Instituto Tecnológico de Costa Rica

Centro Académico de Alajuela

IC210: Programación Orientada a Objetos

Proyecto #1:

Simpletron en Java

Estudiantes:

William Gerardo Alfaro Quiros – 2022437996

Angel Gabriel Vargas Varela - 2021080292

Profesores:

Ing. Allan Cascante

Ing. Martin Flores

Fecha de entrega:

6/09/22

II Semestre, 2022

Simpletron en Java

Este proyecto se basa en la creación de un código en el lenguaje de programación java que pueda ejecutar instrucciones y así simular comportamientos de una computadora. Este simulador es capaz de agregar dos valores y sumarlos, restarlos, dividirlos o realizar cualquier tipo de operación aritmética programada para dos valores. Simpletron es un sistema de registros que contienen un esquema de instrucciones que son cargados en un acumulador mediante el mismo registro de instrucciones. el cual coloca la información para efectuar algún cálculo aritmético o una operación de transferencia de control.

El sistema de Simpletron en cierta manera es similar al lenguaje de ensamblador, el cual funciona mediante registros y no por variables. De esta forma, se utilizan dos variables para luego sumarlas y guardar su resultado en una tercera variable. Esta vez para sumar dos variables deben tener un valor y a su vez, se deben cargar los datos y detener la ejecución de suma, para luego guardar el resultado en una nueva variable y ejecutar el código de finalización el cual despliega un mensaje que indica al usuario que ha finalizado con éxito de cargar y ejecutar las instrucciones. Finalmente, el programa utiliza los datos suministrados por el usuario para realizar la operación programada mediante el set de instrucciones. A grandes rasgos, el programa Simpletron ejecuta programas escritos en un único lenguaje máquina de Simpletron (LMS) que interpreta directamente.

Estrategia de la solución

Para poder empezar el proyecto, fue de suma importancia la revisión extensiva en sitios web para encontrar ejemplos y algunas demonstraciones de programas similares que ejecutan instrucciones mediante mecanismos similares. En un principio fue complejo comprender y comenzar con el proyecto, debido a que no se comprendía del todo cómo se

podría implementar las funciones respectivas ni la distribución apropiada para las clases y sus atributos respectivos.

Después de una basta investigación, se decide crear y declarar cada operación del simpletron, con una clase abstracta que guía la secuencia lógica de los métodos principales y los encapsula mediante los códigos de operación. Además, gracias a la clase abstracta se puede tener un código más ordenado y sin tener una cantidad excesiva de variables dentro de una clase. Finalmente, en base al diagrama de UML se empiezan a construir los métodos necesarios para las operaciones aritméticas y los controles de flujo.

Detalles de implementación

Se creó una clase principal llamada Terminal, la cual tendrá un objeto de tipo Simpletron, ya que dentro de Simpletron realizamos todas las implementaciones de los registros e instrucciones para las operaciones. Con el objetivo de que la clase Simpletron funcione de una manera correcta, se creó una clase abstracta llamada *CodigosOperaciones()*, la cual mantiene cada una de las variables con la etiqueta de *protected*, para poder heredar sus atributos a la clase *Simpletron*, de la cual extenderemos la clase *CodigosOperaciones*.

Una vez obtenidas las clases organizadas, surgen las operaciones de nuestro Simpletron, para estos se estableció un método llamado *execute( )*. La clase *execute* tomará el comando digitado por el usuario, como por ejemplo <+10 007> y lo almacenará en un arreglo de instrucciones, hasta que se ejecute -99999, el cual finaliza la carga de instrucciones del programa y se llamará a los métodos *terminarInstruccion( )*y *funciones( )*. En el método de *terminarInstruccion( )* lo que se toma el registro introducido por el usuario y se procede a aplicarle una división entera a dicha instrucción y posteriormente el módulo, ya que con esto se simplifica la expresión y es posible tomar los dígitos de la “palabra”

introducida por el usuario. Después de simplificar la instrucción, el método *funciones( )* por medio de *cases* revisará el índice de la instrucción y realizará la respectiva operación.

Restricciones o suposiciones

Una gran restricción a la hora de hacer el programa es que se dificulta realizar las validaciones, debido a que es un tema nuevo o en este caso, fue un conflicto que nos habría gustado arreglar o poder mejorar durante este proyecto. Por un lado, al ser el primer curso en el que se introduce al estudiante al lenguaje de programación Java, surgen muchas dudas en cuanto al syntax utilizado. Por ejemplo, para lograr tomar los valores desde el usuario, imprimirlos, crear arreglos, entre otros. Por otro lado, el manejo de clases y el instanciarlas en otras clases, el paso de parámetros o la herencia de atributos puede complicarse un poco a lo largo de la estructura del programa.

Debido a razones de tiempo no se logró exitosamente implementar más excepciones durante en el proyecto, además de que se supuso que quizás una gran forma de mejorarlo en el próximo proyecto es administrar e investigar más a profundidad temas que nos puedan dar un mejor manejo y uso del programa, así como hacer un mejor uso del tiempo.

Problemas encontrados

Surgieron problemas a la hora de crear los registros y hacer que el usuario los tuviera que escribir por terminal, además de que se complicó mucho entender la lógica del simpletron, ya que había que programar instrucciones de manera lógica y en cierta medida había que buscar la lógica de cada instrucción. Especialmente, las funciones de bifurcación resultaron difíciles de implementar y comprender a pesar de ser similares a las del lenguaje de programación Python. Durante el proyecto aparecen problemas de “bugs” o a veces no funcionaban las instrucciones correctamente debido a que el programa no las leía como

corresponde Finalmente se logró comprender que no se debía de leer como tal toda la instrucción, sino su índice y el último número.

Al inicio del proyecto, se crearon solo dos clases, pero al ver que se veía muy desorganizado, se optó por utilizar una clase abstracta y resumir una de nuestras clases, para así mantener un orden y hacer uso de la modularización. A su vez, la creación del proyecto Maven no resultó complicada; sin embargo, a la hora de editar los plugins del archivo pom.xml se encontraron algunos contratiempos al modificar el archivo debido a que contenía algunas dependencias que debían ser modificadas. Por otro lado, las versiones de la versión del compilador de Maven tuvieron que ser reemplazada por v.11 para que funcionara apropiadamente, así como el directorio donde se encuentra la función principal del programa.

Finalmente, el manejo de memoria requiere de atención dado que el acumulador necesita estar en constante modificación del método memoria para poder implementar los valores correctamente.

Diagramas de clase:

Imagen 1: Diagrama de clases UML de Simpletron.

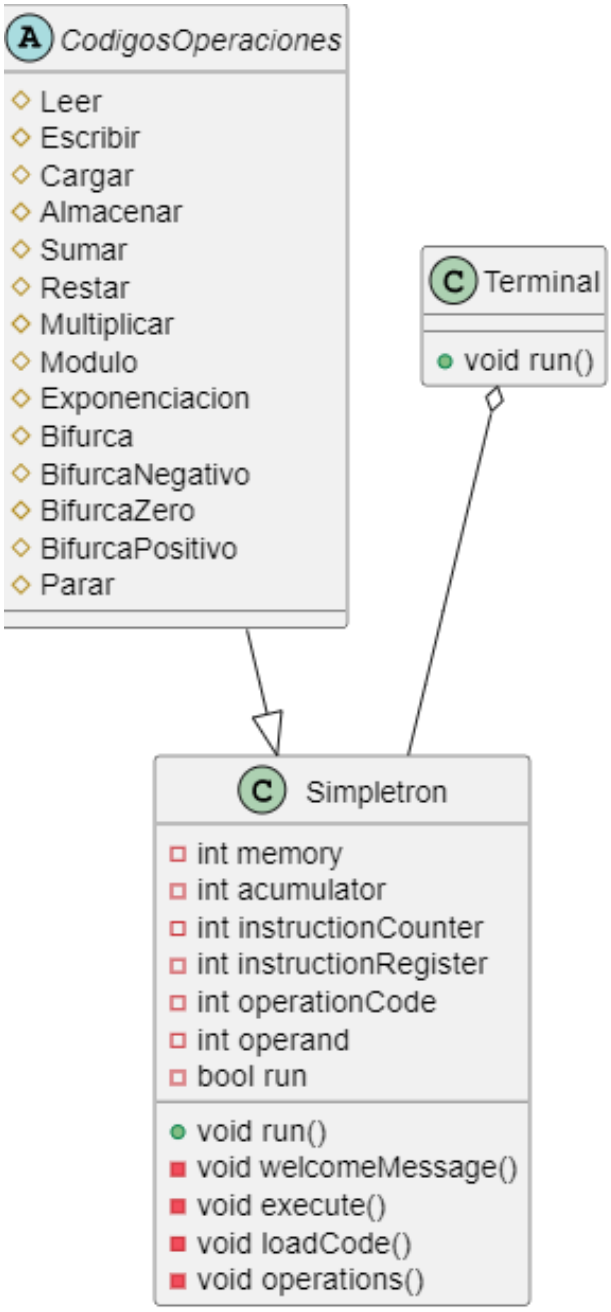

Imagen 2: Código de diagrama de clases hecho con Visual Studio Code.

Conclusiones y recomendaciones:

Este proyecto ayudó a los integrantes a comprender de mejor manera la utilización de la programación orientada a objetos, además que por primera vez implementamos una clase abstracta en Java, lo cual permitió aprender sobre la gran importancia de estas y como complementar con el proyecto.

Simpletron en cierto modo permitió comprender de mejor manera otros lenguajes como ensamblador puesto que utiliza una lógica similar; sin embargo, al implementar esta lógica, vemos de mejor manera cómo se desarrollan los algoritmos de cada lenguaje de programación de bajo nivel. Además de utilizar java, se pudo poner en práctica la

utilización de *PlantUML* y se comprendió mejor cómo realizar un diagrama UML con un proyecto hecho por los estudiantes, poniendo en práctica el pensar y analizar la implementación del código de una manera más simplificada. Finalmente, mediante la investigación de las funciones necesarias para la implementación del Simpletron, se encontraron métodos y algoritmos útiles para el manejo de los números y la memoria dentro de un programa orientado a objetos.

Anexo

Autoevaluación

William Alfaro Quiros

Criterio	**1**	**2**	**3**	**4**
Respeto a los demás				**x**
Tolerancia a las diferencias				**x**
Contribución				**x**
Integración a grupo				**x**

Angel Vargas Varela

Criterio	**1**	**2**	**3**	**4**
Respeto a los demás				**x**
Tolerancia a las diferencias				**x**
Contribución				**x**
Integración a grupo				**x**

Coevaluación

William Alfaro Quiros

Criterio	**1**	**2**	**3**	**4**
Respeto a los demás				**x**
Tolerancia a las diferencias				**x**
Contribución				**x**
Integración a grupo				**x**

Angel Vargas Varela

Criterio	**1**	**2**	**3**	**4**
Respeto a los demás				**x**
Tolerancia a las diferencias				**x**
Contribución				**x**
Integración a grupo				**x**