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:

Allan Cascante

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. Simpletron es un sistema de registros que contienen un esquema de instrucciones que son cargados en un acumulador 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 ya que para realizar una suma no solo 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.

Estrategia de la solución:

Para poder empezar el proyecto, fue de suma importancia ver ejemplos y algunas explicaciones acerca del simpletron en sitios de codigo abierto de internet. En un principio fue complejo comprender y comenzar con el proyecto, debido a que no se comprendia del todo cómo se podría implementar las funciones respectivas

Después de tanta investigación, decidimos hacer o declarar cada operación del simpletron, con una clase abstracta, gracias a que con esto podemos tener las operaciones en un archivo diferente sin que se vea afectado al código de las operaciones, además de que con esta clase abstracta podemos tener un código más ordenado y sin tener una cantidad excesiva de variables dentro de una clase.

Detalles de implementación:

Creamos 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 la operaciones.

Solo que para que la clase Simpletron funcione de una manera correcta, creamos 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.

Ya que tenemos las clases ordenadas, llegó la hora de explicar la clase Simpletron, ya que de ella surgen las operaciones de nuestro Simpletron, para ellos creamos un método llamado run, con el cual vamos a llamar dos métodos los cuales serían welcomeMessage y execute.

En welcomeMessage se imprime un mensaje de bienvenida, en donde se explica el uso del simpletron.

La clase execute tomará el comando digitado por el usuario, como por ejemplo +10007 y los almacenará en una matriz de instrucciones, hasta que se ejecute -99999. En el método execute se llamará a los métodos loadCode y operations, ya que mientras no se digite -99999 el programa los seguirá ejecutando.

En el metedo de loadCode lo que se hará es agarrar el registro introducido por el usuario y dividir dicha instrucción con la división entera y el módulo, ya que con esto podremos simplificar la expresión y tomar el dato que necesitemos.

Después de simplificar la instrucción, el método operations por medio de case 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.

Debido a razones de tiempo no pudimos implementar más excepciones durante en el proyecto, además de que suponemos que quizás un 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.

Problemas encontrados:

Hubo problemas a la hora de crear los registros y hacer que el usuario los tuviera que escribir por pantalla, además de que se nos 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.

Durante el proyecto hubo problemas de bugs o a veces no funcionaban las instrucciones, debido a que no las leía como corresponde, todo gracias a que no debía de leer como tal toda la instrucción, sino su índice y el último número.

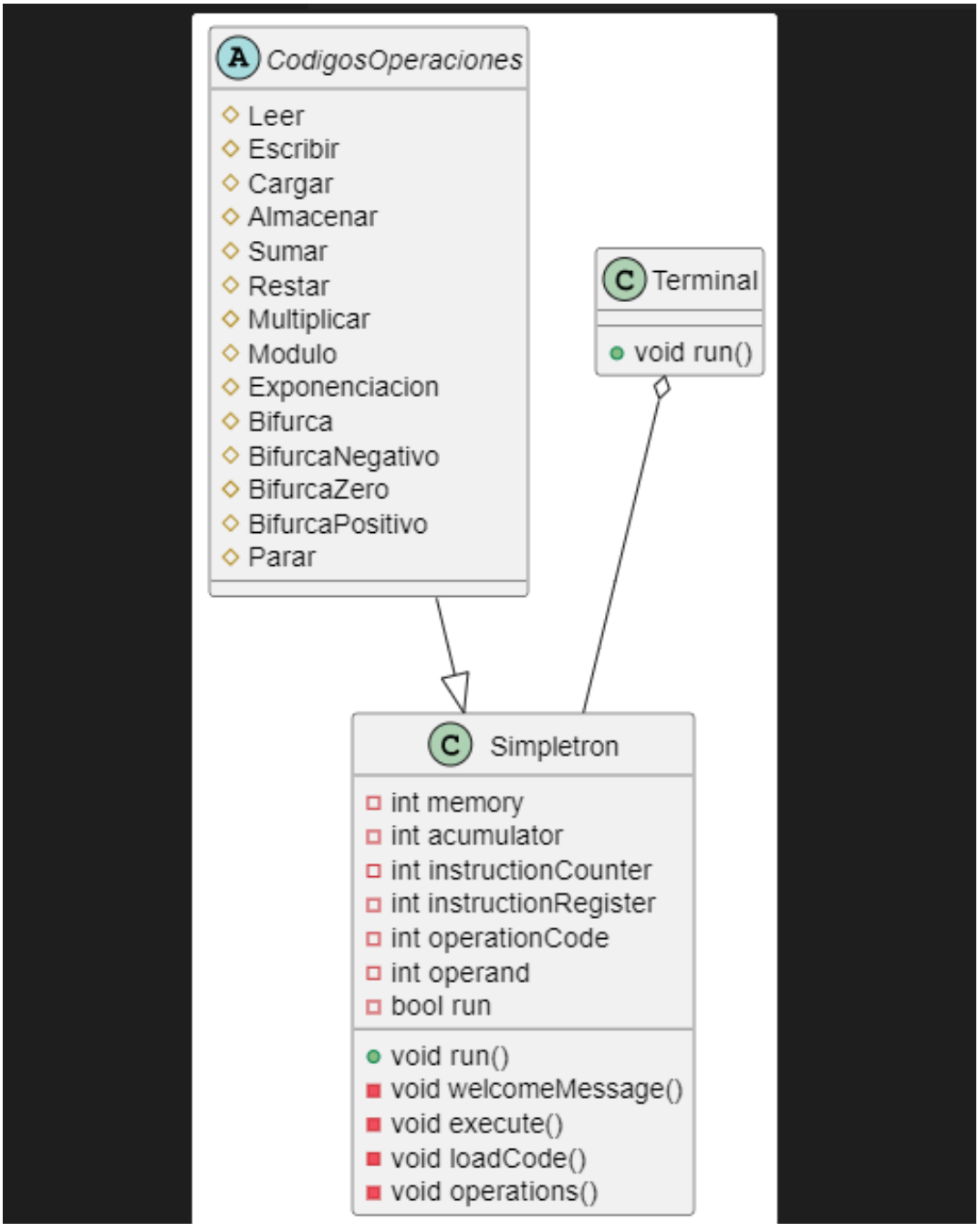
Al inicio del proyecto, tuvimos solo dos clases, pero al ver que se veía muy desordenado, optamos por utilizar una clase abstracta y resumir una de nuestras clases, para así mantener un orden e implementar la modularización.

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 puesto que contenía algunas dependencias que debían ser modificadas. Por otro lado, las direcciones de la versión del compilador de Maven tuvo que ser reemplazada por v..11 para que funcionara apropiadamente.

Finalmente, el manejo de memoria requiere de atención puesto que el acumulador debe de estar en constante modificación de nuestro método memoria para poder implementar los valores correctamente.

Diagramas de clase:

Diagrama de clases UML



Código de diagrama de clases hecho con Visual Studio Code

Conclusiones y recomendaciones:

Este proyecto nos ayudó mucho para comprender de mejor manera la utilización de la programación orientada a objetos, además que por primera vez implementamos una clase abstracta, aprendiendo la gran importancia de estas y como complementar con el proyecto.

Simpletron en cierto modo nos ayudó a comprender de mejor manera otros lenguajes como ensamblador, ya que utiliza una lógica similar, pero al realizar esta lógica, vemos de mejor manera cómo se implementan los algoritmos de cada lenguaje de programación a nivel interno. Además de utilizar java, se pudo poner en práctica la utilización de PlantUML y como realizar un diagrama UML con un proyecto hecho por nosotros, obligandonos en cierto modo a pensar y analizar la implementación del código de una manera más simplificada.

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**