprendizaje profundo ha dominado los últimos cinco años de este campo de investigación. 3) Los investigadores proponen regularmente *corpus* más grandes para la documentación del código fuente. 4) Java y Python son los lenguajes de programación ampliamente utilizados como *corpus*. 5) BLEU es la métrica de evaluación más favorecida por los investigadores.

2.2. Análisis del estado de la práctica

La tabla 2.1 contiene información comparativa de cada uno de los trabajos relacionados con el proyecto de tesis con el fin de observar las similitudes y diferencias más importantes.

Tabla 2.1: Análisis comparativo de los artículos relacionados.

Artículo	**Problema**	**Contribución**	**Herramientas**	**Resultados**	**Estado**
Gutierrez et al. [14].	Es necesario analizar la literatura e identificar problemas y soluciones para la enseñanza de la POO.	Se identificaron soluciones a problemas en la enseñanza de POO de acuerdo con planteamientos por expertos en el área de desarrollo.	- IEEE Xplore® - Scopus® - ACM Digital Library® - Science Direct® - VOSviewer - DEMATEL - TOPSIS	Se determinó la estrategia que mejoraría este problema: uso de técnicas de aprendizaje activo y recompensas intrínsecas.	Concluido y con planes de dar seguimiento.
Funabiki et al. [15].	Se requiere impulsar a los estudiantes en su aprendizaje de POO.	Se desarrolló un sistema asistente de programación Java (JPLAS).	- JSP/-Servlet - Tomcat - MySQL - Ubuntu Server	Se comprobó la efectividad de JPLAS a través de aplicaciones experimentales a estudiantes.	Concluido.

Continúa en la siguiente página

Tabla 2.1 – continuación de la página anterior

Artículo	Problema	Contribución	Herramientas	Resultados	Estado
Zaw et al. [16].	Es necesario un enfoque de código de prueba informativo para el problema de escritura de código.	Se generaron códigos de prueba informativos.	Eclipse IDE	Se puso a prueba el enfoque con diez estudiantes y diez tareas de código que se completaron correctamente al visualizar los códigos de prueba.	Concluido.
Funabiki et al. [17] .	Es necesaria una herramienta para generación de códigos de prueba.	Una plataforma web para proporcionar códigos de prueba informativos para escribir correctamente las tareas de código.	- JSP/Java - Linux - Apache - Tomcat - MySQL - Firefox - HTML, CSS - JavaScript	Se demostró que los estudiantes completaron códigos fuente de alta calidad con información de códigos de prueba.	Propuesta.

Continúa en la siguiente página

Tabla 2.1 – continuación de la página anterior

Artículo	Problema	Contribución	Herramientas	Resultados	Estado
Keung et al. [18] .	Es necesaria una plataforma interactiva de aprendizaje Java para estudiantes en clase.	Una plataforma y entorno interactivo en línea llamado BlueJ-UML , que ayuda a los estudiantes a aprender y practicar la POO en clase.	- BlueJ - UML	Se mostró que las habilidades de programación de los estudiantes se elevaron significativamente y se correlacionó positivamente con su percepción general y adopción de esta nueva técnica presentada en clase.	Propuesta con seguimiento.
Alwabel et al. [19] .	La falta de un mecanismo de corrección de errores de programación en sintaxis más comunes como resultado de errores ortográficos.	Un mecanismo de reparación de errores de programación para cualquier compilador y en cualquier lenguaje.	- Four Way algorithm - Editex Algorithm - CoEdit	Se mejoró la precisión de soluciones al encontrar errores de programación en un análisis de tres experimentos con CoEdit .	Concluido y en busca de mejoras.

Continúa en la siguiente página

Tabla 2.1 – continuación de la página anterior

Artículo	Problema	Contribución	Herramientas	Resultados	Estado
Liu et al. [20] .	Es indispensable que los estudiantes principiantes comprendan los conceptos fundamentales de la POO.	Herramienta para diseño de clases personalizadas en Java .	- UML - Java Swing	Se probó la herramienta con 150 estudiantes universitarios que aprenden Java y se demostró que después de tres semestres de uso, el porcentaje de error disminuyó notablemente.	Concluido.
Su et al. [21] .	Falta de herramientas para facilitar el aprendizaje de la POO.	Una herramienta de aprendizaje de programación orientada a objetos visualizada basada en la web (VLT-OOP).	- Java - HTML5 - JavaScript	Se logró la visualización de conceptos importantes y abstractos de POO, es decir, clase, objeto, método, encapsulación, herencia y polimorfismo.	Concluido.

Continúa en la siguiente página

Tabla 2.1 – continuación de la página anterior

Artículo	Problema	Contribución	Herramientas	Resultados	Estado
Kucera et al. [22] .	Los juegos interactivos son una herramienta funcional para el aprendizaje de principios de POO.	Aplicación de juego 2D para dar solución a problemas por medio de conocimiento adquirido en la introducción del juego.	- Unity engine	Se demostró que los usuarios tuvieron una experiencia de aprendizaje dinámica y divertida al pasar por los tres niveles del juego.	Concluido y en busca de mejoras.
Hourani et al. [23] .	Falta de mecanismo para minimizar la complejidad de la POO sin incumplir con los conceptos claves de diseño.	Un modelo que agrega métricas de complejidad de las siguientes características: abstracción y complejidad de detalles de clase.	- Chidamber metric - Kemmerer metric - DIT, NOC, CBO, y COM	Se logró completar un modelo para medir la complejidad del aprendizaje de la POO para utilizarlo en trabajos futuros con prácticas reales.	Propuesta con seguimiento.

Continúa en la siguiente página

Tabla 2.1 – continuación de la página anterior

Artículo	Problema	Contribución	Herramientas	Resultados	Estado
Atiq et al. [24] .	Falta de comprensión de las emociones experimentadas por los estudiantes durante su aprendizaje de programación.	Proporciona una comprensión cualitativa respaldada por la teoría de las emociones que experimentan los estudiantes.	- MATLAB - Control-value theory (CVT)	Se demostró que los participantes experimentaron frustración con mayor frecuencia, por ello los educadores requieren tomar esto en cuenta para el diseño del plan de estudios	Concluido.
Rodrigues et al. [25] .	Altas tasas de deserción, falta de interés desmotivación de los estudiantes al enfrentar dificultades en los cursos de programación.	Se realizó una investigación de cómo los factores contextuales y del usuario influyen en el efecto de la gamificación en los estudiantes.	- CodeBench	Los resultados respaldan el efecto de las pruebas en el contexto del estudio y muestran que los estudiantes en la versión gamificada obtuvieron un mayor rendimiento académico.	Concluido.

Continúa en la siguiente página

Tabla 2.1 – continuación de la página anterior

Artículo	Problema	Contribución	Herramientas	Resultados	Estado
Ahmed et al. [26] .	Falta de una herramienta para la generación de comentarios automatizada.	Se desarrolló Verifix , una herramienta para sugerir reparaciones de código como retroalimentación a estudiantes.	- Lenguaje C - Maximum Satisiability-Modulo-Theories (MaxSMT)	Se demostró que es posible generar una reparación verificada, incluso, la implementación Verifix supera a Clara, una herramienta de última generación, en términos de tasa de reparación.	Concluido.
Rai et al. [27] .	Falta de conocimiento sobre prácticas de documentación de código.	Análisis de las prácticas actuales de documentación del código y literatura para abordar los problemas y desafíos.	- IEEE Xplore (IEEE) - ACM Digital - Library (ACM) - ScienceDirect-Elsevier - SpringerLink - Arxiv - ACLWeb	El principal hallazgo es que los investigadores se centraron en la generación de resúmenes en lenguaje natural y la predicción de nombres de métodos.	Concluido.

Al realizar un análisis de los diferentes artículos de investigación, se concluye que ninguno de ellos considera directamente el proceso de construcción e inicialización de

objetos, por ello se considera esto como un área de oportunidad para el presente trabajo de investigación y la importancia que esto conlleva, además de que es importante conocer los principales problemas que presentan (principalmente estudiantes del área de programación) durante su aprendizaje.

Si bien existen sistemas o programas eficientes, al comprender de forma correcta el proceso de construcción estos sistemas se vuelven aún más eficientes y al programador le resulta más fácil el desarrollo, es decir, se realiza la codificación de forma más consciente, minimizando así fuentes de error.

2.3. Solución propuesta

A continuación, se presenta la propuesta de solución, el conjunto de las tecnologías seleccionadas que dió solución a este trabajo y la metodología de desarrollo que permitió alcanzar los objetivos de la investigación.

2.3.1. Herramientas tecnológicas utilizadas

- Java, el lenguaje de programación con flexibilidad para tratar la construcción de objetos [1].
- La biblioteca *Blockly* para el editor de código basado en bloques [28].
- React, para crear las interfaces de usuario [29].
- Metodología en cascada, como proceso de desarrollo secuencial [30].
- ItelliJ IDEA como entorno de desarrollo [31].

2.3.2. Justificación de la solución seleccionada

La elección de la solución se basa en la accesibilidad, el desempeño y compatibilidad de las tecnologías que incluye.

Lenguaje de programación

Java, dada la naturaleza del proyecto, el aprendizaje del proceso de construcción e inicialización de objetos está enfocado directamente a este lenguaje.

Biblioteca para bloques de código

Una biblioteca de Google, 100 % desarrollada en JavaScript para generar códigos de programación de forma visual a través de editores basados en bloques. La implementación de Blockly se llevó a cabo a través de React como marco de trabajo. La selección de Blockly se debe a que es altamente personalizable para utilizarse con cualquier lenguaje de programación, además de que se utiliza común y específicamente en proyectos con fines educativos [28].

Marco de trabajo web

React, la biblioteca de JavaScript para construir interfaces de usuario interactivas de forma sencilla y rápida. Se diseñan vistas simples para cada estado en la aplicación, y React se encarga de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien [29]. XP como estilo de desarrollo se centra en aplicación de técnicas de programación, clara comunicación y trabajo en equipo.

Metodología de desarrollo de software

Metodología en cascada, por ser un proceso de gestión de proyectos sencillo y bien definido [30]. Los requisitos se establecen claramente desde el inicio. El modelo en cascada se basa en una planificación exhaustiva previa en la que cada fase se concluye por completo antes de comenzar la siguiente.

IDE

IntelliJ IDEA [31] desarrollado por JetBrains, contiene las funcionalidades necesarias para completar los alcances del proyecto, el IDE brinda recomendaciones inteligentes y destacadas en cada contexto como finalización instantánea de código, análisis de código sobre la marcha, fácil navegación del proyecto y herramientas de recomposición. Además de Java, soporta muchos otros lenguajes de programación a través de plugins, lo que lo hace más atractivo que el resto.

Las tecnologías descritas en esta sección se consideraron suficientes para utilizarse durante el desarrollo del proyecto que se describe en el capítulo 3 en la fase de implementación.

Capítulo 3

Aplicación de la metodología

Este capítulo tiene como objetivo mostrar el uso de la metodología en cascada [30], la cual se utilizó durante el desarrollo de este proyecto de tesis.

Dicha metodología se divide en cinco fases fundamentales:

- Requerimientos: se realiza la recolección y organización de ideas para especificar y recopilar los requisitos del módulo de software.
- Diseño: se plantea una solución técnica, comprende el boceto simple y conciso de las interfaces, del servicio web y de la arquitectura del módulo.
- Implementación: comprende el desarrollo del módulo de aprendizaje mediante el uso de las tecnologías descritas en el la sección 2.3 de este documento.
- Verificación o pruebas: el software desarrollado se somete a una serie de pruebas que ayudan a encontrar y corregir fallas.
- Despliegue y mantenimiento: no aplica para este proyecto, sin embargo, se consideró dentro de los trabajos futuros.

La figura 3.1 muestra el proceso secuencial entre las fases del modelo en cascada.

Figura 3.1: Fases de la metodología en cascada.
Fuente: Creación no propia.

3.1. Requerimientos

En esta fase de la metodología se encuentran las historias de usuario, que permitieron el diseño del módulo de *software*. A continuación, se presenta en forma de tablas las historias de usuario.

Tabla 3.1: Historia de usuario 1

Historia de usuario	
Número: **1**	
Nombre de la historia: **Consulta de pasos del PCIO**	
Prioridad: **Media** (Alta, Media, Baja)	Riesgo en Desarrollo: **Baja** (Alta, Media, Baja)
Puntos Estimados: **1**	Iteración Asignada: **1**
Descripción: **Los usuarios podrán consultar los pasos ordenados del PCIO.**	
Observaciones: Ninguna	

Tabla 3.2: Historia de usuario 2

Historia de usuario	
Número: **2**	
Nombre de la historia: **Generación de código fuente**	
Prioridad: **Alta** (Alta, Media, Baja)	Riesgo en Desarrollo: **Media** (Alta, Media, Baja)
Puntos Estimados: **1**	Iteración Asignada: **1**
Descripción: **Los usuarios generarán código fuente a través del ensamblado de bloques visuales.**	
Observaciones: Ninguna	

Tabla 3.3: Historia de usuario 3

Historia de usuario	
Número: **3**	
Nombre de la historia: **Validación sintáctica de código fuente**	
Prioridad: **Alta** (Alta, Media, Baja)	Riesgo en Desarrollo: **Media** (Alta, Media, Baja)
Puntos Estimados: **1**	Iteración Asignada: **1**
Descripción: **Una vez que el usuario genere el código necesario, podrá enviarlo al sistema de validaciones para verificar que contenga la estructura correcta.**	
Observaciones: Ninguna	

Tabla 3.4: Historia de usuario 4

Historia de usuario	
Número: **4**	
Nombre de la historia: **Obtención de resumen de ejecución**	
Prioridad: **Alta** (Alta, Media, Baja)	Riesgo en Desarrollo: **Media** (Alta, Media, Baja)
Puntos Estimados: **1**	Iteración Asignada: **1**
Descripción: **La obtención de resultados será cada vez que el usuario envíe código fuente para validar, comparar respuestas y obtener un resumen de la ejecución de acuerdo a los pasos del PCIO.**	
Observaciones: Ninguna	

La validación de código se realizará desde el sistema de validaciones en tiempo de ejecución, posteriormente se arrojará al usuario un resumen de ejecución de acuerdo con los pasos del proceso de construcción.

Para la representación de bloques de código necesarios para el aprendizaje del PCIO se optó por una programación en tiempo de diseño a través del ensamblado de bloques. La idea de generar el código fuente de forma manual con el uso de bloques también tiene como propósito que el usuario conozca la estructura del código fuente y lo manipule fácilmente arrastrando y soltando elementos. El usuario descargará el código fuente si así lo requiere para su posterior análisis.

3.2. Diseño del módulo

En esta sección se describe la arquitectura y componentes del módulo, los cuales cumplen con los requisitos para dar solución al problema.

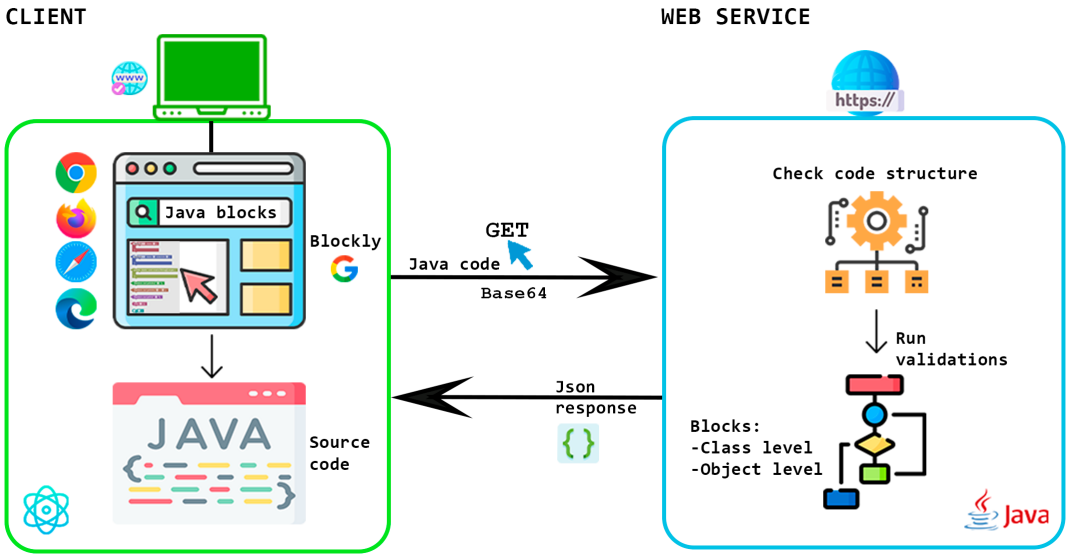


Figura 3.2: Arquitectura del módulo de software.

La comunicación entre la aplicación cliente y el servicio web se muestra en la figura 3.2 donde a través del estilo arquitectónico Cliente-Servidor, se establece un proveedor (*web service*) y un consumidor (*client*).

3.2.1. Arquitectura del módulo

A continuación, se describen las partes que conforman la arquitectura del módulo.

Cliente

Se encuentra la Interfaz Gráfica de Usuario, aquí se genera el código fuente necesario para enviarse a validar.
La creación de bloques de código se realiza a través de Blockly, una biblioteca de Google, 100 % codificada en JavaScript para generar códigos de programación de forma visual a través de editores basados en bloques. La implementación de Blockly se llevó a cabo a través de React [29] como marco de trabajo.
El cliente consume el servicio web por medio del URL definido y el protocolo HTTP, se realiza el envío de código cifrado en base64 y dado que viaja en formato JSON a través del método GET, con el cifrado se evita la inconsistencia de código debido a corchetes y llaves que este incluya.

Servicio Web

Provee un *endpoint* para establecer comunicación con el cliente, aquí es donde el código fuente se procesa, se realizan validaciones de estructura y sintaxis de código y se llevan a cabo comprobaciones línea por línea para identificar los pasos del proceso de construcción aplicados durante la ejecución:

1. ¿Hay herencia en clase principal (clase pública)?
    a) ¿Hay herencia en la clase padre?
    b) Variables y métodos estáticos.
2. Variables y métodos estáticos de la clase principal.
3. ¿Se generan objetos nombrados o anónimos de otra clase?
4. Inicializadores.

Finalmente se realiza la ejecución del código a través de JVM para obtener la salida correspondiente y devolver al usuario una respuesta en formato JSON con un resumen del código analizado.

3.2.2. Componentes del módulo

En este apartado se describen los componentes principales definidos para el módulo.

Bloques de código

Para la correcta interacción del usuario con el módulo se generaron bloques específicos y se personalizaron de acuerdo con lo que se pretende apoyar en cada paso del proceso de construcción, se contemplaron tres categorías: general, a nivel clase y a nivel objeto. La definición de bloques se describe en las tablas 3.5, 3.6 y 3.7.

Tabla 3.5: Bloques de código generales.

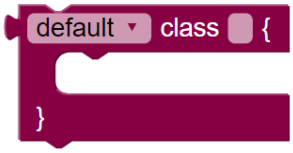
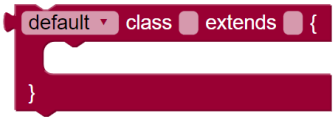
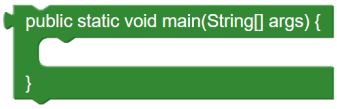
Bloque	Forma visual	Descripción
Bloque de clase.		Genera el cuerpo de la clase con modificador de acceso y nombre personalizados.
Bloque de clase con herencia.		Genera el cuerpo de la clase con la opción para heredar de otra clase.
Bloque de método *main*.		Genera el cuerpo vacío del método de arranque de la aplicación.
		Continúa en la siguiente página

Tabla 3.5 – continuación de la página anterior

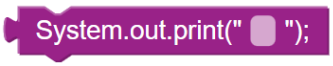
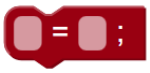
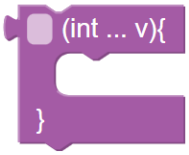
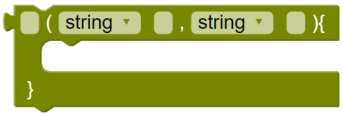
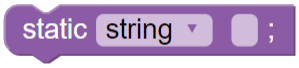
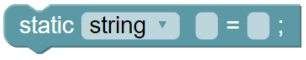
Bloque	Forma visual	Descripción
Bloque de impresión en consola.		Agrega la línea de impresión con texto personalizado con salto de línea.
Bloque de impresión en consola.		Agrega la línea de impresión con texto personalizado sin salto de línea.
Bloque de invocación al super constructor.		Agrega la línea de super() para enviar valores como parámetros.
Bloque de objeto anónimo.		Agrega línea de objeto anónimo con nombre personalizado.
Bloque de invocación a método.		Agrega línea de código para invocar a un método a través de un objeto nombrado.
Bloque de acceso a posición de arreglo.		Agrega línea de código para hacer referencia a la posición de un arreglo (del 1 al 10).
Bloque de asignación de valor.		Agrega línea de código para asignar valor a una variable u objeto.

Tabla 3.6: Bloques de código a nivel clase o estáticos.

Bloque	Forma visual	Descripción
Bloque de constructor.		Genera el cuerpo del construtor con argumentos de longitud variable.
Bloque de constructor.		Genera el cuerpo del construtor sin parámetros.
Bloque de constructor.		Genera el cuerpo del construtor con un parámetro.
Bloque de constructor.		Genera el cuerpo del construtor con dos parámetros.
Bloque de variable de clase.		Agrega línea de variable con valor default.
Bloque de variable de clase.		Agrega línea de variable con valor explícito.

Continúa en la siguiente página

Tabla 3.6 – continuación de la página anterior

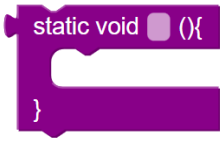
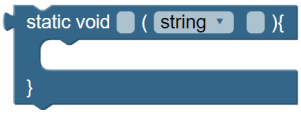
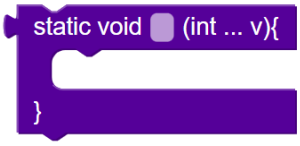
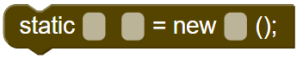
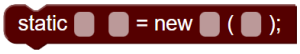
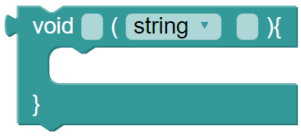
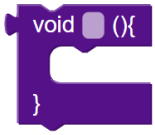
Bloque	Forma visual	Descripción
Método de clase.		Genera el cuerpo vacío de método estático sin valor de retorno.
Método de clase.		Genera el cuerpo vacío de método estático sin valor de retorno y con un parámetro.
Método de clase.		Genera el cuerpo vacío de método estático sin valor de retorno y con argumentos de longitud variable.
Variable de clase tipo referencia.		Agrega línea de variable inicializada con un tipo.
Variable de clase tipo referencia.		Agrega línea de variable inicializada con tipo y un parámetro.
Bloque de inicializador de clase.		Genera el cuerpo de inicializador estático.

Tabla 3.7: Bloques de código a nivel objeto.

Bloque	Forma visual	Descripción
Bloque de método de instancia.		Genera el cuerpo de método de instancia sin valor de retorno y con un parámetro.
Bloque de método de instancia.		Genera el cuerpo del método de instancia sin valor de retorno y con argumentos de longitud variable.
Bloque de método de instancia.		Genera el cuerpo de método de instancia sin parámetros.
Bloque de variable de instancia tipo referencia.		Agrega línea de variable inicializada.
Bloque de variable de instancia tipo referencia.		Agrega línea de variable inicializada con un valor de parámetro.
Bloque de inicializador de instancia.		Genera cuerpo vacío de inicializador.

Como se observa en la tabla 3.7, la diferencia de algunos bloques con los bloques de la categoría a nivel clase radica únicamente en que no incluyen la palabra reservada `static`.

Pasos del PCIO

En la literatura, Bruce Eckel [12] reporta algunos fragmentos del proceso de construcción: las primeras líneas de ejecución serán aquellas que contengan datos y métodos de clase, esto se refiere a todo aquello que incluye la palabra reservada static, con esto se entiende que le pertenece a la clase todo aquello que es estático, porque no es necesario crear un objeto nombrado o anónimo para acceder a miembros y métodos estáticos. En el siguiente ejemplo se crea una clase con una variable estática inicializada:

```
class StaticTest {
    static int i = 47;
}
```

Hay dos formas de invocar una variable estática, la primera es a través de un objeto nombrado, ejemplo `obj.i` o hacer referencia directa a través del nombre de la clase:

```
StaticTest.i++;
```

Aquí se observa la diferencia con un miembro no estático, los miembros no estáticos se consideran pertenecientes al objeto que se crea o miembros de instancia. Ejemplo, en una clase con un método de instancia:

```
class Test {
    void noStatic(){}
}
```

La invocación al método no estático es posible sólo a través de un objeto nombrado o anónimo:

```
Test t1 = new Test();
t1.noStatic();
new Test().noStatic();
```

En el orden de inicialización se ejecutará primero todo aquello que es estático siempre que no haya sido inicializado por la creación de un objeto anterior, y después todo aquello que no es estático.
De forma general Bruce Eckel menciona seis pasos correspondientes al PCIO:

1. La primera vez que se crea un objeto de una clase o la primera vez que se acceda a un método o campo estático de la clase, el intérprete de Java localizará el archivo `nombredeclase.class`, esto se hace a través de la variable de entorno *Classpath*.
2. Se ejecutan los inicializadores estáticos. Esto significa que la inicialización estática tiene lugar una sola vez cuando el objeto de clase se carga por primera vez.
3. Cuando se crea un objeto nuevo, el primer paso del proceso de construcción de este es asignar suficiente espacio de almacenamiento en memoria para alojarse.
4. Los primitivos de la clase se inicializan con valores *default*, esto está configurado automáticamente de modo que se establece valor en cero para números, el equivalente para valores booleanos y char y las referencias en *null*.
5. Se ejecuta cualquier inicialización que ocurra en el punto de definición del campo.
6. Se ejecutan los constructores.

Aunque los pasos anteriores describen correctamente el proceso de construcción, no son suficientes para lograr una comprensión detallada dado que el autor no considera lo siguiente:

- El detalle de la carga de clase.
- Los campos e inicializadores se ejecutan en el orden que aparezcan en el código de forma descendente.
- La forma de invocar al superconstructor cuando se utiliza la herencia entre clases.

Con base en la literatura revisada y en conjunto con pruebas de ejecución de programas especialmente diseñados para observar el comportamiento en cada parte estructural de código (ejemplo: 3.7), se refinó la descripción de los pasos que describen adecuada-

mente y con mayor claridad y comprensión el PCIO. A continuación, se presentan los pasos de dicho proceso de forma detallada:

A nivel clase:

1. Búsqueda de clases mediante *Classpath* (paso explícito para versiones de Java 1.8 y anteriores).
2. Reserva de memoria para la clase.
3. Ejecución de campos estáticos con valores *default*.
4. Ejecución de campos estáticos con valores explícitos.
5. Ejecución de bloques estáticos (inicializadores de clase).

A nivel objeto:

6. Reserva de memoria para el objeto.
7. Ejecución de campos de instancia con valores *default*.
8. Ejecución de campos de instancia con valores explícitos.
9. Ejecución de bloques de instancia (inicializadores).
10. Ejecución de los constructores.
    - Siempre se ejecuta una instrucción de `super()`, ubicada en la primera línea, lo cual significa:
      - Si en la superclase existe un constructor sin argumentos, se ejecuta de forma implícita (Java realiza este paso automáticamente).
      - Si en la superclase existe uno o más constructores con argumentos, se ejecuta de forma explícita con invocación al superconstructor con argumentos establecidos.

Los pasos 3 al 5 (a nivel clase) y 7 al 9 (a nivel objeto) son intercambiables según sea el orden de declaración en código.

3.2.3. Análisis de requerimientos

Los requerimientos proporcionan la información de aquellas características observables que el módulo de software contendrá para satisfacer la necesidad inicial.

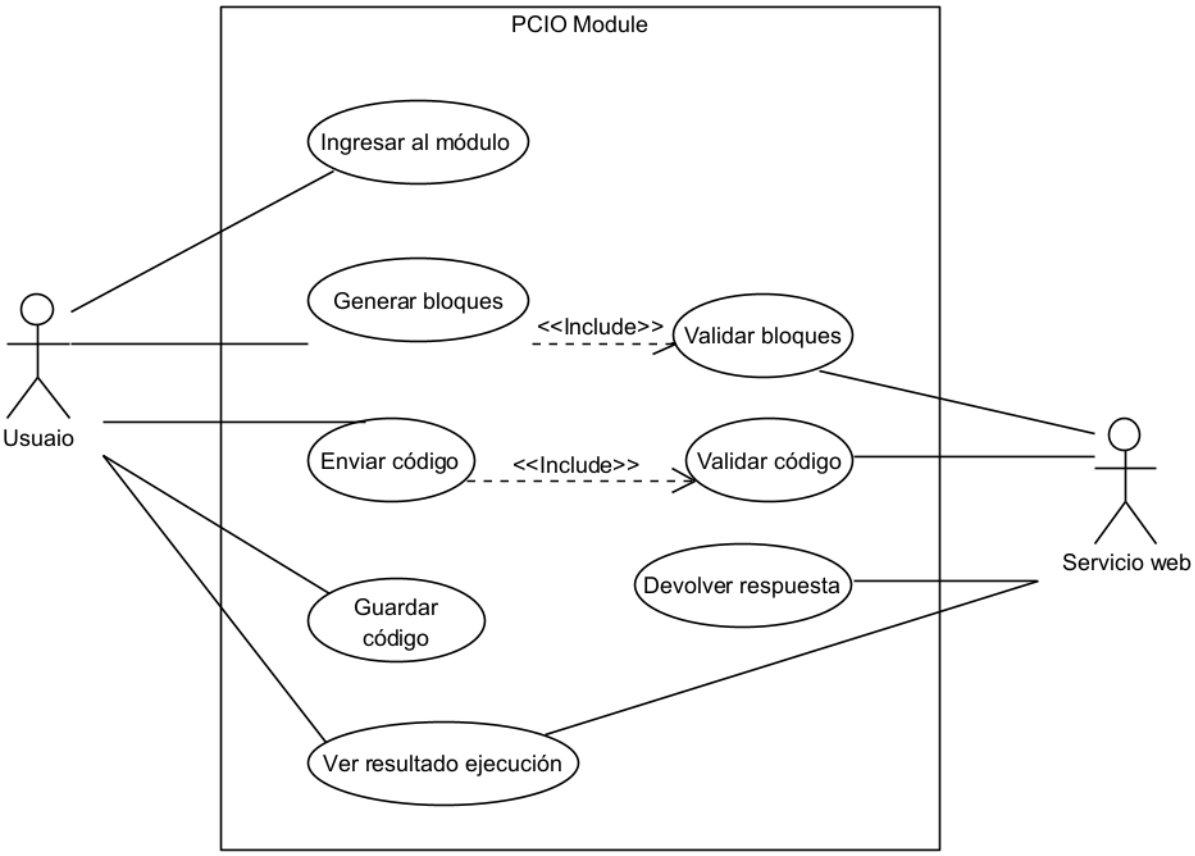


Figura 3.3: Diagrama de Casos de Uso

Los requerimientos corresponden únicamente al actor usuario y servicio web, las tareas se representan en la figura 3.3 a través de casos de uso.

Los actores principales son el usuario de la aplicación y el servicio web, el primero realiza uso directamente con la interfaz gráfica donde genera bloques de código, envío y guardado del mismo y visualización de respuestas; mientras que el segundo actor es el sistema como servicio web, el cual se encarga de validar la estructura del código y ejecutar de acuerdo con el proceso de construcción de objetos.

Clave	Caso de uso	Descripción
CU01	Consultar ejemplos	El usuario consultará ejemplos de código fuente con elementos suficientes para repaso del PCIO.
CU02	Consultar pasos	El usuario consultará los pasos detallados del PCIO.

Tabla 3.9: Descripción del diagrama de caso de uso de la figura 3.4

En la tabla 3.8 se encuentra la descripción del diagrama de casos de uso de la figura 3.3.

Tabla 3.8: Descripción del diagrama de caso de uso de la figura 3.3.

Clave	Caso de uso	Descripción
CU01	Ingresar al módulo	El usuario ingresará al módulo de software a través de cualquier navegador web.
CU02	Ensamblar bloques	El usuario generará el código fuente necesario a través del efecto de arratrar y soltar bloques visuales.
CU03	Enviar código	Una vez generado el código fuente, el usuario enviará a validar la estructura del mismo.
CU04	Guardar código	El usuario guardará el código fuente generado si así lo desea.
CU05	Ver resultado ejecución	El usuario visualizará el resumen de ejecución del código enviado con los pasos del PCIO.
CU06	Validar bloques	El sistema de validaciones verificará la estructura del código fuente.
CU07	Validar código	El sistema de validaciones describirá los pasos ejecutados durante la ejecución y compilación del código fuente.
CU08	Devolver respuesta	El sistema de validaciones devolverá la respuesta de ejecución una vez realizadas todas las validaciones.

La figura 3.9 incluye la interacción con casos de uso adicionales que se describen en la tabla 3.9.

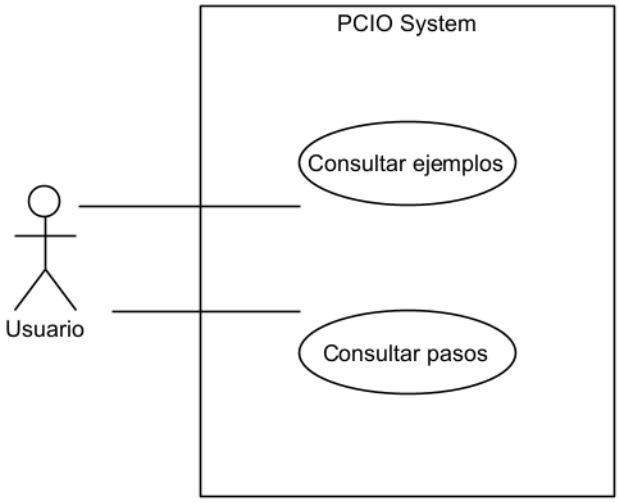


Figura 3.4: Diagrama de Casos de Uso - Adicionales

3.2.4. Modelo de funcionamiento

Para representar la secuencia o proceso de comunicación/interacción entre el usuario y el módulo de aprendizaje, se diseñaron los siguientes diagramas de actividades detallados a continuación.

En la figura 3.5 se presenta el diagrama de actividades del proceso de envío y validación de código fuente, cabe destacar que este proceso es el primordial en este proyecto.

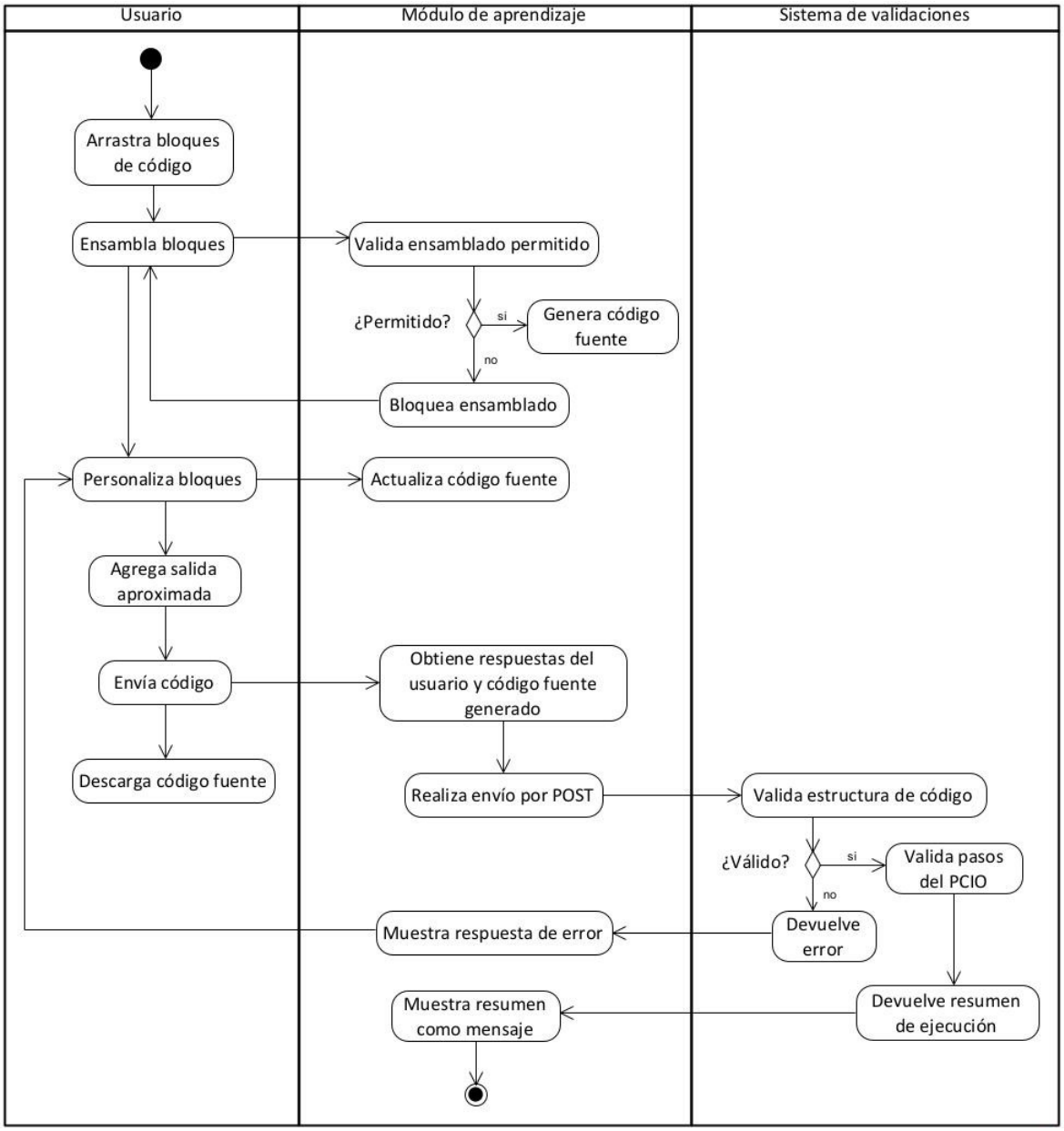


Figura 3.5: Diagrama de actividades - proceso de envío y validación.

Las interacciones adicionales que el usuario realizará durante su navegación dentro del módulo, tales como: acceder al listado de pasos del PCIO y revisar ejemplos de código para replicarlos en la construcción con bloques, se presentan en el diagrama en la figura 3.6.

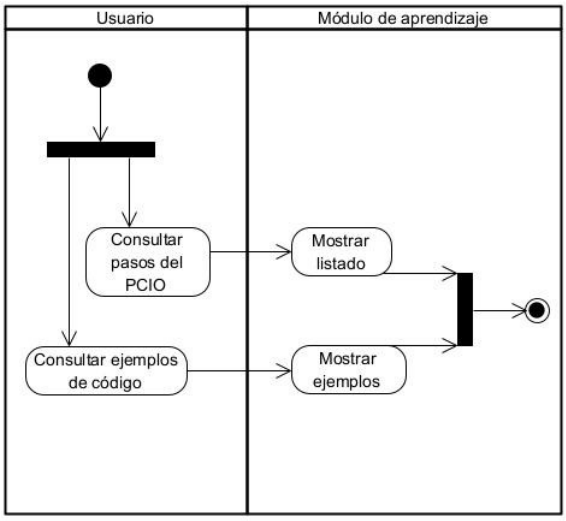


Figura 3.6: Diagrama de actividades - opciones adicionales del módulo.

3.2.5. Diseño del módulo

En esta sección se presentan los *Wireframes* o representaciones visuales para las interfaces gráficas del módulo de aprendizaje.

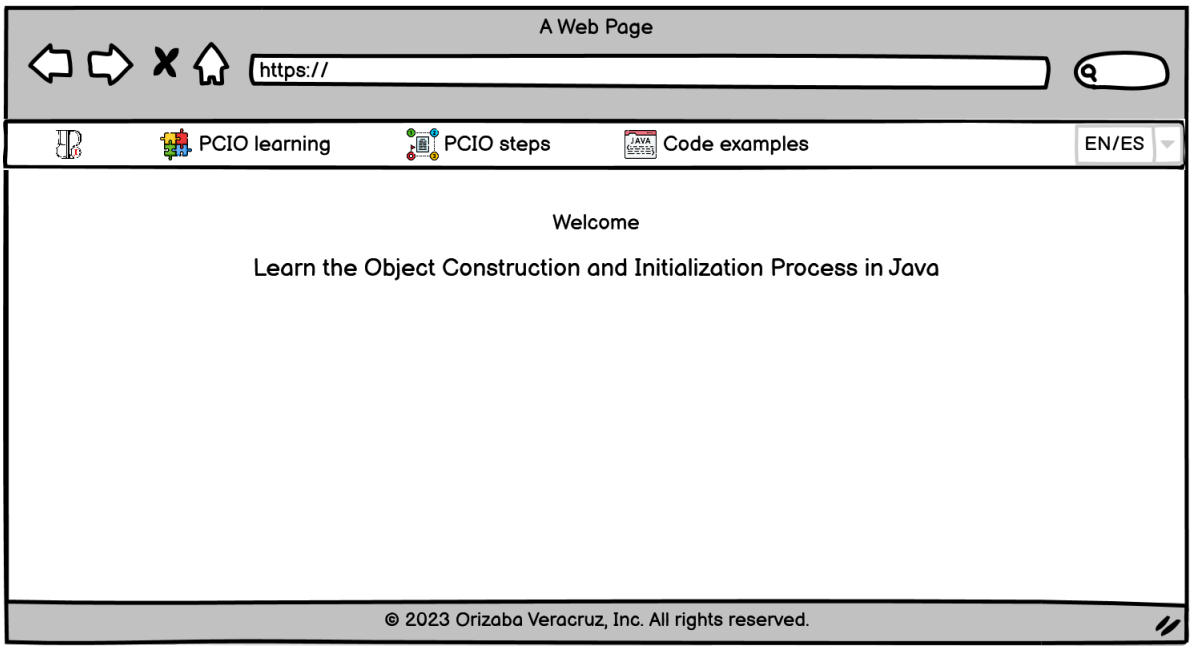


Figura 3.7: *Wireframe* ventana de inicio del módulo.

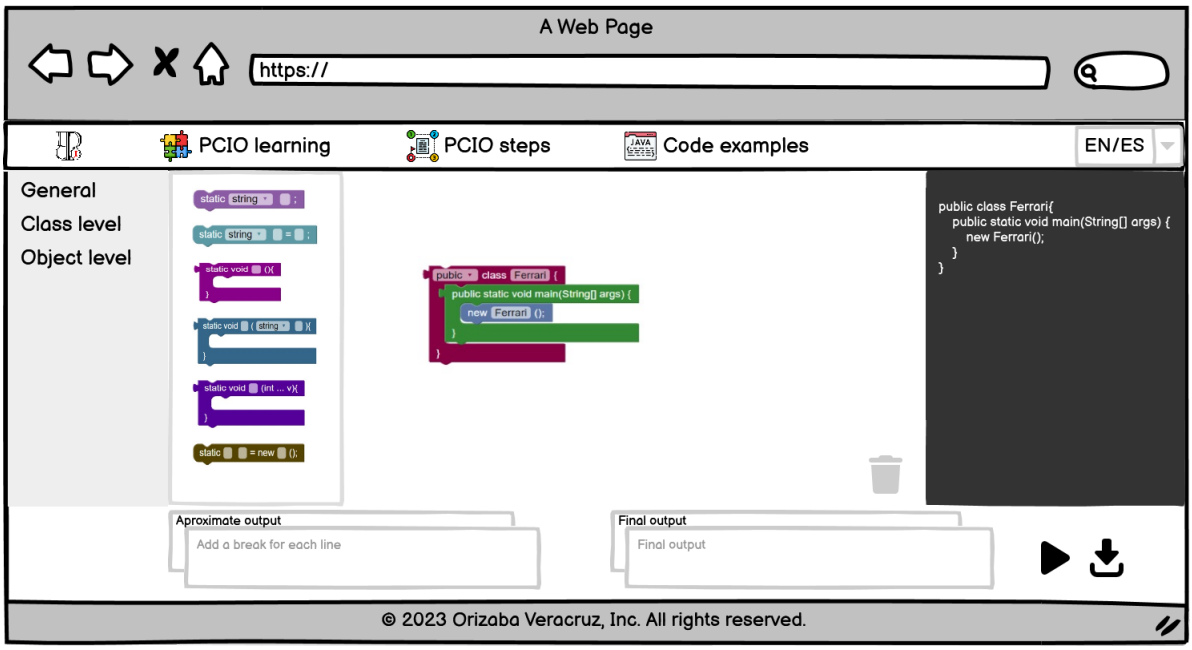


Figura 3.8: *Wireframe* ventana de aprendizaje del PCIO.

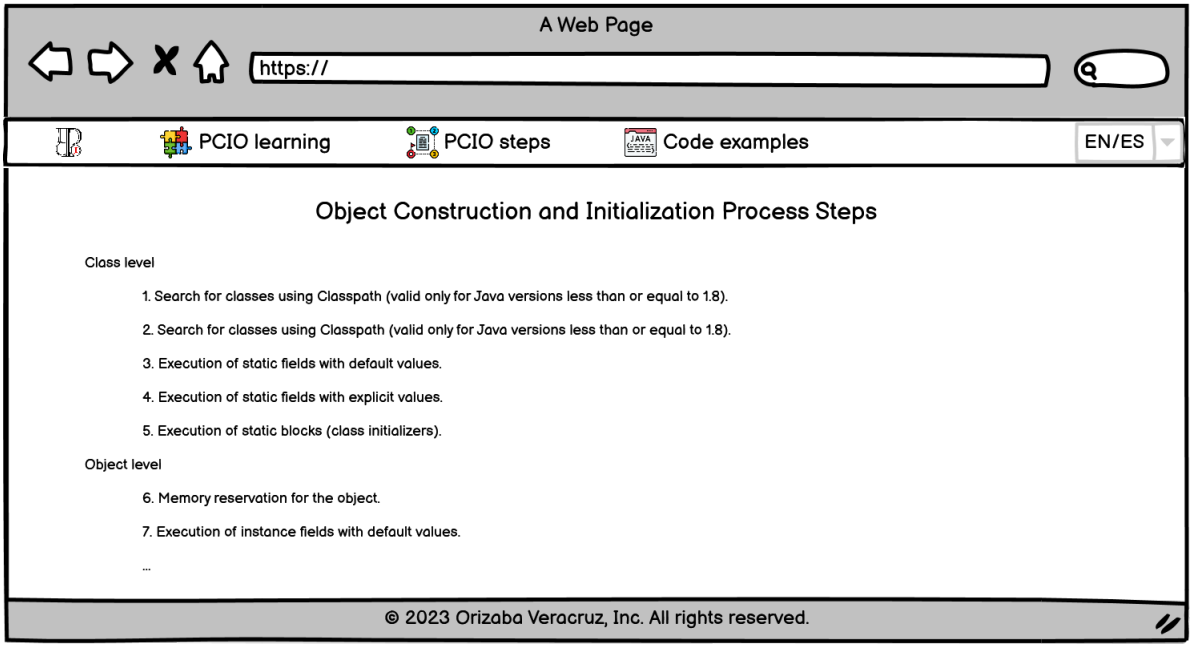


Figura 3.9: *Wireframe* ventana de pasos del PCIO.

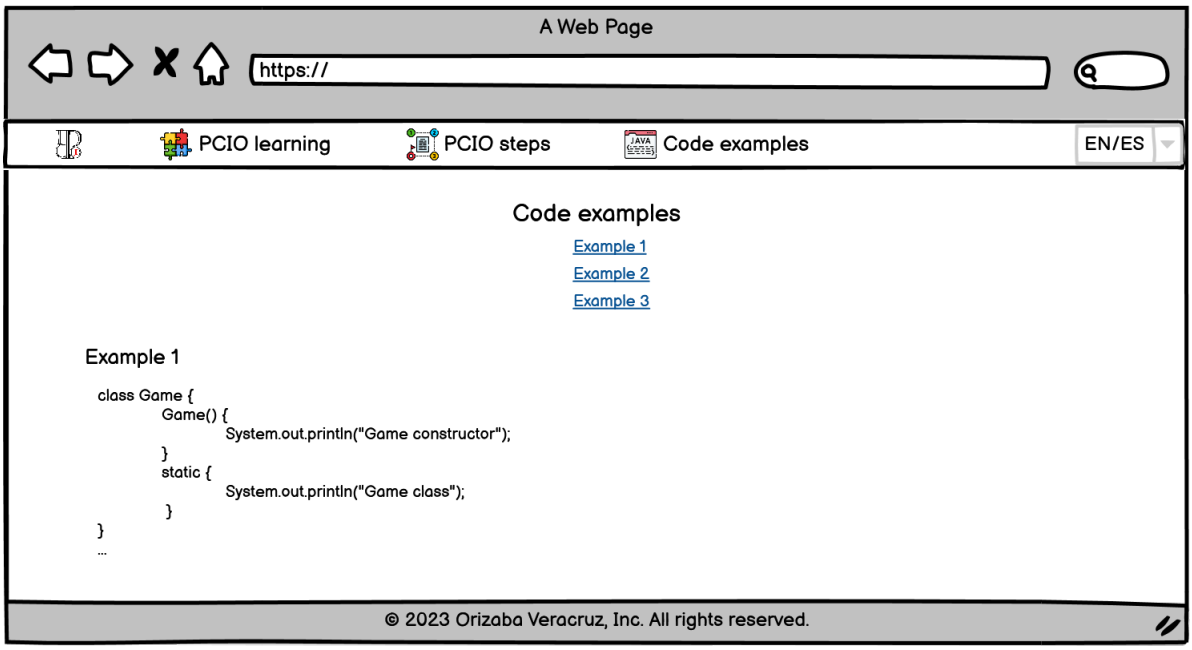


Figura 3.10: *Wireframe* ventana de ejemplos de código.

La figura 3.7 incluye el inicio donde se añade un mensaje de bienvenida al módulo, esta incluye el uso de menú para acceder a las demás ventanas.

En la ventana de la figura 3.8 se encuentran los bloques de código dentro del panel lateral izquierdo divididos por categorías, un espacio de trabajo en el centro de la ventana donde se realiza el ensamblado de bloques, un panel lateral derecho donde se genera automáticamente el código fuente; en la parte inferior se encuentra una caja de texto para agregar la salida aproximada y otra para visualizar mensajes de ejecución, así como la salida exacta del código; finalmente se localiza un botón de ejecución o envío de código y el botón de descarga de código.

Como sección informativa se añade una ventana donde se listan los pasos detallados del PCIO, tal como se muestra en la figura 3.9.

Se añaden algunos ejemplos de código para ser replicados por el usuario en la ventana de la figura 3.10.

3.2.6. Mapa de navegación

A continuación, se presenta el mapa navegacional del módulo de aprendizaje en la figura 3.11, el usuario navegará entre las 4 ventanas ya descritas en los *mockups* anteriores. Este mapa permite al usuario acceder a cada sección de contenido de manera intuitiva y ordenada.

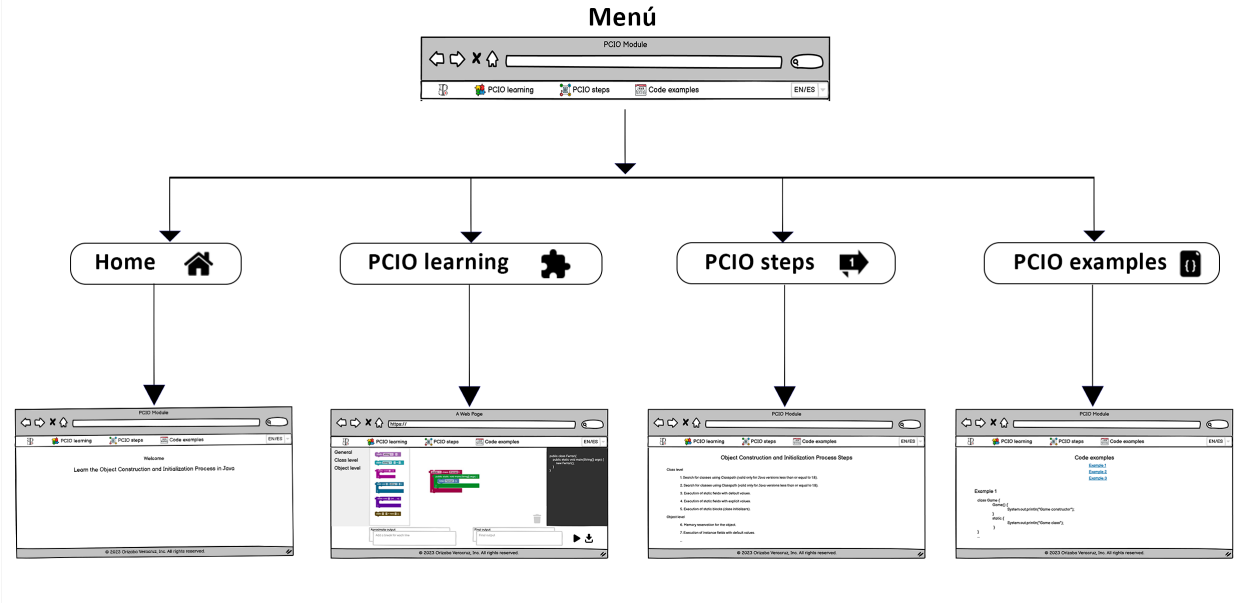


Figura 3.11: Mapa de navegación del módulo.

3.2.7. Modelo de contenido

La composición de todo el módulo de *software*, incluyendo el servicio web se representa en el diagrama de paquetes de la figura 3.12, la comunicación entre cliente y el servicio web se realiza a través de `utils` y el controlador del servicio.

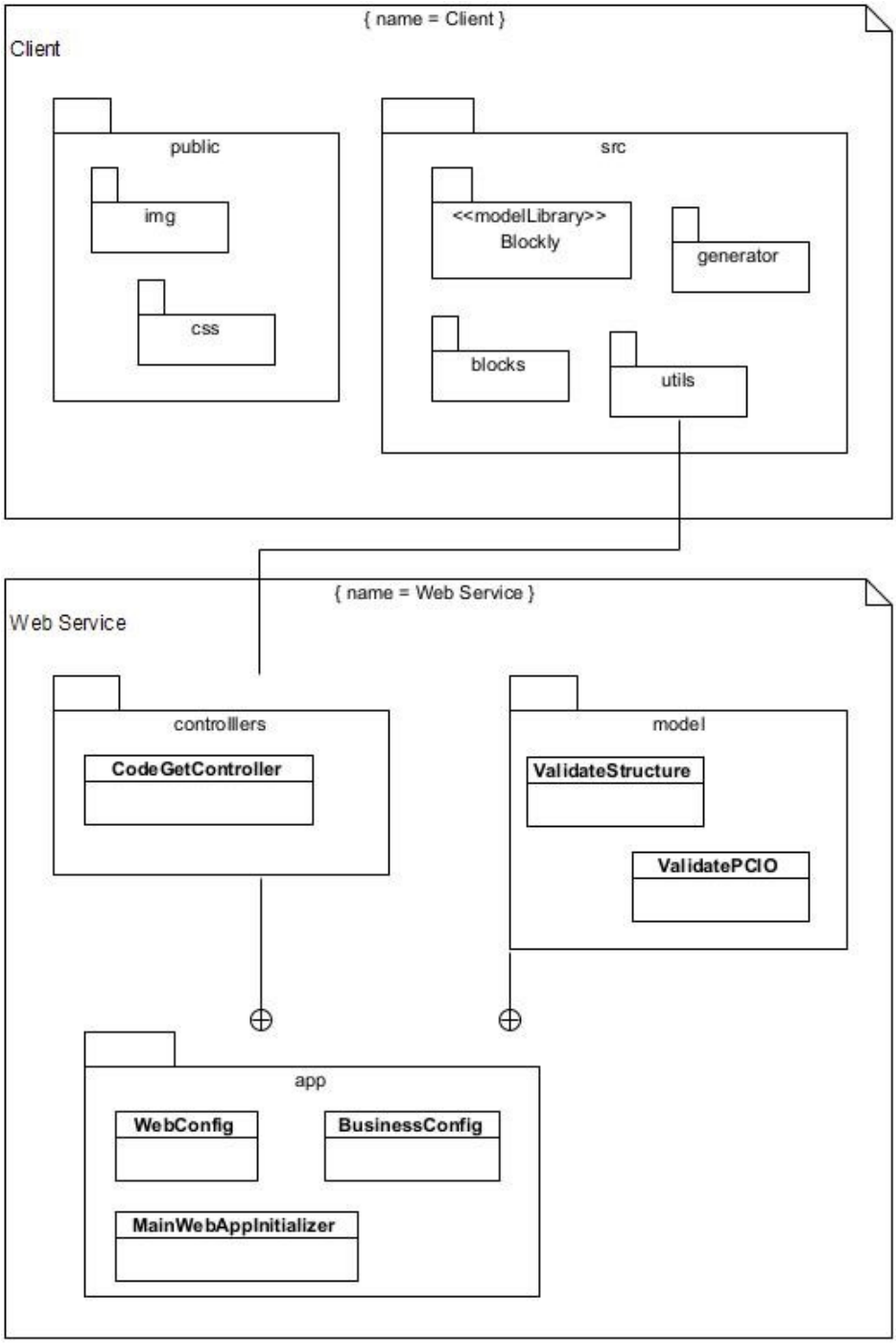


Figura 3.12: Diagrama de paquetes.

3.3. Implementación

En este apartado se describe el desarrollo de las interfaces del módulo de aprendizaje, así como el funcionamiento real de cada componente del servicio web.

3.3.1. Implementación de Blockly

La implementación de Blockly permitió la personalización de bloques de código para el lenguaje de programación Java, como se describió en la sección de componentes de módulo, el editor por bloques incluye solo aquellos necesarios para el aprendizaje del proceso de construcción, específicamente.

Debido a que Blockly no incluye Java como uno de los lenguajes de programación precargados, todos los bloques se generaron manualmente a través de componentes.

El siguiente fragmento de código se utilizó para crear un nuevo generador de bloques, específicamente para código Java. El código 3.1 está escrito en JavaScript en conjunto con React.

Código 3.1: Generador de bloques.

```
import * as Blockly from 'blockly/core';

const javagenerator = new Blockly.Generator('JAVA');

javagenerator.scrub_ = (block, code, opt_thisOnly) => {
    const nextBlock = block.nextConnection &&
        block.nextConnection.targetBlock();
    return (nextBlock && !opt_thisOnly) ?
        code + '\n' + javagenerator.blockToCode(nextBlock) : code;
};

export default javagenerator;
```

Por medio de los componentes que Blockly incluye, se definieron los bloques que serán requeridos dentro de los bloques diseñados para Java, tales como bloque, categoría, valor, campo y sombra, el código 3.2 define esta implementación.

Una vez listo el ambiente de desarrollo Blockly, se generó cada bloque de los descritos en las tablas 3.5, 3.6 y 3.7. A continuación, se muestran tres ejemplos de desarrollo de bloques personalizados, uno por cada categoría.

Código 3.2: Bloques internos.

```
import React from 'react';
import BlocklyComponent from './BlocklyComponent';

export default BlocklyComponent;

const Block = (p) => {
    const { children, ...props } = p;
    props.is = "blockly";
    return React.createElement("block", props, children);
};

const Category = (p) => {
    const { children, ...props } = p;
    props.is = "blockly";
    return React.createElement("category", props, children);
};

const Value = (p) => {
    const { children, ...props } = p;
    props.is = "blockly";
    return React.createElement("value", props, children);
};

const Field = (p) => {
    const { children, ...props } = p;
    props.is = "blockly";
    return React.createElement("field", props, children);
};

const Shadow = (p) => {
    const { children, ...props } = p;
    props.is = "blockly";
    return React.createElement("shadow", props, children);
};

export { Block, Category, Value, Field, Shadow }
```

La categoría general incluye 12 bloques, uno de los relevantes es el bloque de clase con uso de herencia, mismo que se presenta en el código 3.3.

Código 3.3: Bloque de clase con herencia.

```
import * as Blockly from 'blockly/core';
import javagenerator from '../../generator/generator.java';

const blockName = 'class_inheritance'
const classInheritanceBlock = {
    'type': blockName,
    'message0': '%1 class %2 extends %3{',
    'args0': [
       {'type': 'field_dropdown', 'name': 'BLOCK_MODIFIER', 'options': [
           ['default', ''],
           ['pubic', 'public'],
           ['private', 'private'],
           ['protected', 'protected'],
       ]},
       {'type': 'field_input', 'name': 'BLOCK_KEY'},
       {'type': 'field_input', 'name': 'BLOCK_EXTENDS'},
    ],
    'message1': '%1 }',
    'args1': [
       {'type': 'input_statement', 'name': 'CLASS_MEMBERS'}
    ],
    'colour': 903,
    'output': null,
    'previousStatement': null,
    'nextStatement': null,
}
Blockly.Blocks[blockName] = {
    init: function () {
      this.jsonInit(classInheritanceBlock);
    }
}
javagenerator[blockName] = (block) => {
    const modifier = block.getFieldValue('BLOCK_MODIFIER');
    const name = block.getFieldValue('BLOCK_KEY');
    const extendsfrom = block.getFieldValue('BLOCK_EXTENDS');
    const body = javagenerator.statementToCode(block, 'CLASS_MEMBERS');
    const code = `${modifier} class ${name} extends
       ${extendsfrom}{\n${body}\n}\n`;

    return code;
}
export default classInheritanceBlock;
```

Para la categoría a nivel clase se incluyeron 12 bloques, con el fin de permitir n cantidad de parámetros en el llamado al constructor, se diseñó el método con argumentos de longitud variable en el código 3.4.

Código 3.4: Bloque de constructor con argumentos de longitud variable.

```
import * as Blockly from 'blockly/core';
import javagenerator from '../../generator/generator.java';

const blockName = 'constructor_vl_args'

const ConstructorArgsBlock = {
    'type': blockName,
    'message0': '%1(int ... v){',
    'args0': [
       {'type': 'field_input', 'name': 'BLOCK_KEY', 'value':'Constructor'},
    ],
    'message1': '%1 }',
    'args1': [
       {'type': 'input_statement', 'name': 'FUNCTION_MEMBERS'}
    ],
    'colour': 299,
    'output': null,
    'previousStatement': null,
    'nextStatement': null,
}

Blockly.Blocks[blockName] = {
    init: function () {
      this.jsonInit(ConstructorArgsBlock);
    }
}

javagenerator[blockName] = (block) => {
    const name = block.getFieldValue('BLOCK_KEY');
    const statementMembers = javagenerator.statementToCode(block,
       'FUNCTION_MEMBERS');
    const code = `${name}(int ... v){\n${statementMembers}\n}`;

    return code;
}

export default ConstructorArgsBlock;
```

En el caso de la categoría a nivel objeto, incluye 6 bloques, un ejemplo es el inicializador de instancia, creado con el código 3.5.

Código 3.5: Bloque de inicializador de instancia.

```
import * as Blockly from 'blockly/core';
import javagenerator from '../../generator/generator.java';

const blockName = 'instance_initializer'

const instanceInitializerBlock = {
    'type': blockName,
    'message0': '{ %1 %2 }',
    'args0': [
       {'type': 'input_dummy'},
       {'type': 'input_statement', 'name': 'MEMBERS'},
    ],
    'colour': 860,
    'output': null,
    'previousStatement': null,
    'nextStatement': null,
}

Blockly.Blocks[blockName] = {
    init: function () {
      this.jsonInit(instanceInitializerBlock);
    }
}

javagenerator[blockName] = (block) => {
    const statementMembers = javagenerator.statementToCode(block, 'MEMBERS');
    const code = '{\n' + statementMembers + '\n}';

    return code;
}

export default instanceInitializerBlock;
```

La personzalización de bloques fue posible gracias a la amplia capacidad de Blockly para crear cualquier componente de código, independientemente el lenguaje de programación que se utilice.

3.3.2. Implementación de Axios

Axios es el cliente HTTP para realizar la petición al servicio web a través de promesas. Una promesa o *promise* es un objeto que representa el éxito o fracaso de una operación asíncrona [32].

Código 3.6: Implementación de Axios.

```
axios.get(URL_WS_PCIO)
.then(response => response.data).then((data) =>{
    output = data[0];
    acertado = data[1];
    msg = data[2];

    createNotification('info',acertado,msg)
    document.getElementById("textResponse").value = output;
});
```

3.3.3. Interfaces finales

Para el desarrollo de la aplicación cliente se utilizó el entorno de desarrollo web React, la biblioteca de JavaScript para crear interfaces de usuario. En esta sección se presentan las vistas finales de la aplicación.

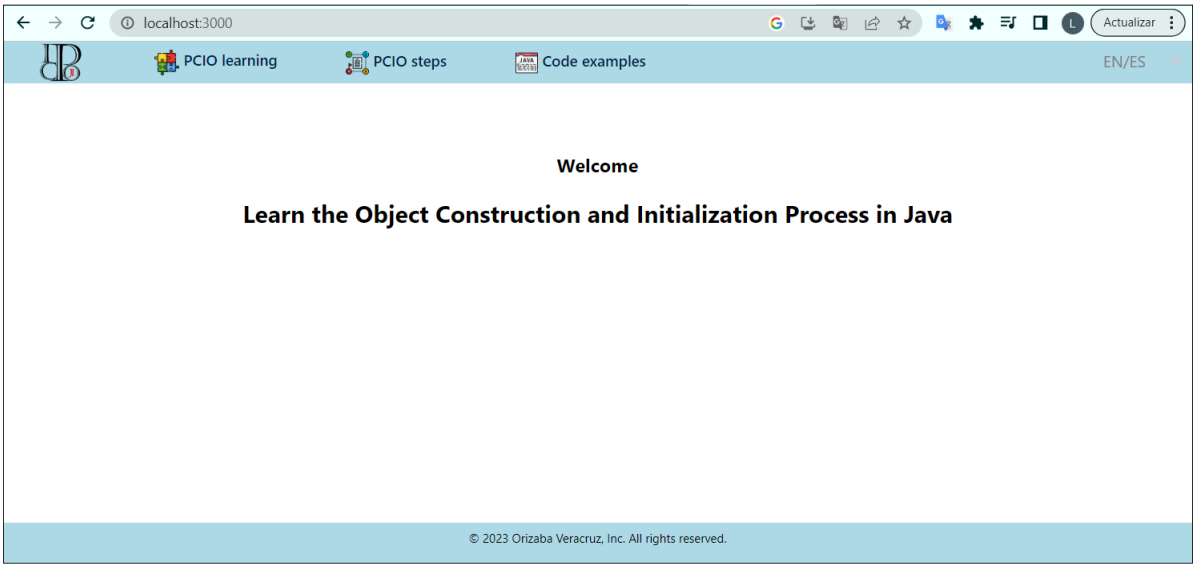


Figura 3.13: Ventana de inicio.

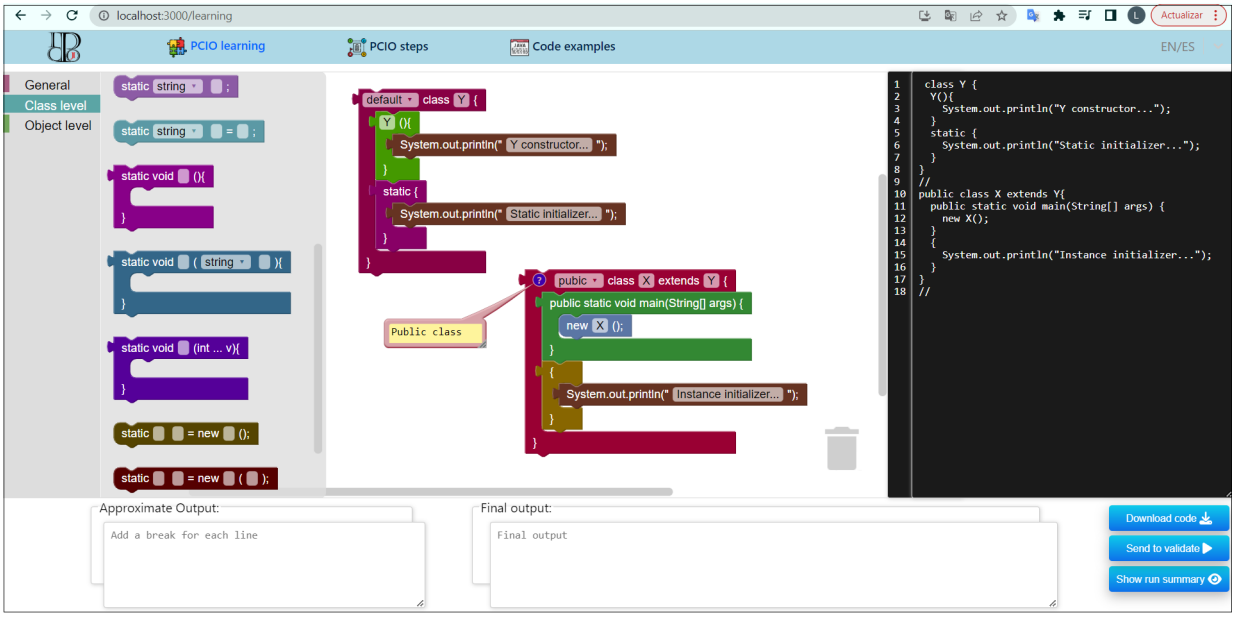


Figura 3.14: Ventana de Aprendizaje del PCIO.

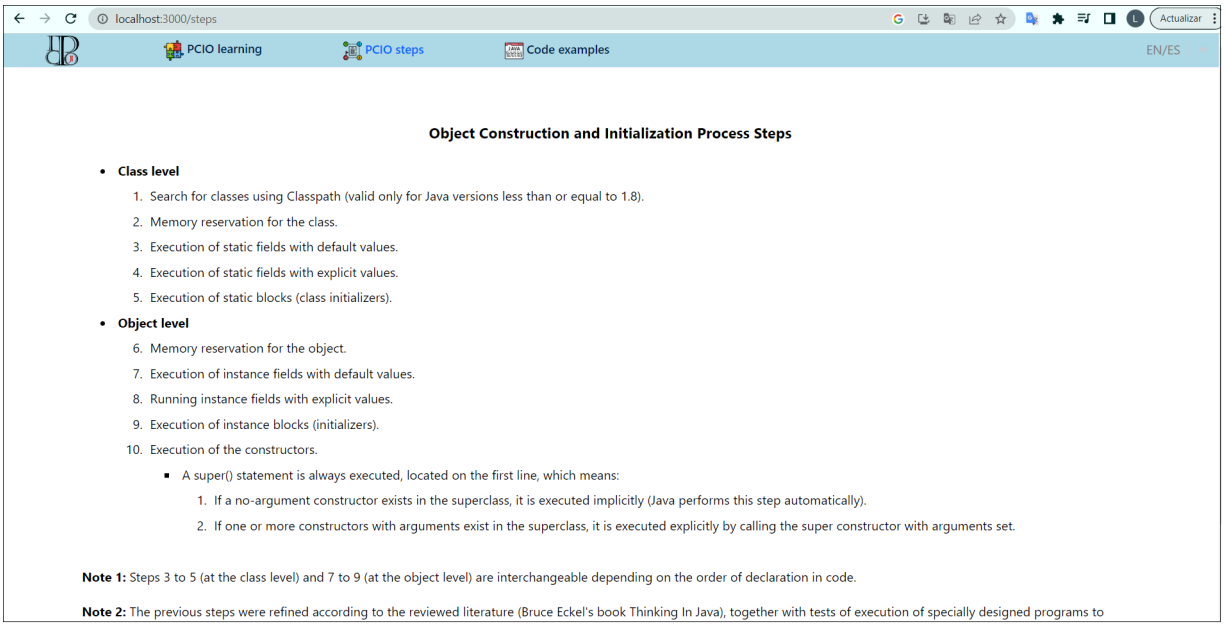


Figura 3.15: Ventana de Pasos del PCIO.

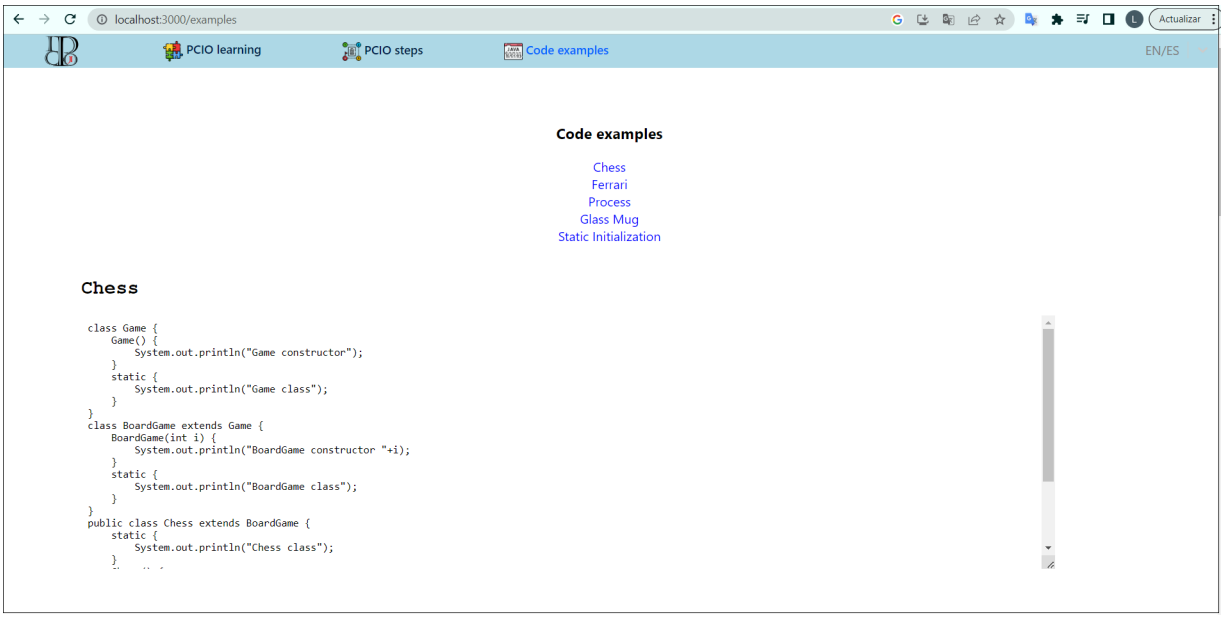


Figura 3.16: Ventana de Códigos Ejemplo.

La ventana de la figura 3.13 muestra el inicio del módulo, la ventana en la figura 3.14 implementa Blockly para la generación de código fuente, la figura 3.15 indica el orden de los pasos del PCIO solo como ventana o sección informativa y la figura 3.16 es una sección de ejemplos de código Java para utilización del usuario.

Los idiomas disponibles para mostrar el contenido de todas las ventanas son inglés y español, el usuario selecciona el lenguaje de su preferencia desde el selector ubicado en la parte posterior derecha (figura 3.17).

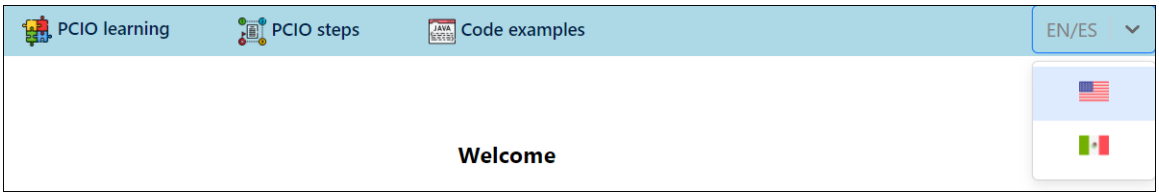


Figura 3.17: Selección de idioma del módulo.

3.4. Pruebas

En la fase de pruebas se utilizaron códigos Java para verificar el correcto funcionamiento de Blockly. A continuación, el código 3.7 muestra un ejemplo que contiene todos los elementos claves para comprender el PCIO, estos elementos se replicaron con bloques de código visuales.

Después del ensamblado de bloques visuales que se presenta en la figura 3.18, se obtuvo un código fuente idéntico al código 3.7, con esto se comprueba el funcionamiento para programar en tiempo de diseño a través de la herramienta y las funcionalidades que Blockly ofrece.

Código 3.7: Código StaticInitialization para pruebas.

Fuente: Eckel. B. (2006). *Constructor initialization*. Thinking in Java.

```
class Bowl {
    Bowl(int marker) {
        System.out.println("Bowl(" + marker + ")");
    }
    void f1(int marker) {
        System.out.println("f1(" + marker + ")");
    }
    { System.out.println("Bowl.<init>"); }
}

class Table {
    static Bowl bowl1 = new Bowl(1);
    Table() {
        System.out.println("Table()");
        bowl2.f1(1);
    }
    void f2(int marker) {
        System.out.println("f2(" + marker + ")");
    }
    static Bowl bowl2 = new Bowl(2);
}

class Cupboard {
    Bowl bowl3 = new Bowl(3);
    static Bowl bowl4 = new Bowl(4);
    Cupboard() {
        System.out.println("Cupboard()");
        bowl4.f1(2);
    }
    void f3(int marker) {
        System.out.println("f3(" + marker + ")");
    }
    static Bowl bowl5 = new Bowl(5);
}

class Initial {
    { System.out.println("Super"); }
    static { System.out.println("Config now"); }
    Initial() {
        System.out.println("Constructor Initial");
    }
}
public class StaticInitialization extends Initial {
    StaticInitialization() {
        System.out.println("Constructor derived class StaticInitialization");
    }
    public static void main(String[] args) {
        System.out.println("Creating new Cupboard() in main");
        new Cupboard();
        System.out.println("Creating new Cupboard() in main");
        new Cupboard();
        table.f2(1);
        cupboard.f3(1);
        new StaticInitialization();
    }
    static Table table = new Table();
    static Cupboard cupboard = new Cupboard();
}
```

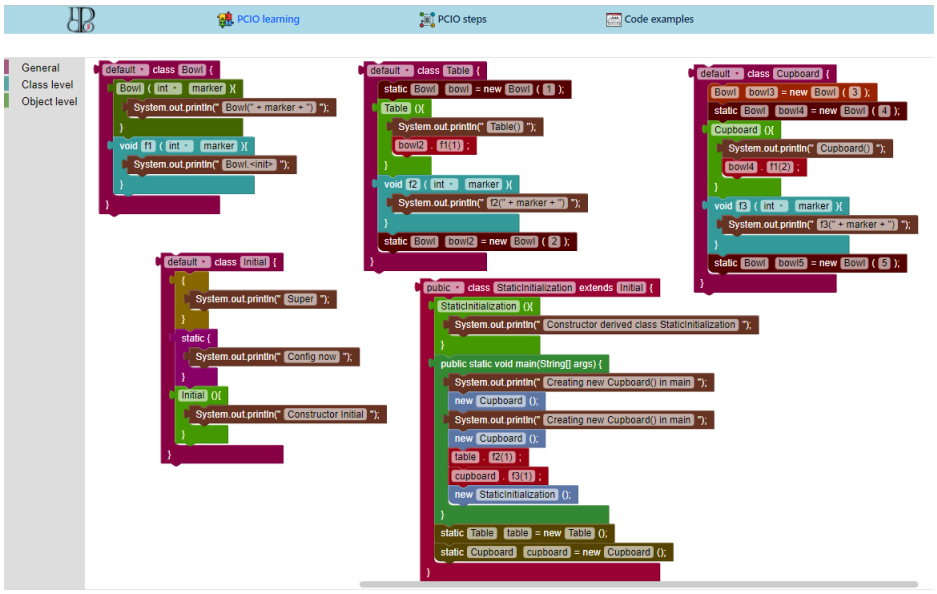
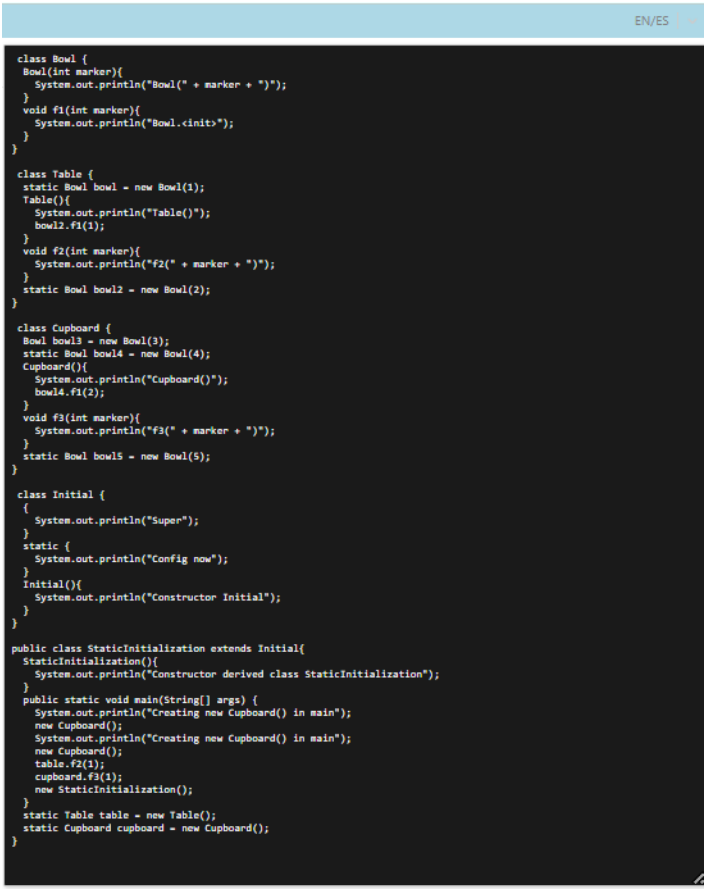


Figura 3.18: Ensamblado de bloques.



```
class Bowl {
  Bowl(int marker){
    System.out.println("Bowl(" + marker + ")");
  }
  void f1(int marker){
    System.out.println("Bowl.<init>");
  }
}

class Table {
  static Bowl bowl = new Bowl(1);
  Table(){
    System.out.println("Table()");
    bowl2.f1(1);
  }
  void f2(int marker){
    System.out.println("f2(" + marker + ")");
  }
  static Bowl bowl2 = new Bowl(2);
}

class Cupboard {
  Bowl bowl3 = new Bowl(3);
  static Bowl bowl4 = new Bowl(4);
  Cupboard(){
    System.out.println("Cupboard()");
    bowl4.f1(2);
  }
  void f3(int marker){
    System.out.println("f3(" + marker + ")");
  }
  static Bowl bowl5 = new Bowl(5);
}

class Initial {
  {
    System.out.println("Super");
  }
  static {
    System.out.println("Config now");
  }
  Initial(){
    System.out.println("Constructor Initial");
  }
}

public class StaticInitialization extends Initial{
  StaticInitialization(){
    System.out.println("Constructor derived class StaticInitialization");
  }
  public static void main(String[] args) {
    System.out.println("Creating new Cupboard() in main");
    new Cupboard();
    System.out.println("Creating new Cupboard() in main");
    new Cupboard();
    table.f2(1);
    cupboard.f3(1);
    new StaticInitialization();
  }
  static Table table = new Table();
  static Cupboard cupboard = new Cupboard();
}
```

Figura 3.19: Código resultante.

Como se observa en la figura 3.19, se agregan tantos bloques de código como sea necesario ya que no existe límite o longitud de caracteres, el código a generar será tan extenso como se desee. Así, se valida el funcionamiento de Blockly y el código resultante se encuentra listo para ser enviado y procesado en el sistema de validaciones.

Figura 3.20: Respuesta del servicio web.

Una vez procesado el código, la respuesta del servicio web se muestra en la figura 3.20, en primer se encuentra el resumen de ejecución de código, aquí se indican los pasos del PCIO que aplicaron o se utilizaron en el ejemplo de la figura 3.19, en segundo lugar se encuentra la salida correcta del código y en tercer lugar un mensaje de "¡Felicidades!" en caso de haber acertado la respuesta y "¡Sigue intentándolo!" en el caso contrario; de esta forma se pretende que el usuario comprenda y analice esta serie de pasos con diferentes códigos de programación.

La comunicación entre cliente-servidor se realiza correctamente, asimismo cada componente de la arquitectura realiza las tareas que le fueron asignadas y con esto se obtiene un prototipo funcional del módulo.

Capítulo 4

Resultados

Este capítulo tiene como objetivo presentar los resultados de este proyecto de tesis y el caso de estudio, de acuerdo con la problemática abordada y objetivo propuesto para el módulo de *software* de apoyo para el aprendizaje del Proceso de Construción e Inicialización de Objetos en Java. Este sistema pretente brindar la retroalimentación necesaria para comprender correctamente el PCIO a través de un resumen de ejecución. A continuación, se describe el caso de estudio para este proyecto de Tesis.

4.1. Caso de estudio

Dado que este proyecto está inclinado principalmente hacia el área educativa, se eligió llevar a cabo pruebas con estudiantes de nivel licenciatura de la Universidad Tecnológica del Centro de Veracruz ubicada en Avenida Universidad 350, 94910 Cuitláhuac, Ver.

Con la participación del grupo "A" de noveno cuatrimestre de Ingeniería en Desarrollo y Gestión de Software, se obtuvieron los resultados que se describen en la secciones 4.1.1, 4.1.2 y 4.2.1.

4.1.1. Conocimiento previo sobre el PCIO

En un grupo de 22 estudiantes, listados en la tabla 4.1 se encontró que ninguno de ellos conocía el tema de interés, aunque en algún punto de su trayecto escolar se les impartieron materias que incluían temas como Programación Orientada a Objetos en Java, nunca abordaron el PCIO de forma específica. Antes de comenzar con el proceso de aprendizaje de dicho tema, se aplicó la siguiente encuesta como una introducción donde se observó que todos los estudiantes desconocían completamente el tema a pesar de haber utilizado el lenguaje Java en clases de programación.

Tabla 4.1: Resultados encuesta inicial del PCIO.

Estudiante	**Conoce el PCIO**	**Utiliza Java**	**Domina objetos en Java**	**Construir objetos es fácil**
1	No	Sí	Sí	Sí
2	No	Sí	Sí	Sí
3	No	Sí	Sí	No
4	No	Sí	Sí	Sí
5	No	Sí	Sí	Sí
6	No	Sí	Sí	Sí
7	No	Sí	Sí	No
8	No	Sí	Sí	Sí
9	No	Sí	Sí	Sí
10	No	Sí	Sí	Sí
11	No	Sí	Sí	Sí
12	No	Sí	Sí	Sí
13	No	Sí	Sí	Sí
14	No	Sí	Sí	Sí
15	No	Sí	Sí	Sí
16	No	Sí	Sí	No
17	No	Sí	Sí	No
18	No	Sí	Sí	Sí
19	No	Sí	No	Sí
20	No	Sí	Sí	Sí
21	No	Sí	Sí	Sí
22	No	Sí	Sí	Sí

De forma general todos los estudiantes habían utilizado Java, la mayoría decía dominar objetos y más de la mitad del grupo consideraban que construir objetos es fácil.

4.1.2. Prueba 1 del PCIO

El anexo 1 de este documento muestra la primera prueba que se aplicó a los estudiantes para evaluar el conocimiento previo al aprendizaje del PCIO, como se muestra en las figuras 4.1, 4.2, 4.3, 4.4 y 4.5, se incluyeron dos códigos cortos para obtener el conocimiento de los alumnos, sin embargo, la salida del código que los estudiantes identificaron fue incorrecta, al igual que las respuestas de los otros 17 estudiantes, de forma general, nadie de los 22 estudiantes pasó la primera prueba correspondiente al PCIO.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 1 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 15 de Marzo de 2023

Carrera: Desarrollo y Gestión de Software **Cuatrimestre:** 9° **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Game {
    Game() {
        System.out.println("Game constructor");
    }
}

class BoardGame extends Game {
    BoardGame(int i) {
        System.out.println("BoardGame constructor "+i);
    }
    static {
        System.out.println("BoardGame class");
    }
}

public class Chess extends BoardGame {
    {
        System.out.println("Chess object");
    }
    Chess() {
        super(5);
        System.out.println("Chess constructor");
    }
    public static void main(String[] args) {
        new Chess();
    }
}
```

Salida:

chess object
BoardGame constructor 5
Chess constructor

Código 2:

```
class BeautifulCar {
    void SpeedUp(){
        System.out.println("SpeedUp()");
    }
    static {
        System.out.println("A Beautiful Car {}");
    }
    BeautifulCar(){
        System.out.println("BeautifulCar()");
    }
}

public class Ferrari extends BeautifulCar {
    { System.out.println("Ferrari {}"); }
    public static void main(String[] args) {
        BeautifulCar f = new Ferrari();
        f.SpeedUp();
        new Ferrari();
    }
}
```

Salida:

SpeedUp()
Ferrari {}

Nota: no hay errores de sintaxis, ambos códigos arrojan una salida de acuerdo con las impresiones en consola.

Figura 4.1: Ejemplo 1 Prueba 1 del PCIO.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

PRUEBA 1 DEL PROCESO DE CONSTRUCCIÓN E INICIALIZACIÓN DE OBJETOS

Nombre: [redacted] **Fecha:** 15 de Marzo

Carrera: Desarrollo y Gestión de Software **Cuatrimestre:** 9 **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Game {
    Game() {
        System.out.println("Game constructor");
    }
}

class BoardGame extends Game {
    BoardGame(int i) {
        System.out.println("BoardGame constructor "+i);
    }
    static {
        System.out.println("BoardGame class");
    }
}

public class Chess extends BoardGame {
    {
        System.out.println("Chess object");
    }
    Chess() {
        super(5);
        System.out.println("Chess constructor");
    }
    public static void main(String[] args) {
        new Chess();
    }
}
```

Salida:

El siguiente código va a arrojar las siguientes Salidas

0
1 Game constructor"
2 Board Game constructor
3 Board Game class
4 Chess object.
5 Chess constructor

El código se va a detener Cuando llegue a 5

Código 2:

```
class BeautifulCar {
    void SpeedUp(){
        System.out.println("SpeedUp()");
    }
    static {
        System.out.println("A Beautiful Car {}");
    }
    BeautifulCar(){
        System.out.println("BeautifulCar()");
    }
}

public class Ferrari extends BeautifulCar {
    { System.out.println("Ferrari {}"); }
    public static void main(String[] args) {
        BeautifulCar f = new Ferrari();
        f.SpeedUp();
        new Ferrari();
    }
}
```

Salida:

Nota: no hay errores de sintaxis, ambos códigos arrojan una salida de acuerdo con las impresiones en consola.

Figura 4.2: Ejemplo 2 Prueba 1 del PCIO.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

PRUEBA 1 DEL PROCESO DE CONSTRUCCIÓN E INICIALIZACIÓN DE OBJETOS

Nombre: [redacted] **Fecha:** 15 de marzo del 2023

Carrera: Ingeniería en Desarrollo y Gestión de Software (IDGS) **Cuatrimestre:** 9no **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Game {
    Game() {
        System.out.println("Game constructor");
    }
}

class BoardGame extends Game {
    BoardGame(int i) {
        System.out.println("BoardGame constructor "+i);
    }
    static {
        System.out.println("BoardGame class");
    }
}

public class Chess extends BoardGame {
    {
        System.out.println("Chess object");
    }
    Chess() {
        super(5);
        System.out.println("Chess constructor");
    }
    public static void main(String[] args) {
        new Chess();
    }
}
```

Salida:

"Game constructor"
"BoardGame constructor 5"
"Chess object"
"Chess constructor"

Código 2:

```
class BeautifulCar {
    void SpeedUp(){
        System.out.println("SpeedUp()");
    }
    static {
        System.out.println("A Beautiful Car {}");
    }
    BeautifulCar(){
        System.out.println("BeautifulCar()");
    }
}

public class Ferrari extends BeautifulCar {
    { System.out.println("Ferrari {}"); }
    public static void main(String[] args) {
        BeautifulCar f = new Ferrari();
        f.SpeedUp();
        new Ferrari();
    }
}
```

Salida:

"SpeedUp"
"A Beautiful Car"
"BeautifulCar Ferrari"

Nota: no hay errores de sintaxis, ambos códigos arrojan una salida de acuerdo con las impresiones en consola.

Figura 4.3: Ejemplo 3 Prueba 1 del PCIO.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 1 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 15 marzo 2023

Carrera: Ingeniería en Desarrollo y Gestión de Software **Cuatrimestre:** 9 **Grupo:** f1

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Game {
    Game() {
        System.out.println("Game constructor");
    }
}

class BoardGame extends Game {
    BoardGame(int i) {
        System.out.println("BoardGame constructor "+i);
    }
    static {
        System.out.println("BoardGame class");
    }
}

public class Chess extends BoardGame {
    {
        System.out.println("Chess object");
    }
    Chess() {
        super(5);
        System.out.println("Chess constructor");
    }
    public static void main(String[] args) {
        new Chess();
    }
}
```

Salida:

"Board Game constructor
"Chess constructor"

Código 2:

```
class BeautifulCar {
    void SpeedUp(){
        System.out.println("SpeedUp()");
    }
    static {
        System.out.println("A Beautiful Car {}");
    }
    BeautifulCar(){
        System.out.println("BeautifulCar()");
    }
}

public class Ferrari extends BeautifulCar {
    { System.out.println("Ferrari {}"); }
    public static void main(String[] args) {
        BeautifulCar f = new Ferrari();
        f.SpeedUp();
        new Ferrari();
    }
}
```

Salida:

"Ferrari A Beautiful Car
SpeedUp"

Nota: no hay errores de sintaxis, ambos códigos arrojan una salida de acuerdo con las impresiones en consola.

Figura 4.4: Ejemplo 4 Prueba 1 del PCIO.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 1 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 15/Marzo/2023

Carrera: Ing. Desarrollo y gestión de software **Cuatrimestre:** 9no **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Game {
    Game() {
        System.out.println("Game constructor");
    }
}

class BoardGame extends Game {
    BoardGame(int i) {
        System.out.println("BoardGame constructor "+i);
    }
    static {
        System.out.println("BoardGame class");
    }
}

public class Chess extends BoardGame {
    {
        System.out.println("Chess object");
    }
    Chess() {
        super(5);
        System.out.println("Chess constructor");
    }
    public static void main(String[] args) {
        new Chess();
    }
}
```

Salida:

#Game Constructor
#Board Game Constructor i
#Chess object
#Chess constructor "i"

//Puede ser un cuadro 5x5

Código 2:

```
class BeautifulCar {
    void SpeedUp(){
        System.out.println("SpeedUp()");
    }
    static {
        System.out.println("A Beautiful Car {}");
    }
    BeautifulCar(){
        System.out.println("BeautifulCar()");
    }
}

public class Ferrari extends BeautifulCar {
    { System.out.println("Ferrari {}"); }
    public static void main(String[] args) {
        BeautifulCar f = new Ferrari();
        f.SpeedUp();
        new Ferrari();
    }
}
```

Salida:

Beatiful car (Ferrari)
SpeedUp Ferrari

Nota: no hay errores de sintaxis, ambos códigos arrojan una salida de acuerdo con las impresiones en consola.

Figura 4.5: Ejemplo 5 Prueba 1 del PCIO.

Después de obtener el nivel de conocimiento de los estudiantes, se llevó a cabo un proceso de aprendizaje de una semana en la cual se analizaron diferentes códigos de programación que incluían los elementos necesarios para abarcar el PCIO, en este lapso de tiempo se hizo uso del módulo de aprendizaje para conocer los pasos de este, asimismo para comprender los bloques de código utilizados. El código 4.1 y 3.7 fueron los más completos y extensos que se utilizaron para que los estudiantes visualizaran y comprendieran el funcionamiento del PCIO paso a paso.

Las dudas más comunes fueron:

1. ¿En qué parte del código se inicia la ejecución?
2. ¿Cómo funciona la jerarquía de clases?
3. ¿Por qué el constructor se ejecuta al final?

Con la finalidad de que los estudiantes comprendieran mejor las líneas de código se realizaron varios ejercicios de ejecución de código en hojas de papel y posterior a eso se replicaron en la herramienta de aprendizaje para obtener el resumen de ejecución.

En este caso de estudio se observó la falta de comprensión que existe en los estudiantes sobre el PCIO, la mayoría desconoce el proceso e incluso mencionan no haberlo escuchado antes. A pesar de que todos consideraban manejar objetos, al realizar la primera prueba se determinó la falta de comprensión que existe. Esto determinó el nivel de conocimiento en los alumnos para establecer el tiempo de aprendizaje suficiente antes aplicar la segunda prueba.

Código 4.1: Código Java para su análisis con estudiantes.

```
class Energia {
   { System.out.println("Energia.expulsión"); }
   Energia(int a) { System.out.println("Energia(" + a + ")"); }

   void m(int a) { System.out.println("m(" + a + ")"); }

   { System.out.println("Energia.radiación"); }
}

class Particula {
   static Energia e1 = new Energia(1);
   Particula(){
      System.out.println("Particula()");
      e2.m(1);
   }
   void p(int a) { System.out.println("p(" + a + ")"); }
   static Energia e2 = new Energia(2);
}

class Enlance {
   Energia e1 = new Energia(3);
   static Energia e2 = new Energia(4);
   Enlance() {
      System.out.println("Enlace()");
      e2.m(2);
   }
   void q (int a) { System.out.println("q(" + a + ")"); }
   static Energia e3 = new Energia(5);
}

class Ciencia {
   Ciencia() { System.out.println("Ciencia()"); }
   { System.out.println("Ciencia moderna"); }
   static { System.out.println("Ciencia formal"); }
}

public class Fisica extends Ciencia {
   static Enlance enlance = new Enlance();

   Fisica() { System.out.println("Fisica()"); }

   public static void main (String[] args) {
      new Enlance();
      new Enlance();
      e.p(1);
      enlance.q(1);

      new Fisica();
      Fisica f = new Fisica();
   }

   static Particula e = new Particula();
}
```

(a) Participante (b) Participante

Figura 4.6: Estudiantes participantes.

4.2. Resultados

Posterior a los días de aprendizaje del PCIO de los estudiantes, se aplicó una segunda prueba para conocer el nivel actual de conocimiento después de haberse familiarizado con este proceso más a detalle.

4.2.1. Prueba 2 del PCIO

En el anexo 2 de este documento se visualiza la segunda prueba aplicada al grupo de estudiantes, en esta se agregó un código con más elementos de los que contiene la prueba 1, en esta ocasión hubo un cambio considerable en la identificación de la salida del código ya que al menos 15 de 22 estudiantes respondieron correctamente.

Las figuras 4.7, 4.8, 4.9, 4.10 y 4.11 muestran 5 ejemplos de estudiantes que lograron comprender de mejor manera el PCIO después de una semana de aprendizaje.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 2 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 09-05-2023

Carrera: IDGS **Cuatrimestre:** 9 **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Component {
    static {
        System.out.println("C");
    }
    {
        System.out.println("ent...");
    }
    Component(int x) {
        System.out.println("Component: " + x);
    }
    Component(){
        System.out.println("Component constructor");
    }
}

class Part extends Component {
    Component c2 = new Component(33);
    {
        System.out.println("Shhh!");
    }
    Component c1 = new Component(-4);
    Part() {
        super(22);
    }
}

public class Process extends Component {
    static Component c = new Component(19);
    Process() {
        System.out.println("Bye Constructor!");
        new Component(5);
    }
    public static void main(String[] args) {
        System.out.println(c);
        new Process();
    }
    static {
        System.out.println("Hi!");
    }
    {
        System.out.println("Bye!");
    }
}
```

Salida:

1 C
2 Ent...
3 Component: 19
4 Hi!
5 @referencia
6 Ent...
7 Component Constructor
8 Bye!
9 Bye Constructor
10 Ent...
11 Component: 5

Nota: no hay errores de sintaxis, el código arroja una salida correcta de acuerdo con las impresiones en consola.

Figura 4.7: Ejemplo 1 Prueba 2 del PCIO.

TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 2 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 09/05/2023

Carrera: Desarrollo y Gestión de Software **Cuatrimestre:** 9° **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Component {
    static {
        System.out.println("C");
    }
    {
        System.out.println("ent...");
    }
    Component(int x) {
        System.out.println("Component: " + x);
    }
    Component(){
        System.out.println("Component constructor");
    }
}

class Part extends Component {
    Component c2 = new Component(33);
    {
        System.out.println("Shhh!");
    }
    Component c1 = new Component(-4);
    Part() {
        super(22);
    }
}

public class Process extends Component {
    static Component c = new Component(19);
    Process() {
        System.out.println("Bye Constructor!");
        new Component(5);
    }
    public static void main(String[] args) {
        System.out.println(c);
        new Process();
    }
    static {
        System.out.println("Hi!");
    }
    {
        System.out.println("Bye!");
    }
}
```

Salida:

C
ent...
Component: 19
Hi!
@Referencia
ent...
Component constructor
Bye!
Bye Constructor
ent...
Component: 5

Nota: no hay errores de sintaxis, el código arroja una salida correcta de acuerdo con las impresiones en consola.

Figura 4.8: Ejemplo 2 Prueba 2 del PCIO.

TECNOLOGICO NACIONAL DE MÉXICO

SEP SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 2 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 09 de mayo del 2023

Carrera: IDGS **Cuatrimestre:** 9no **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Component {
   -static {
        System.out.println("C");
    }
    {
        System.out.println("ent...");
    }
    Component(int x) {
        System.out.println("Component: " + x);
    }
    Component(){
        System.out.println("Component constructor");
    }
}

class Part extends Component {
    Component c2 = new Component(33);
    {
        System.out.println("Shhh!");
    }
    Component c1 = new Component(-4);
    Part() {
        super(22);
    }
}

public class Process extends Component {
    static Component c = new Component(19);
    Process() {
        System.out.println("Bye Constructor!");
        new Component(5);
    }
    public static void main(String[] args) {
        System.out.println(c);
        new Process();
    }
    static {
        System.out.println("Hi!");
    }
    {
        System.out.println("Bye!");
    }
}
```

Salida:

1- C
2- ent...
3- Component: 19
4- Hi!
5- @195componentf64x
6- ent...
7- Component Constructor
8- Bye
9- Bye Constructor
10- ent...
11- Component: 5

Nota: no hay errores de sintaxis, el código arroja una salida correcta de acuerdo con las impresiones en consola.

Figura 4.9: Ejemplo 3 Prueba 2 del PCIO.

TECNOLOGICO NACIONAL DE MEXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 2 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 9 de mayo 2023

Carrera: Tecnologías de la información **Cuatrimestre:** 9 **Grupo:** A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Component {
    static {
        System.out.println("C");
    }
    {
        System.out.println("ent...");
    }
    Component(int x) {
        System.out.println("Component: " + x);
    }
    Component(){
        System.out.println("Component constructor");
    }
}

class Part extends Component {
    Component c2 = new Component(33);
    {
        System.out.println("Shhh!");
    }
    Component c1 = new Component(-4);
    Part() {
        super(22);
    }
}

public class Process extends Component {
    static Component c = new Component(19);
    Process() {
        System.out.println("Bye Constructor!");
        new Component(5);
    }
    public static void main(String[] args) {
        System.out.println(c);
        new Process();
    }
    static {
        System.out.println("Hi!");
    }
    {
        System.out.println("Bye!");
    }
}
```

Salida:

1. C
2. ent...
3. Component: 19
4. Hi
5. Component C
6. ent...
7. Component Constructor
8. Bye!
9. Bye Constructor!
10. ent.
11. Component: 5

Nota: no hay errores de sintaxis, el código arroja una salida correcta de acuerdo con las impresiones en consola.

Figura 4.10: Ejemplo 4 Prueba 2 del PCIO.

TECNOLOGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Prueba 2 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: [redacted] **Fecha:** 09/May/23

Carrera: Desarrollo y gestión de software **Cuatrimestre:** 9no **Grupo:** ~~#6.5~~ A

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1:

```
class Component {
    static {
        System.out.println("C");
    }
    {
        System.out.println("ent...");
    }
    Component(int x) {
        System.out.println("Component: " + x);
    }
    Component(){
        System.out.println("Component constructor");
    }
}

class Part extends Component {
    Component c2 = new Component(33);
    {
        System.out.println("Shhh!");
    }
    Component c1 = new Component(-4);
    Part() {
        super(22);
    }
}

public class Process extends Component {
    static Component c = new Component(19);
    Process() {
        System.out.println("Bye Constructor!");
        new Component(5);
    }
    public static void main(String[] args) {
        System.out.println(c);
        new Process();
    }
    static {
        System.out.println("Hi!");
    }
    {
        System.out.println("Bye!");
    }
}
```

Salida:

1# C
2# ent...
3# Component: 19
4# Hi!
5# @component...
6# ent...
7# Component Constructor
8# Bye!
9# Bye constructor!
10# ent...
11# Component: 5

Nota: no hay errores de sintaxis, el código arroja una salida correcta de acuerdo con las impresiones en consola.

Figura 4.11: Ejemplo 5 Prueba 2 del PCIO.

Durante la realización de la segunda prueba, los estudiantes se mostraron más seguros y menos nerviosos al momento de identificar la salida del código y aunque el tiempo límite para realizar la prueba fue de una hora, la mayoría de estudiantes terminaron antes.

4.2.2. Encuesta sobre el PCIO

Con el propósito de conocer como fue la experiencia de los estudiantes con el PCIO, se realizó una encuesta en la que se obtuvo la información de los gráficos que se muestran a continuación.

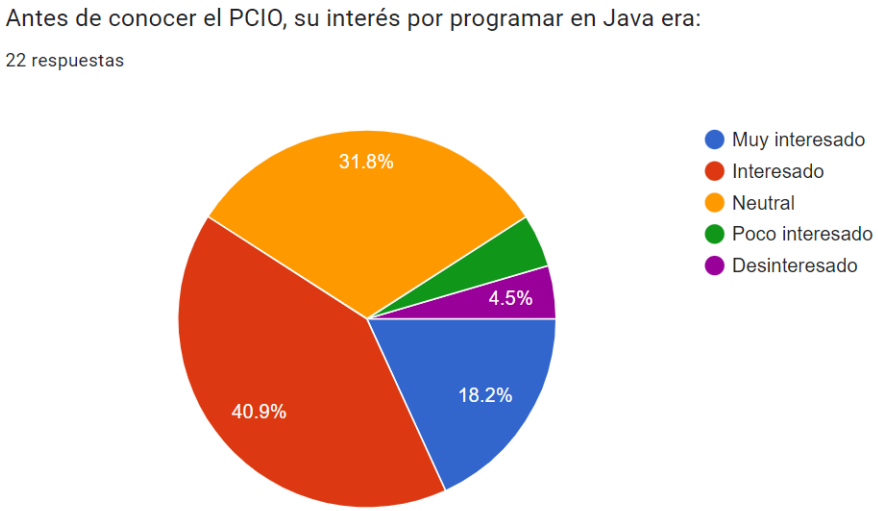


Figura 4.12: Gráfico - Interés por programar en Java antes de conocer el PCIO.

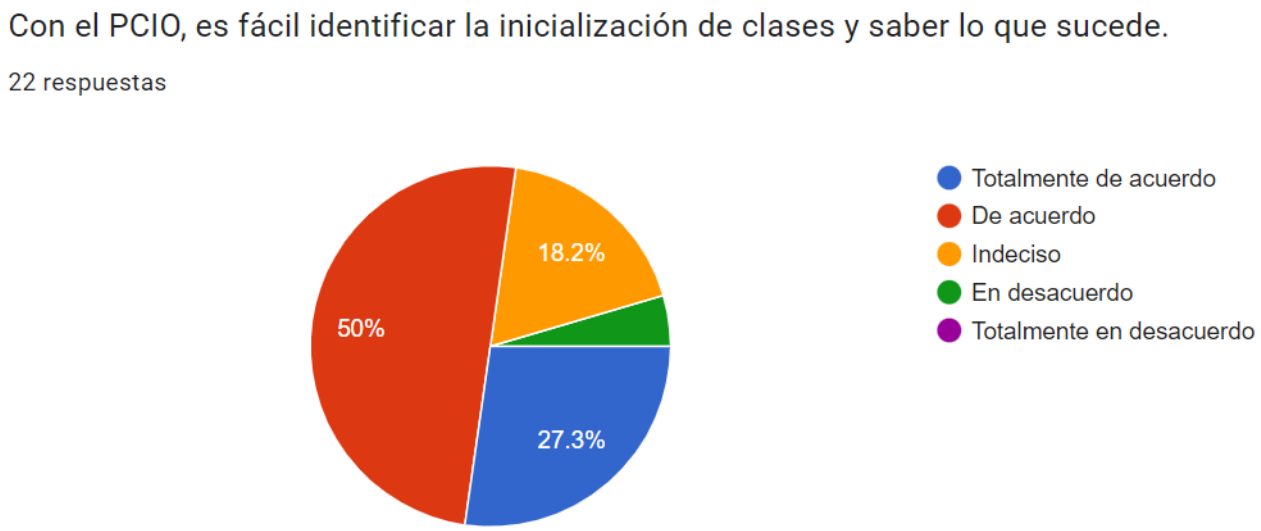


Figura 4.13: Gráfico - Con el PCIO, es fácil identificar la inicialización de clases y saber lo que sucede.

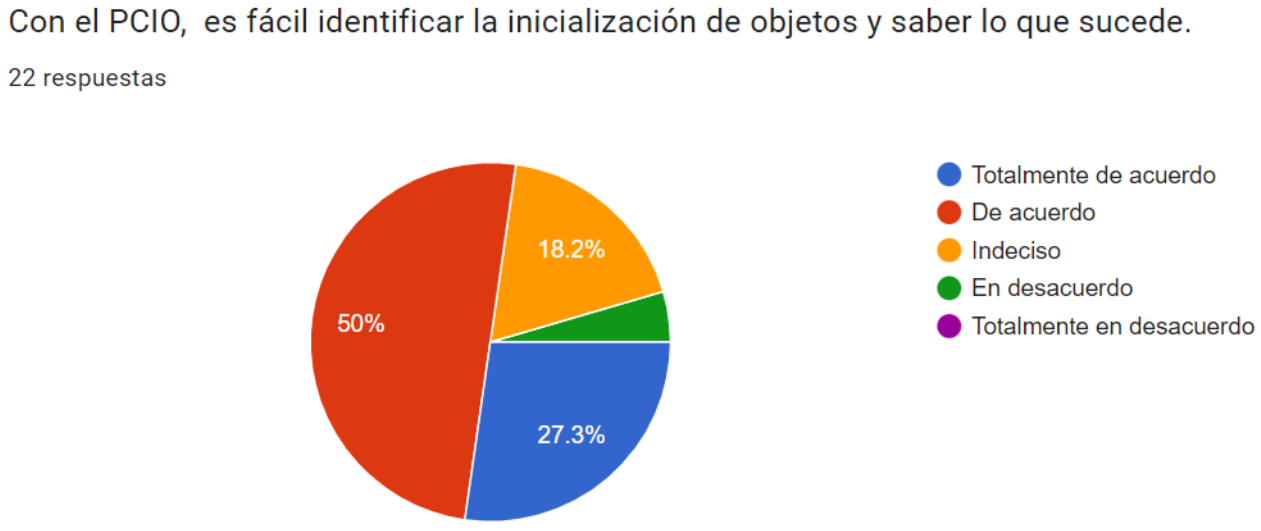


Figura 4.14: Gráfico - Con el PCIO, es fácil identificar la inicialización de objetos y saber lo que sucede.

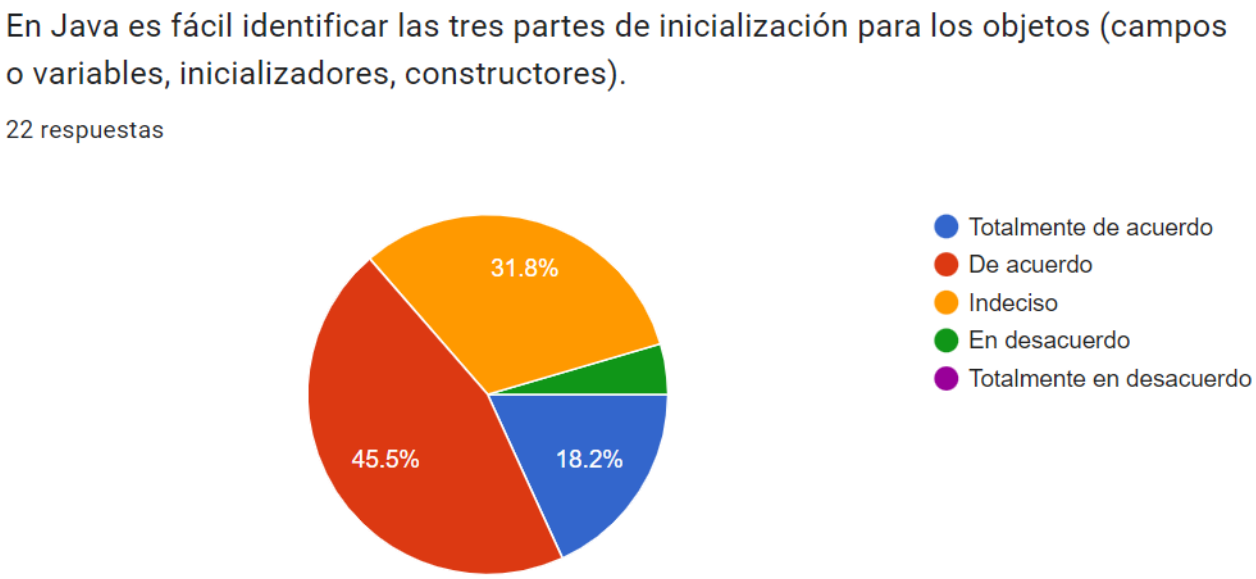


Figura 4.15: Gráfico - En Java es fácil identificar las tres partes de inicialización para los objetos (campos o variables, inicializadores, constructores).

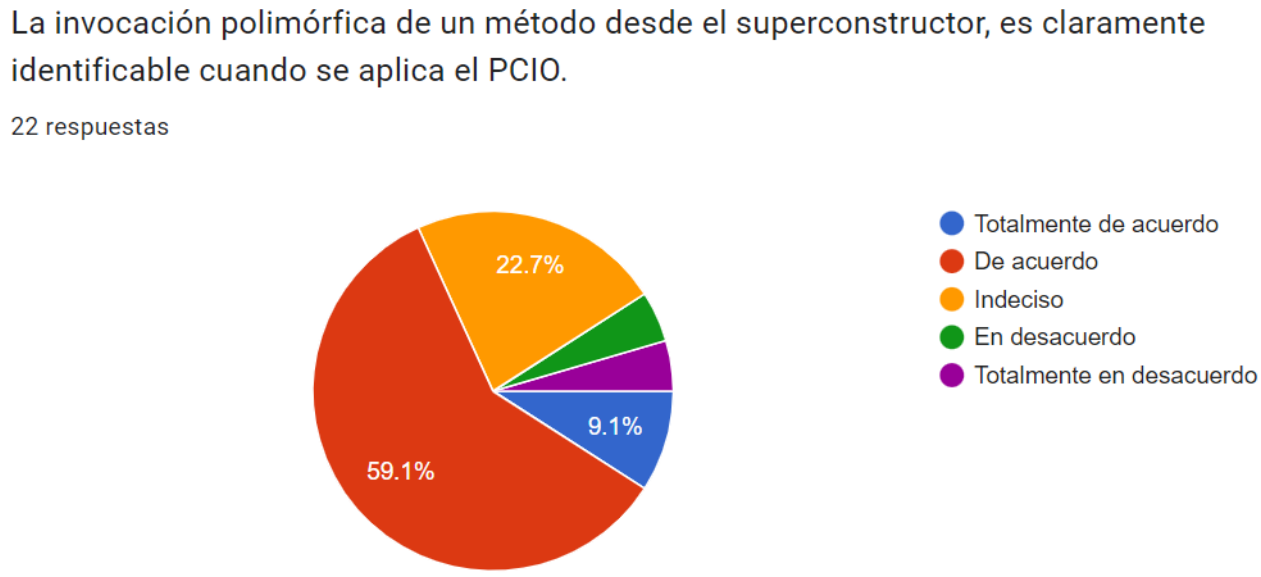


Figura 4.16: Gráfico - La invocación polimórfica de un método desde el superconstructor, es claramente identificable cuando se aplica el PCIO.

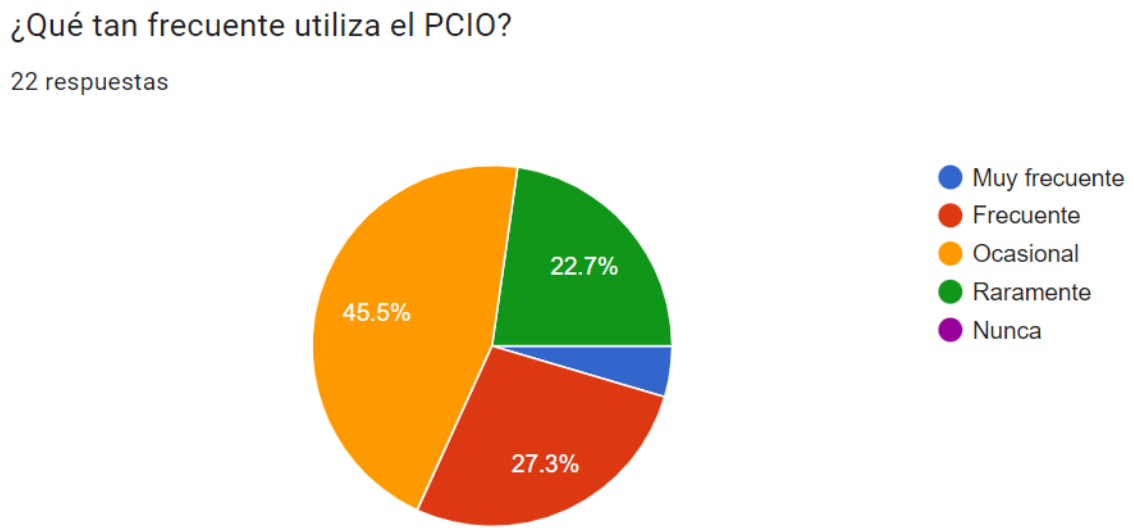


Figura 4.17: Gráfico - ¿Qué tan frecuente utiliza el PCIO?

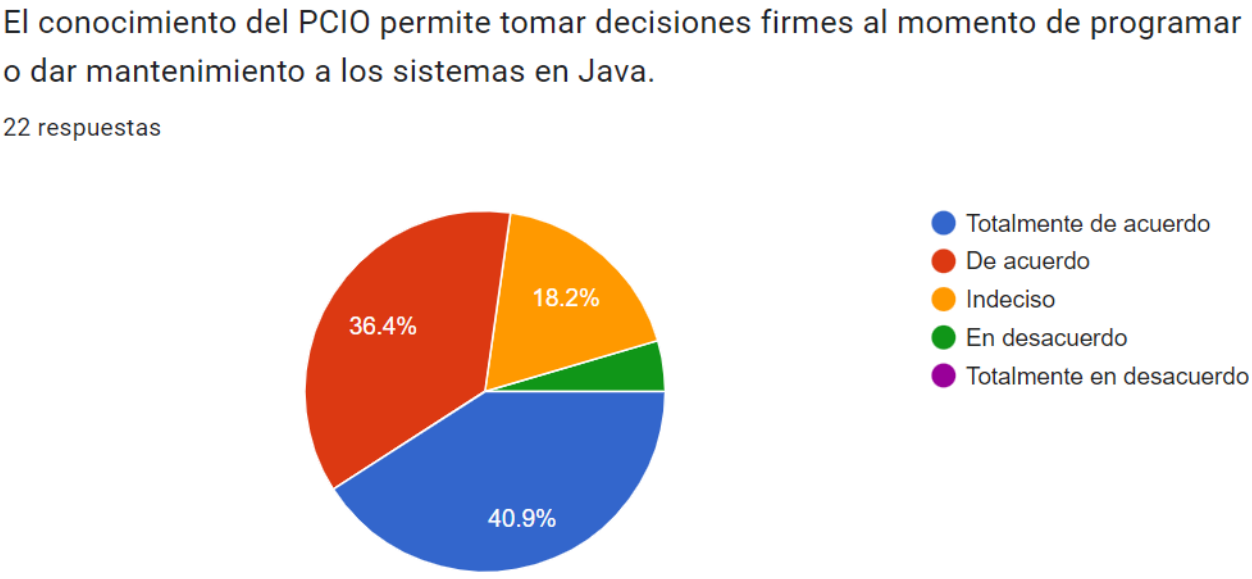


Figura 4.18: Gráfico - El conocimiento del PCIO permite tomar decisiones firmes al momento de programar o dar mantenimiento a los sistemas en Java.

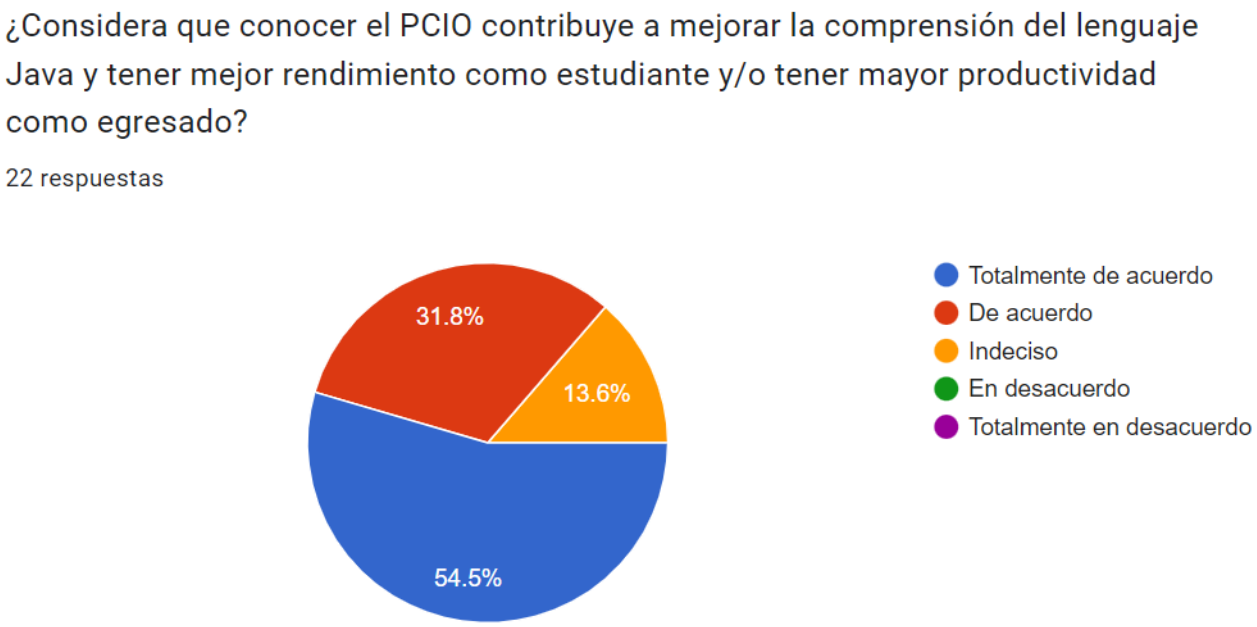


Figura 4.19: Gráfico - ¿Considera que conocer el PCIO contribuye a mejorar la comprensión del lenguaje Java y tener mejor rendimiento como estudiante y/o tener mayor productividad como egresado?

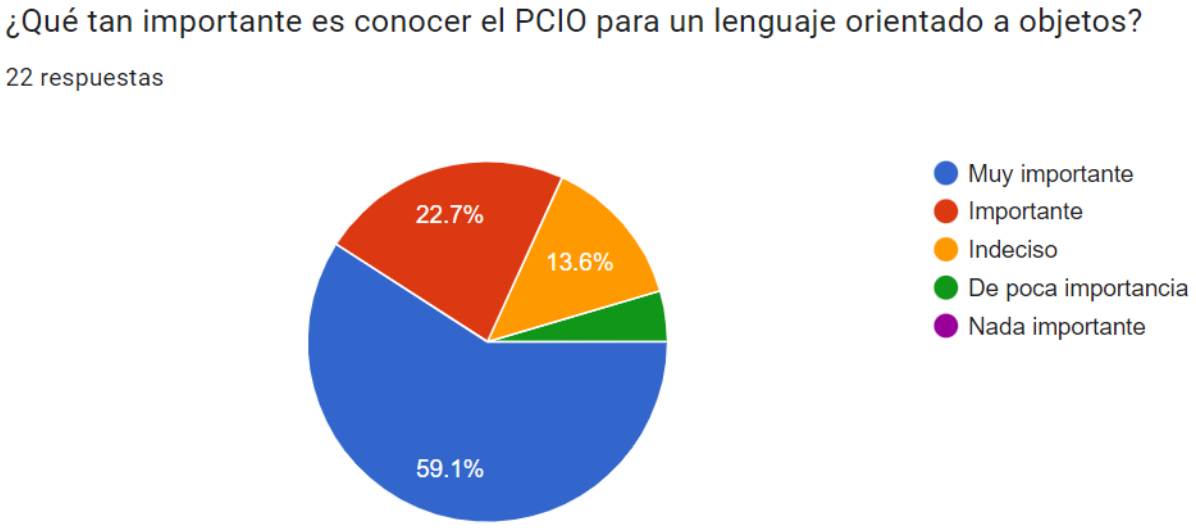


Figura 4.20: Gráfico - ¿Qué tan importante es conocer el PCIO para un lenguaje orientado a objetos?

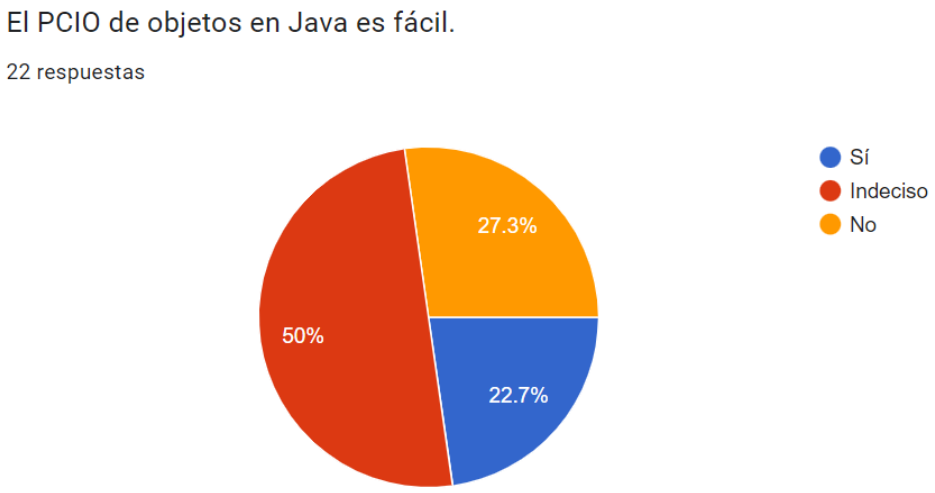


Figura 4.21: Gráfico - El PCIO de objetos en Java es fácil.

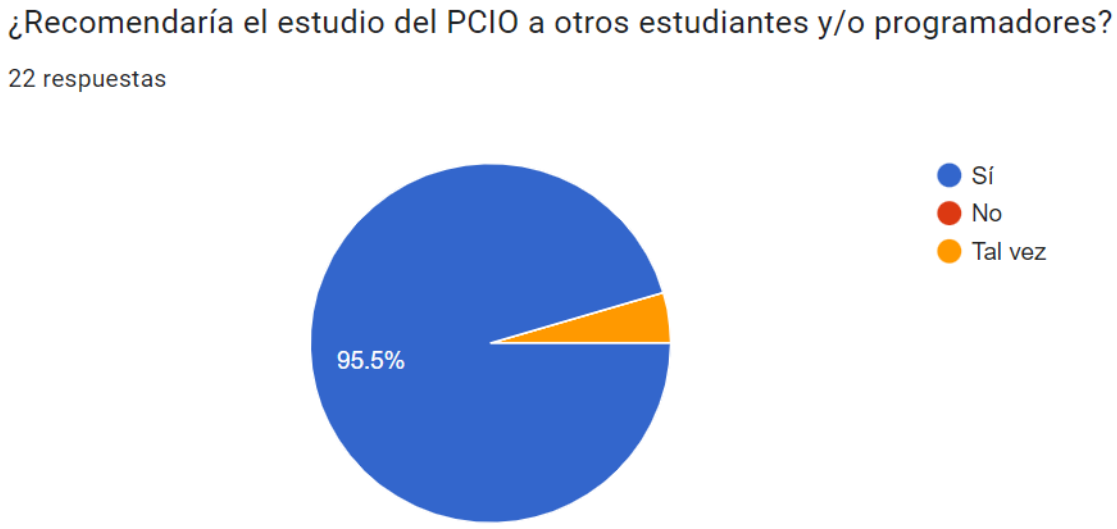


Figura 4.22: Gráfico - ¿Recomendaría el estudio del PCIO a otros estudiantes y/o programadores?

La encuesta aplicada arrojó los siguientes resultados:

En la figura 4.12 se identificó que al menos 40 % del grupo de estudiantes estaban interesados en programar en Java antes de conocer el PCIO, asimismo el 50 % en las figuras 4.13 y 4.14 concuerda que a través del PCIO es fácil identificar clases y objetos para saber lo que sucede al momento de la ejecución.

Para la determinación de las partes de inicialización para objetos, el 45.5 % del grupo respondió que está de acuerdo en que es fácil identificar campos o variables, inicializadores y constructores (figura 4.15); por otro lado en la figura 4.16 solo 2 estudiantes estuvieron en desacuerdo en que la invocación polimórfica de un método desde el superconstructor, es claramente identificable cuando se aplica el PCIO.

Ninguno de los estudiantes dijo nunca haber utilizado el PCIO, todos lo han utilizado en algún momento, el 45.5 % ocasionalmente y el 27.7 % con frecuencia, esto se observa en la figura 4.17. Más del 30 % del grupo, en las figuras 4.18 y 4.19 considera que conocer el PCIO permite tomar decisiones firmes al momento de programar y contribuye a mejorar la comprensión del lenguaje Java.

Casi el 60 % de los estudiantes consideran que es muy importante conocer el PCIO para un lenguaje orientado a objetos (figura 4.20). De igual forma se realizó la siguiente pregunta para responder de forma abierta:

De forma general describe ¿qué es lo que más se le complicó al momento de analizar código Java de acuerdo con el PCIO?

Las respuestas más constantes fueron:

1. Entender el orden de ejecución.
2. Identificar cuáles bloques corresponden a la clase y cuáles al objeto.

3. La jerarquía de clases cuando hay herencia.

4. El punto de partida de la ejecución.

5. El desconocimiento de algunos términos y su implementación.

Finalmente en la figura 4.21 se preguntó si consideran que el PCIO es fácil, a lo que la mitad del grupo dijo estar indeciso, el 27 % considera que no es fácil y el 22 % lo considera fácil. Después de todo el proceso de aprendizaje que el grupo experimentó, en la figura 4.22 el 95 % recomendaría el estudio del PCIO a otros estudiantes y/o programadores.

Capítulo 5

Conclusiones y recomendaciones

En este capítulo se presentan las conclusiones y las recomendaciones correspondientes al proyecto de tesis titulado "Desarrollo de un módulo de software para apoyar el aprendizaje del Proceso de Construcción e Inicialización de Objetos en Java".

5.1. Conclusiones

Java es un lenguaje de programación que, al igual que otros lenguajes orientados a objetos, utiliza los principios del paradigma de POO (encapsulación, abstracción, herencia y polimorfismo), asimismo todos los objetos que se crean en un programa se consideran instancias de una clase predefinida. El proceso de creación de objetos va más allá de hacer una referencia de tipo, sin embargo, e independientemente del lenguaje de programación, en la literatura consultada no se reportó ningún trabajo directamente vinculado con el proceso de construcción e inicialización de objetos. Lo anteriormente descrito se considera un área de oportunidad para el presente trabajo.

Considerando las características que brinda Java para trabajar con objetos y su proceso de construcción, se llevó a cabo el desarrollo de un módulo de aprendizaje como apoyo para comprensión del proceso de construcción e inicialización de objetos. De esta forma se busca que los usuarios adquieran una amplia comprensión del lenguaje Java,

específicamente en el término de objetos.

Igualmente durante el desarrollo de sistemas, es necesario hacerlo de forma consciente a lo que se está programando, es por esto que la correcta comprensión del proceso de construcción e inicialización de objetos es un tema importante para el área de programación.

Finalmente, este proyecto abordó el proceso de construcción de forma total con el propósito de complementar la enseñanza. Una vez finalizado, se aplicó un caso de estudio académico donde se demostró que los estudiantes comprenden mejor el PCIO cuando se utilizan herramientas adicionales, esto se comprueba en la sección 4.2.1 con la aplicación de la segunda prueba. Así, se resalta la importancia e impacto positivo que se obtiene con la realización de este módulo de aprendizaje.

5.2. Recomendaciones

En la actualidad, desde la educación universitaria los estudiantes aprenden programación en diferentes lenguages, sin embargo, al concluir sus estudios el PCIO es completamente incomprendido. Las estrategias que utlizan los docentes para enseñar programación es muy general, el objetivo de este proyecto es abordar el PCIO de forma detallada para alcanzar que estudiantes e incluso docentes tengan más herramientas para conocer este proceso y evitar problemas de comprensión durante su carrera escolar y laboral.

A continuación, se enumeran recomendaciones para futuros trabajos.

1. Agregar otros lenguajes de programación orientados a objetos como C++, C#, Python, PHP, entre otros.
2. Incluir asistente que indique los errores del usuario durante el ensamblado de bloques de código.

Las recomendaciones anteriores surgieron como observaciones durante el caso de estudio.

5.3. Trabajo a futuro

Dentro de los planes a futuro se tiene: llevar a cabo pruebas del módulo de *software* en grupos con estudiantes de posgrado y doctorado; así también es altamente deseable ejecutar pruebas con programadores interesados en alcanzar una certificación Java.

De esta forma se pretende evaluar la herramienta para recibir la realimentación necesaria, realizar mejoras y liberar un módulo multiplataforma.

Anexos

En esta sección se muestran las pruebas en blanco aplicadas durante el caso de estudio, los ejemplos resueltos y no resueltos se encuentran en las secciones 4.1.2 y 4.2.1 de este documento.

Prueba 1 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: _____ **Fecha**: _____

Carrera: _____ **Cuatrimestre**: _____ **Grupo**: _____

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1: **Salida:**

```
class Game {
    Game() {
        System.out.println("Game constructor");
    }
}

class BoardGame extends Game {
    BoardGame(int i) {
        System.out.println("BoardGame constructor "+i);
    }
    static {
        System.out.println("BoardGame class");
    }
}

public class Chess extends BoardGame {
    {
        System.out.println("Chess object");
    }
    Chess() {
        super(5);
        System.out.println("Chess constructor");
    }
    public static void main(String[] args) {
        new Chess();
    }
}
```

Código 2: **Salida:**

```
class BeautifulCar {
    void SpeedUp(){
        System.out.println("SpeedUp()");
    }
    static {
        System.out.println("A Beautiful Car {}");
    }
    BeautifulCar(){
        System.out.println("BeautifulCar()");
    }
}

public class Ferrari extends BeautifulCar {
    { System.out.println("Ferrari {}"); }
    public static void main(String[] args) {
        BeautifulCar f = new Ferrari();
        f.SpeedUp();
        new Ferrari();
    }
}
```

Nota: no hay errores de sintaxis, ambos códigos arrojan una salida de acuerdo con las impresiones en consola.

Prueba 2 del Proceso de Construcción e Inicialización de Objetos en Java

Nombre: _____ **Fecha**: _____

Carrera: _____ **Cuatrimestre**: _____ **Grupo**: _____

Actividad: escribe la salida correcta de cada uno de los códigos que se muestran a continuación.

Código 1: **Salida:**

```
class Component {
    static {
        System.out.println("C");
    }
    {
        System.out.println("ent...");
    }
    Component(int x) {
        System.out.println("Component: " + x);
    }
    Component(){
        System.out.println("Component constructor");
    }
}

class Part extends Component {
    Component c2 = new Component(33);
    {
        System.out.println("Shhh!");
    }
    Component c1 = new Component(-4);
    Part() {
        super(22);
    }
}

public class Process extends Component {
    static Component c = new Component(19);
    Process() {
        System.out.println("Bye Constructor!");
        new Component(5);
    }
    public static void main(String[] args) {
        System.out.println(c);
        new Process();
    }
    static {
        System.out.println("Hi!");
    }
    {
        System.out.println("Bye!");
    }
}
```

Nota: no hay errores de sintaxis, el código arroja una salida correcta de acuerdo con las impresiones en consola.

Bibliografía

[1] B. J. Evans, J. Gough, and C. Newland, *Overview of the JVM*, 1st ed. Sebastopol: O'Reilly Media, Inc., May 2018, pp. 73–76.

[2] R. Edix, "Los lenguajes de programación más usados." [Online]. Available: https://www.edix.com/es/instituto/lenguajes-de-programacion/.

[3] "Tiobe index for january 2023." [Online]. Available: https://www.tiobe.com/tiobe-index/.

[4] Github, "Pypl popularity of programming language." [Online]. Available: https://pypl.github.io/PYPL.html/.

[5] "Types of computer software." [Online]. Available: https://www.educba.com/types-of-computer-software/.

[6] F. J. G. Peñalvo, "Software educativo: evolución y tendencias," pp. 19–29, 2002.

[7] M. C. Contreras, *Integrated Development Environment – IDE.* Birmingham B3 2PB, UK: Packt Publishing Ltd., 2017, pp. 14–14.

[8] "Los 13 tipos de aprendizaje: ¿cuáles son?" [Online]. Available: https://psicologiaymente.com/desarrollo/tipos-de-aprendizaje/.

[9] A. Ávila Freites, N. Quintero, and G. Hernández, "El uso de estrategias docentes para generar conocimientos en estudiantes de educación superior," vol. 16, no. 3, pp. 56–76, 2010.

[10] F. L. Ostenero and A. M. G. Serrano, *Paradigmas de la computación*, 1st ed. Madrid, España: Editorial Centro de Estudios Ramón Areces, S.A., 2014, pp. 5–7.

[11] "¿qué es la tecnología java y por qué la necesito?" [Online]. Available: https://www.java.com/es/download/help/whatis_java.html.

[12] B. Eckel, *Introduction to Objects*, 4th ed. Upper Saddle, New Jersey: Pearson Education, Inc., 2006, pp. 5–6.

[13] G. Booch, R. Maksimchuk, M. Engle, B. Young, and J. Conallen, *Elements of the Object Model*, 4th ed. Boston: Pearson Education, Inc., 2007, pp. 29–74.

[14] L. E. Gutiérrez, C. A. Guerrero, and H. A. López-Ospina, "Ranking of problems and solutions in the teaching and learning of object-oriented programming," pp. 1–30, Feb 2022.

[15] N. Funabiki, Y. Matsushima, T. Nakanishi, K. Watanabe, and N. Amano, "A java programming learning assistant system using test-driven development method," *IAENG International Journal of Computer Science*, vol. 40, no. 1, pp. 38–46, Mar 2013.

[16] K. Zaw, N. Funabiki, E. Mon, and W.-C. Kao, "An informative test code approach for studying three object-oriented programming concepts by code writing problem in java programming learning assistant system," *IEEE 7th Global Conference on Consumer Electronics (GCCE)*, pp. 629–633, Oct 2018.

[17] N. Funabiki, R. Kusaka, and N. Ishihara, "A proposal of test code generation tool for java programming learning assistant system," *IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 51–56, Mar 2017.

[18] J. Keung, Y. Xiao, Q. Mi, and V. C. Lee, "Bluej-uml: Learning object-oriented programming paradigm using interactive programming environment," *International Symposium on Educational Technology (ISET)*, pp. 47–51, Jul 2018.

[19] A. Alwabel, "Coedit: A novel error correction mechanism in compilers using spe-

lling correction algorithms," *Journal of King Saud University - Computer and Information Sciences*, pp. 1–11, Jul 2021.

[20] Q. Liu, "jcab: Making java class design easier for novice programmers," *IEEE Frontiers in Education Conference (FIE)*, pp. 1–6, Oct 2018.

[21] J.-M. Su and F.-Y. Hsu, "Building a visualized learning tool to facilitate the concept learning of object-oriented programming," *6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pp. 516—520, Jul 2017.

[22] E. Kučera, O. Haffner, and R. Leskovský, "Multimedia application for object-oriented programming education developed by unity engine," *Cybernetics & Informatics (K&I)*, pp. 1—8, Jan 2020.

[23] H. Hourani, H. Wasmi, and T. Alrawashdeh, "A code complexity model of object oriented programming (oop)," *IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 560—564, Apr 2019.

[24] Z. Atiq and M. C. Loui, "A qualitative study of emotions experienced by first-year engineering students during programming tasks," *ACM Transactions on Computing Education*, vol. 22, no. 32, pp. 1—23, Jun 2022.

[25] L. Rodrigues, F. Pereira, A. Toda, P. Palomino, W. Oliveira, M. Pessoa, L. Carvalho, D. Oliveira, E. Oliveira, A. Cristea, and S. Isotani, "Are they learning or playing? moderator conditions of gamification's success in programming classrooms," *ACM Transactions on Computing Education*, vol. 22, no. 30, pp. 1—27, Jun 2022.

[26] U. Z. Ahmed, Z. Fan, J. Yi, O. I. Al-Bataineh, and A. Roychoudhury, "Verifix: Verified repair of programming assignments," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 74, pp. 1—31, Jul 2022.

[27] S. Rai, R. C. Belwal, and A. Gupta, "A review on source code documentation," *ACM Transactions on Intelligent Systems and Technology*, vol. 13, no. 84, pp. 1—44, Jun 2022.

[28] Google, “Try blockly.” [Online]. Available: https://developers.google.com/blockly/.

[29] “La biblioteca para interfaces de usuario web y nativas.” [Online]. Available: https://es.react.dev/.

[30] Adobe, “Waterfall methodology: A complete guide,” 2022. [Online]. Available: https://business.adobe.com/blog/basics/waterfall/.

[31] “Intellij idea overview.” [Online]. Available: https://www.jetbrains.com/help/idea/discover-intellij-idea.html/.

[32] “Promise.” [Online]. Available: https://javascript.info/promise-basics/.