Proyecto #1 Simpletron

Curso Programación Orientada a Objetos

Grupo 20

Estudiante:

Marco Rodríguez Vargas Carnet: 2022149445

Manfred Jones Sanabria Carnet: 2022176476

II Semestre

Profesores:

Martin Flores

Allan Cascante

Fecha de entrega:

6 de octubre del 2022 a las 11:59 pm.

Índice

Simpletron

Introducción

Simpletron es un simulador de una computadora abstracta, básicamente es una máquina simple pero poderosa, que solo ejecuta programas escritos en el único lenguaje que entiende, el lenguaje máquina de Simpletron (LMS). Simpletron para efectos de este proyecto fue desarrollado por medio del lenguaje de programación Java, utilizando el paradigma de programación orientada a objetos.

Estrategia de la solución

Nuestra estrategia para la solución del proyecto Simpletron dio comienzo con la primera fase para resolver un problema en programación, analizar el problema y determinar cuales son los requerimientos del problema (este paso es importante porque si algo falla en este primer paso el resto del proyecto presentará una solución poco acertada). La siguiente fase fue crear el diagrama de clases para tener una mejor visión de la posible solución que se planteó en la fase anterior. Una vez tuvimos el diagrama de clases nos encargamos de crear un pseudocódigo para orientarnos mejor con la posible implementación de la solución planteada por el equipo.

Detalles de implementación

El programa posee 4 clases: InitializeSimpletron, Simpletron, Memory y Operation; en orden de jerarquía. La clase InitializeSimpletron se encarga de inicializar el programa, como su nombre lo indica, mediante la creación de un objeto de la clase Simpletron. Esta contiene el método main.

La clase Simpletron es donde se realiza la lectura de las instrucciones y donde se cargan individualmente para ser ejecutadas, en conjunción con Memory y Operation. Simpletron tiene un objeto de tipo Memory, además de un constructor que imprime el mensaje de bienvenida e inicializa los métodos de leer y ejecutar. El primer método posee un input y lleva un contador de la localidad en memoria, este va a leer la entrada del usuario, donde si está se encuentra en el rango -99998, 99998, va a ser introducida a la memoria al llamar a un método de la clase Memory. En

caso de que esté fuera del rango se notifica de error, y si la entrada es -99999, se termina la leída.

El segundo método lleva el contador de instrucción y define la instrucción actual accediendo a una variable de instancia de Memory. Este llama a un método de la clase Memory, el cual carga la instrucción actual para ser ejecutada. Se cuentan con condicionales que controlan el flujo en caso de bifurcación: en caso de que la variable de instancia “branch” sea diferente de 0, el contador de instrucción pasa a ser el contenido de “branch”, el cual es una localidad de memoria. El programa termina si “branch” es -1, lo que fija la instrucción actual a 44000 y finaliza.

La clase Memory tiene un arreglo de enteros que corresponde a las instrucciones en memoria, el acumulador, “branch” y el registro especial. El método addMemory simplemente coloca la entrada en la posición de memoria, y el método loadInstructionMemory define el registroInstruccion a partir del arreglo y el contador, crea un objeto de tipo Operation, con registroInstruccion como argumento. Se crea un arreglo para manejar las tres variables enteras y se llama al método operations de la clase Operation. Cuando se ejecuta la operación, se modifican las tres variables a partir del arreglo local.

La clase operation posee la variable del operando y la del código, las cuales se inicializan con un constructor a partir del registroInstruccion, donde el código son los primero dos dígitos y el operando los últimos tres. El método operations se encarga de realizar las operaciones en sí. En este inicialmente se define un arreglo para almacenar el acumulador, branch y registro especial. Cuenta con un switch para manejar los diferentes códigos, donde en cada uno ya sea se pide o imprime un valor, se realiza una operación aritmética con el acumulador, realizar un salto o bifurcación a una localidad en memoria, inicializar el controlador de ciclos, o terminar el programa. En las operaciones donde se deba modificar el acumulador, branch o registro especial, se realiza la modificación en el arreglo y al final se retorna. Si se recibe un código que no es ninguno de los casos, se procede con un error, además, también se verifican errores de división entre cero y desbordamiento del acumulador, todos terminando el programa.

Restricciones o suposiciones

Al momento de leer las entradas, se supone que el usuario va a introducir instrucciones congruentes y el línea con el funcionamiento de Simpletron y LMS, por ejemplo:

000> 10002
001> 11002
002> 00000
003> 44000
004> -99999

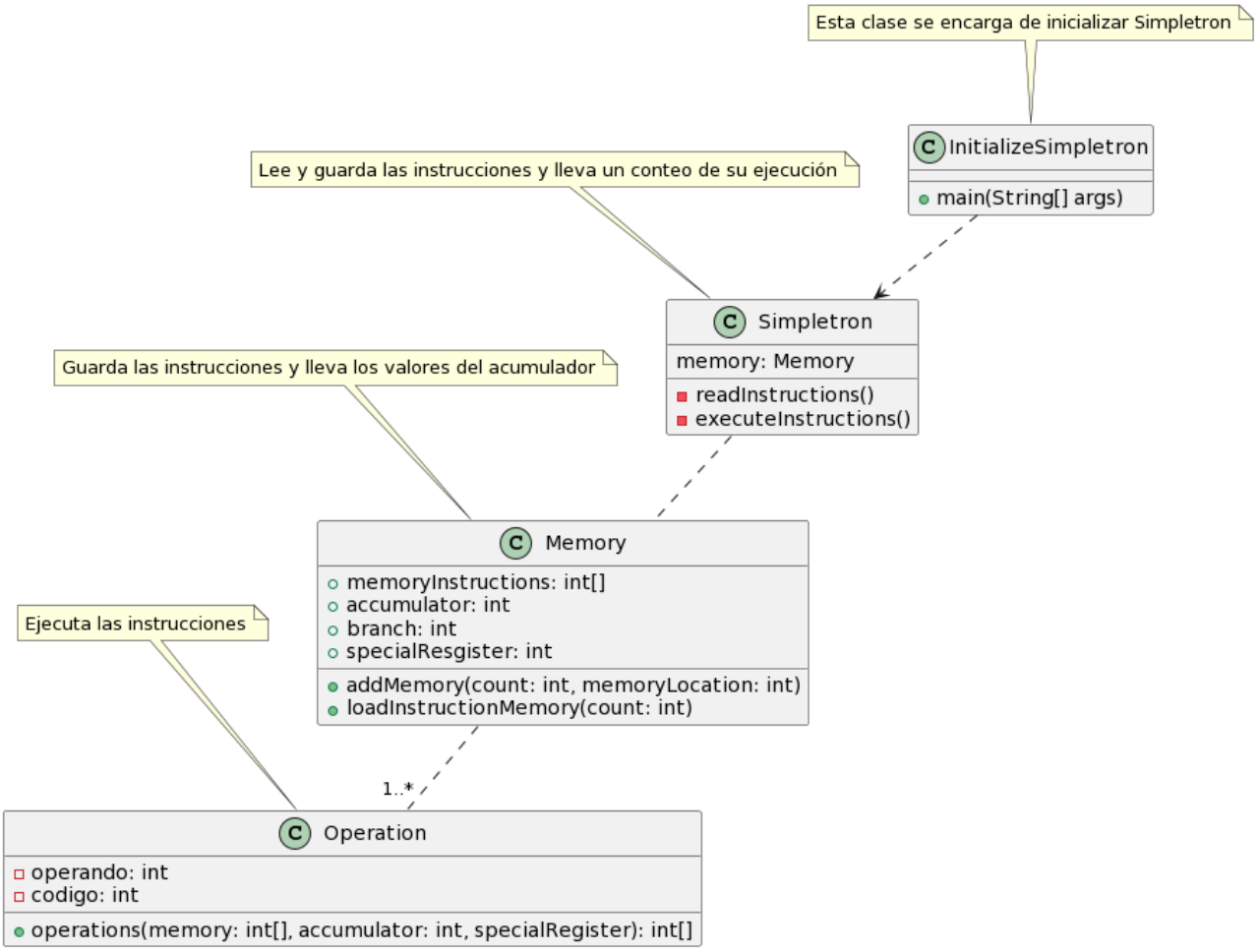
Estas instrucciones se pueden usar para recibir una variable, guardarla en la localidad de memoria 002 y luego imprimirla, sin embargo, como se asigna el espacio vacío antes de finalizar (instrucción 44000), cuando se llegue a la localidad 002 se va a intentar ejecutar la instrucción pero daría error de código, por lo que las localidades de memoria que se quieran destinar para guardar valores deben ser asignadas después de la última instrucción 44000 pero antes del valor centinela -99999, para un correcto funcionamiento de Simpletron.

Problemas encontrados

Al inicio, durante la fase de planeamiento, no estábamos seguros con cómo modelar las clases a usar en el programa y que variables y métodos podríamos agregar en cada una, para esto realizamos el diagrama de clases y determinamos las clases específicas con sus miembros.

En las operaciones donde se realiza una bifurcación del programa, al inicio nos encontrábamos en confusión con la forma en implementar esta característica, por lo que optamos por definir la variable “branch” a nivel del objeto memory, la que puede ser modificada en caso de salto o bifurcación. La misma se utiliza para finalizar la ejecución del programa.

Diagramas de clases



Conclusiones y recomendaciones

Finalmente, podemos afirmar que la implementación de Simpletron es posible por medio del paradigma de programación orientado a objetos. Como se ha podido observar, el proyecto ayudó a reforzar los conceptos aprendidos en clase. De esta manera, se consiguió aplicar lo estudiado durante el curso y como se puede llegar a implementar en código la programación orientada a objetos. En suma, el presente proyecto de manera simultánea ayudó a trabajar los pilares de la ingeniería en software y la importancia de la calidad.