



INSTITUTO TECNOLÓGICO DE IZTAPALAPA
INGENIERIA EN SISTEMAS COMPUTACIONALES

REPORTE DEL PROYECTO FINAL

TEMA:

Random-Access Machine (secuencial)

Asignatura:

LENGUAJES Y AUTOMATAS 1

PROFESOR:

Abiel Tomas Parra Hernández

PRESENTA:

(INTEGRANTES), NUM CONTROL

Jiménez Leyva Cesar, 171080169 -45%

José Gabriel Sánchez Sánchez, 171080010-45%

Víctor Manuel Maravilla Flores 171080140 1%

Pérez González Fernando Daniel 171080092 -9%

CIUDAD DE MÉXICO

Junio/2021



INDICE

Resumen - Random Accces Machine (RAM)	4
INTRODUCCIÓN.....	5
OBJETIVOS	6
JUSTIFICACIÓN.....	7
MARCO TEÓRICO (Conceptos y fundamentos).....	8
METODOLOGÍA DE TRABAJO	16
DESARROLLO E IMPLEMENTACIÓN.....	18
RESULTADOS.....	21
CONCLUSION	25
FUENTES DE INFORMACIÓN.....	26



INDICE DE FIGURAS

Ilustración 1- Representación gráfica básica de una RAM).....	12
Ilustración 2.....	13
Ilustración 3.....	14
Ilustración 4.....	14
Ilustración 5.....	15
Ilustración 6 Desarrollo de código	19
Ilustración 7 Desarrollo de código	19
Ilustración 8 Desarrollo de código	20
Ilustración 9 Desarrollo de código	20
Ilustración 10.....	21
Ilustración 11.....	22
Ilustración 12.....	22
Ilustración 13.....	23
Ilustración 14.....	24
Ilustración 15.....	24

RESUMEN - Random Access Machine (secuencial) (RAM-SECUENCIAL)

Existen 3 cosas que nuestro modelo básico de maquina necesita:

- Memoria
- Entradas/ Salidas
- Programación

Se conocen varios modelos para definir una Maquina de Acceso Aleatorio, pero utilizaremos uno Definido por Steven Skinea en su libro "The Algorithm Desing Manual".

Hablemos de la memoria; este se usa para las entradas y salidas, y tambien para la ejecución del programa, pero para simplificar las cosas, dividamos la memoria en tres partes, las cuales se encargaran de cada función. Tanto entrada como el alojamiento del programa solo son de lectura, lo que significa que NO se pueden modificar, y en un algoritmo cuando se menciona los requerimientos de este, es referencia a la cantidad de memoria que utiliza.

Generalmente no existe un límite de memoria disponible, así que podemos tener la cantidad de memoria que queramos pero se dividirá en celdas únicas con una capacidad limitada. No pueden almacenar valores tan altos, pero habrá suficientes celdas de memoria como el programa necesite.

Si un programa se es ejecutado en la RAM, lo que nos importa es el tiempo que toma para realizar una entrada, hay 3 reglas que determinan que tanto la RAM ejecuta un programa:

- 1) Operaciones simples: Tales como sumas, multiplicaciones, condicionales tipo if o for, toman 1 solo proceso.
- 2) Si tu programa tiene algún bucle, se tomara no solo como una operación si no se toman en cuenta las veces que es ejecutado; por ejemplo si tienes un bucle con una operación simple que se realiza 100 veces, serán 100 procesos.
- 3) El acceso a la memoria es libre, si se lee o se escribe algo en alguna celda de la memoria, es posible; Se trata de un proceso inmediato.

Con esto podemos llegar a determinar el tiempo de ejecución de un algoritmo, programa o comparar tiempos de ejecución en dos o más programas. Solo es contar el número de procesos dados a la RAM para que realice una entrada/acción.



INTRODUCCIÓN

La teoría de autómatas es una rama de la teoría de la computación que estudia las máquinas abstractas y los problemas que éstas son capaces de resolver. La teoría de autómatas está estrechamente relacionada con la teoría del lenguaje formal ya que los autómatas son clasificados a menudo por la clase de lenguajes formales que son capaces de reconocer. Es un modelo matemático para una máquina de estado finito (FSM en inglés).

Como parte de nuestro proyecto para la clase de Lenguajes y Autómatas I, se nos planteó el analizar y realizar un modelo teórico de una Máquina abstracta, siendo más específicos la Máquina de Acceso Aleatorio (RAM en inglés), la cual debería incluir un trabajo escrito que contenga la investigación realizada por cada integrante, así como sus ideas y explicación de uso, refutado con un marco teórico y un modelo de trabajo igualmente seleccionado por el equipo donde está previsto la forma de organización y planeación por el mismo, finalizando con un desarrollo utilizando diferentes recursos gráficos, visuales y estructurales.



OBJETIVOS

Objetivos General

Desarrollar un software para comprobar la maquina de acceso aleatorio secuencial Insertando Registros en la memoria

Objetivos Específicos

Realizar un menú de opciones y registros en la IDE NetBeans para que el usuario ingrese los datos

Realizar un software agradable e intuitivo para el fácil entendimiento del usuario

Diseñar el algoritmo para el funcionamiento del software

Documentar la investigación de la máquina de acceso aleatorio secuencial



JUSTIFICACIÓN

Para la justificación diremos que para tener un RANDOM-ACCESS MACHINE se tiene que tener en cuenta la memoria RAM, y su buen uso de dicha función.

También se tomará en cuenta del cómo se hace la estructura de dicho, la escritura y su lectura del cómo tiene el orden de hacerlo

El modelo de cálculo RAM (Random Access Machine) mide el tiempo de ejecución de un algoritmo sumando el número de pasos necesarios para ejecutar el algoritmo en un conjunto de datos. El modelo RAM opera según los siguientes principios:

- Los bucles y subrutinas son operaciones complejas compuestas por múltiples pasos de tiempo.
- Todo el acceso a la memoria toma exactamente un paso de tiempo.

Este modelo encapsula la funcionalidad principal de las computadoras, pero no las imita por completo. Por ejemplo, una operación de suma y una operación de multiplicación valen un solo paso de tiempo, sin embargo, en realidad, una máquina necesitará más operaciones para calcular un producto en comparación con una suma.

La razón por la que el modelo RAM hace estas suposiciones es porque al hacerlo permite un equilibrio entre la simplicidad y la imitación completa de la máquina subyacente, lo que resulta en una herramienta que es útil en la práctica.

El análisis exacto de algoritmos es una tarea difícil. La naturaleza del análisis de algoritmos es ser independiente de la máquina y del lenguaje. Por ejemplo, si su computadora se vuelve dos veces más rápida después de una actualización reciente, la complejidad de su algoritmo seguirá siendo la misma.

MARCO TEÓRICO (Conceptos y fundamentos)

Teoría de la computación

Es un conjunto de conocimientos racionales, sistematizados, y funcionales, que se centran en el estudio de la abstracción de los procesos que ocurren en la realidad con el fin de reproducirlos con ayuda de sistemas formales, es decir, a través de códigos de caracteres e instrucciones lógicas, reconocibles por el ser humano, con capacidad de ser modeladas en las limitaciones de dispositivos que procesan información y efectúan cálculos, tales como el ordenador. Para ello se apoya en la teoría de autómatas para simular y estandarizar dichos procesos, así como para formalizar los problemas y darles solución.

Principales subramas

Teoría de autómatas

Artículo principal: Teoría de autómatas

Esta teoría provee modelos matemáticos que formalizan el concepto de computadora o algoritmo de manera suficientemente simplificada y general para que se puedan analizar sus capacidades y limitaciones. Algunos de estos modelos juegan un papel central en varias aplicaciones de las ciencias de la computación, incluyendo procesamiento de texto, compiladores, diseño de hardware e inteligencia artificial.

Existen muchos otros tipos de autómatas como las máquinas de acceso aleatorio, autómatas celulares, máquinas ábaco y las máquinas de estado abstracto; sin embargo en todos los casos se ha mostrado que estos modelos no son más generales que la máquina de Turing, pues la máquina de Turing tiene la capacidad de simular cada uno de estos autómatas. Esto da lugar a que se piense en la máquina de Turing como el modelo universal de computadora.

Teoría de la complejidad computacional

Artículo principal: Complejidad computacional

Aun cuando un problema sea computable, puede que no sea posible resolverlo en la práctica si se requiere mucha memoria o tiempo de ejecución. La teoría de la complejidad computacional estudia las necesidades de memoria, tiempo y otros recursos computacionales para resolver problemas; de esta manera es posible explicar por qué unos problemas son más difíciles de resolver que otros. Uno de los mayores logros de esta rama es la clasificación de problemas, similar a la tabla periódica, de acuerdo a su dificultad. En esta clasificación los problemas se separan por clases de complejidad.

Esta teoría tiene aplicación en casi todas las áreas de conocimiento donde se desee resolver un problema computacionalmente, porque los investigadores no solo desean utilizar un método para resolver un problema, sino utilizar el más rápido. La teoría de la complejidad computacional también tiene aplicaciones en áreas como la criptografía, donde se espera que descifrar un código secreto

sea un problema muy difícil a menos que se tenga la contraseña, en cuyo caso el problema se vuelve fácil.

Teoría de la computabilidad

Esta teoría explora los límites de la posibilidad de solucionar problemas mediante algoritmos. Gran parte de las ciencias computacionales están dedicadas a resolver problemas de forma algorítmica, de manera que el descubrimiento de problemas *imposibles* es una gran sorpresa. La teoría de la computabilidad es útil para no tratar de resolver algorítmicamente estos problemas, ahorrando así tiempo y esfuerzo.

Los problemas se clasifican en esta teoría de acuerdo a su grado de *imposibilidad*:

Los computables son aquellos para los cuales sí existe un algoritmo que siempre los resuelve cuando hay una solución y además es capaz de distinguir los casos que no la tienen. También se les conoce como *decidibles*, *resolubles* o *recursivos*.

Los semicomputables son aquellos para los cuales hay un algoritmo que es capaz encontrar una solución si es que existe, pero ningún algoritmo que determine cuando la solución no existe (en cuyo caso el algoritmo para encontrar la solución entraría a un bucle infinito). El ejemplo clásico por excelencia es el problema de la parada. A estos problemas también se les conoce como *listables*, *recursivamente enumerables* o *reconocibles*, porque si se enlistan todos los casos posibles del problema, es posible *reconocer* a aquellos que sí tienen solución.

Los incomputables son aquellos para los cuales no hay ningún algoritmo que los pueda resolver, no importando que tengan o no solución. El ejemplo clásico por excelencia es el problema de la implicación lógica, que consiste en determinar cuándo una proposición lógica es un teorema; para este problema no hay ningún algoritmo que en todos los casos pueda distinguir si una proposición o su negación es un teorema.

Hay una versión más general de esta clasificación, donde los problemas incomputables se subdividen a su vez en problemas más difíciles que otros. La herramienta principal para lograr estas clasificaciones es el concepto de reducibilidad: Un problema A se reduce al problema B si bajo la suposición de que se sabe resolver el problema B es posible resolver al problema A ; esto se denota por $A \leq_t B$, e informalmente significa que el problema A *no es más difícil de resolver* que el problema B . Por ejemplo, bajo la suposición de que una persona sabe sumar, es muy fácil enseñarle a multiplicar haciendo sumas repetidas, de manera que multiplicar se reduce a sumar.

La Teoría de la Computación estudia modelos abstractos de los dispositivos concretos que conocemos como computadores, y analiza lo que se puede y no se puede hacer con ellos. Este estudio teórico se inició varias décadas antes de la aparición de los primeros computadores reales y continúa creciendo, a medida que la computación incrementa su sofisticación.

Entre los muchos tópicos que conforman la teoría de la computación, sólo tendremos la oportunidad de tratar someramente los dos siguientes:

Modelos de computación. Las investigaciones en este campo comenzaron en la década de los 30 del siglo XX con el trabajo del lógico norteamericano Alonzo Church (1903-1995) y del matemático británico Alan Turing

(1912-1954). Church introdujo el formalismo conocido como cálculo λ -A y enunció la tesis -hoy conocida como tesis de Church- de que las funciones efectivamente computables, es decir, computables por cualquier método computacional concebible, son exactamente las funciones λ -computables.

En contraste con el enfoque más abstracto de Church, Turing (quien fue alumno doctoral de Church en la universidad de Princeton) propuso un modelo concreto de máquina computadora, hoy conocida como la máquina de Turing, capaz de simular las acciones de cualquier otro dispositivo físico de computación secuencial.

Curiosamente, las propuestas de Church y Turing se publicaron exactamente en el mismo año: 1936. Los dos formalismos resultaron ser equivalentes y, desde entonces, se han propuesto muchos otros modelos de computación. Como todos han resultado ser equivalentes entre sí, ha ganado

3.4 INTRODUCCIÓN

aceptación universal la tesis de Church-Turing: no hay modelo de computación más general ni poderoso que la máquina de Turing.

En la teoría de la computabilidad y en la teoría de la complejidad computacional, un modelo de computación es la definición un conjunto de operaciones permitibles usadas en el cómputo y sus respectivos costos. Solo asumiendo un cierto modelo de computación es posible analizar los recursos de cómputo requeridos, como el tiempo de ejecución o el espacio de memoria, o discutir las limitaciones de algoritmos o computadores.

Algunos ejemplos de modelos incluyen las máquinas de Turing, las funciones recursivas, cálculo λ , y sistema de producción.

En la ingeniería dirigida por modelos, el modelo de computación explica cómo el comportamiento del sistema entero es el resultado del comportamiento de cada uno de sus componentes.

En el campo del tiempo de ejecución del análisis de algoritmos, es común especificar un modelo computacional en términos de operaciones primitivas permitidas que tengan un costo unitario, o simplemente operaciones costo unitario. Un ejemplo comúnmente usado es la máquina de acceso aleatorio, que tiene costo unitario para acceso de lectura y escritura para todas sus celdas de memoria. En este respecto, se diferencia del modelo de máquina de Turing mencionado arriba.

Random Access Machine (sequential) (RAM-SECUENCIAL)

En lógica matemática y en ciencias de la computación teórica, una máquina de registro es una clase genérica de máquinas abstractas usadas en una manera similar a una máquina de Turing. Toma su nombre por sus uno o más "registros", en lugar de la cinta y el cabezal de una máquina de Turing (o cintas y cabezales) el modelo usa múltiples registros con dirección única, cada uno de los cuales mantiene un simple número entero positivo.

Hay por lo menos 4 subclases encontradas en la literatura, aquí son enumeradas desde la más primitiva a la más avanzada como computadora:

- **Máquina contadora:** El más primitivo y más reducido modelo. Carece de direccionamiento indirecto. Las instrucciones están en la máquina de estado finito en la manera de la arquitectura Harvard.
- **Máquina de puntero:** Una mezcla de la máquina contadora y los modelos de máquina de acceso aleatorio. Menos común y más abstracta que cualquiera de estos modelos. Las instrucciones están en la máquina de estado finito de la manera de la arquitectura Harvard.
- **Máquina de acceso aleatorio (RAM):** Una máquina contadora con direccionamiento indirecto y, usualmente, un conjunto de instrucciones aumentado. Las instrucciones están en la máquina de estado finito a la manera de la arquitectura Harvard.
- **Máquina de acceso aleatorio con programa almacenado (RASP):** Una máquina de acceso aleatorio con instrucciones en sus registros análogos a la máquina universal de Turing; así que es un ejemplo de la arquitectura de von Neumann. Pero a diferencia de una computadora, el modelo idealizado con efectivamente infinitos registros (y si es usada, efectivamente infinitos registros especiales tales como el acumulador). A diferencia de una computadora o aún de un procesador RISC, el conjunto de instrucciones es muy reducido en el número de instrucciones.

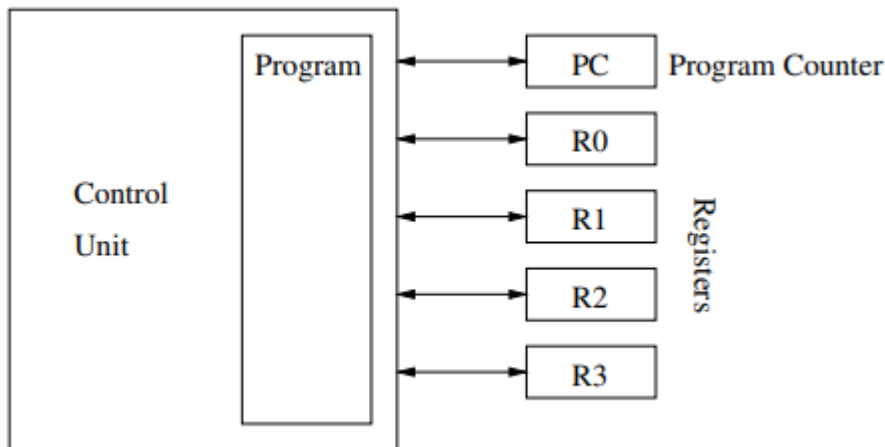


Ilustración 1- Representación gráfica básica de una RAM)

La máquina de acceso aleatorio (RAM) consiste de una memoria con M posiciones que puede ser ilimitado. Para nuestros propósitos, M es un número arbitrariamente grande, pero finito. Cada posición de memoria almacena un dato al que puede accederse aleatoriamente usando una dirección única.

Un procesador operando bajo el control de un algoritmo secuencial. El procesador es capaz de cargar y almacenar datos desde y en la memoria, y ejecutar operaciones aritméticas y lógicas básicas.

Posee un número constante de registros internos (o locales) para realizar cálculos sobre los datos.

Una Unidad de Acceso a la Memoria (MAU, Memory Access Unit) cuyo propósito es crear un camino desde el procesador a una posición arbitraria de la memoria. Cada vez que el procesador desea leer de o escribir a una posición de memoria, proporciona a la MAU la dirección de dicha posición. Usando esta dirección, se establece una conexión directa entre el procesador y la posición de memoria.

Cada paso del algoritmo consiste de (hasta) tres fases:

1. Una fase de LECTURA, durante la cual el procesador lee un dato desde una posición arbitraria de la memoria en uno de sus registros internos,
2. Una fase de CÓMPUTO, durante la cual el procesador realiza una operación básica sobre los contenidos de uno o dos de sus registros.

3. Una fase de ESCRITURA, durante la cual el procesador escribe el contenido de un registro en una posición de memoria arbitraria.

Se puede entender que un Modelo de Acceso Aleatorio puede asemejarse a una memoria de registro tipo CPU usados en las computadoras modernas.

Explicación Grafica (RAM Model - Georgia Tech - Computability, Complexity, Theory: Computability)

En vez de operar como un alfabeto finito como una máquina de Turing, el modelo RAM funciona con enteros no negativos, los cuales pueden ser arbitrariamente largos. Cuenta con registros que pueden ser útiles para almacenar operandos de operaciones básicas y un almacenaje infinito en comparación a las cintas de una máquina de Turing regular.

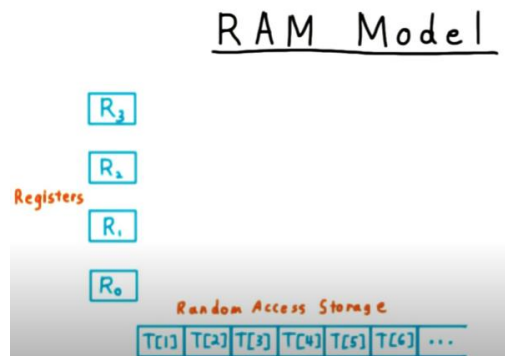


Ilustración 2

Existen dos diferencias clave entre este tipo de memoria y las de la cinta de Turing, una es que cada posición se almacena un número, y la otra es que cualquier elemento puede ser leído con una simple instrucción en vez de tener que posicionar o mover la cinta a la posición deseada.

Aparte del almacenamiento, la máquina también contiene el programa mismo, expresado en una secuencia de instrucciones y un registro especial denominado el contador del programa, que se encarga de revisar cual instrucción es la que se debe ejecutar.

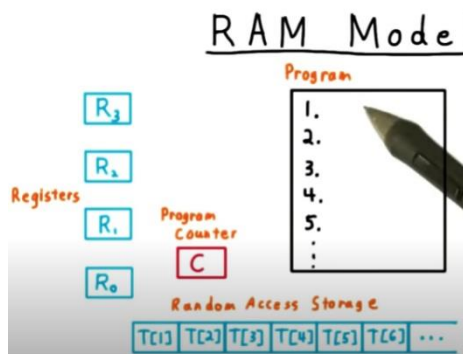


Ilustración 3

Cada instrucción es una de un valor finito que se asemeja a las instrucciones del código ensamblador. Ejemplo la instrucción *read j*, que lee el contenido de la dirección J en la memoria y la coloca en el registro 0.

Registro 0 tiene un estado especial, y está relacionado con casi todo tipo de operación.

También está la operación *write j*, que escribe en la dicha dirección desde el registro 0.

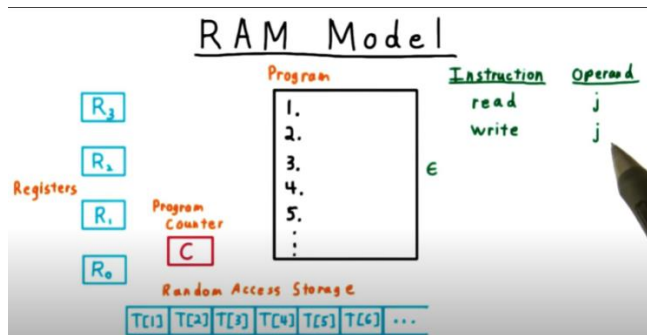


Ilustración 4

Si queremos mover información entre nuestro registros, tenemos la función *load* que escribe en el registro 0 y lo almacena en el registro que deseemos, de igual forma incrementa el registro 0 con la cantidad de R_j (j siendo un valor cualesquiera).

Todo este tipo de operaciones hacen que el contador del programa aumente por 1 después de finalizar.

Para desplazarse por la lista en el programa, existen una serie especial de instrucciones *jump* que cambian el contador del programa. A veces depende del valor en el registro 0, y finalmente tenemos la instrucción *halt* para detener el programa.

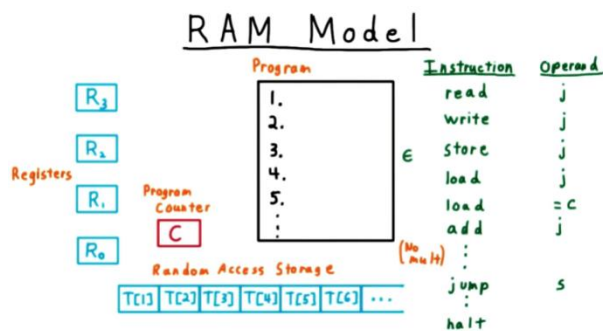


Ilustración 5

El valor final en el registro 0 determina si se acepta o rechaza el valor, cabe mencionar que en esta definición no existe multiplicación, pero se puede lograr siguiendo una serie de sumas.

Una máquina de Turing con acceso aleatorio consiste en:

- Un número cualquiera que indica número de registros
- La secuencia de instrucciones $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ donde cada π es una instrucción.

La configuración de una RAM consiste en:

- El valor del contador del programa (iniciando en 0)
- Los valores de los registros $R_0, \dots, R_{k-1}, \in N_0$
- Los valores en memoria o RAM que se pueden expresar como una función $T = N_1 \rightarrow N_0$

METODOLOGÍA DE TRABAJO

La metodología de trabajo es una herramienta muy potente para definir la pautas y procedimientos, conjunto de procedimientos racionales utilizados para alcanzar el objetivo o la gama de objetivos que rige una investigación científica, una exposición doctrinal o tareas que requieran habilidades, conocimientos o cuidados específicos. Con frecuencia puede definirse la metodología como el estudio o elección de un método pertinente o adecuadamente aplicable a determinado objeto.

Metodología Scrum

La metodología Scrum es un marco de trabajo o framework que se utiliza dentro de equipos que manejan proyectos complejos. Es decir, se trata de una metodología de trabajo ágil que tiene como finalidad la entrega de valor en períodos cortos de tiempo y para ello se basa en tres pilares: la transparencia, inspección y adaptación.

1. Transparencia

Con el método Scrum todos los implicados tienen conocimiento de qué ocurre en el proyecto y cómo ocurre. Esto hace que haya un entendimiento “común” del proyecto, una visión global.

2. Inspección

Los miembros del equipo Scrum frecuentemente inspeccionan el progreso para detectar posibles problemas. La inspección no es un examen diario, sino una forma de saber que el trabajo fluye y que el equipo funciona de manera auto-organizada.

3. Adaptación

Cuando hay algo que cambiar, el equipo se ajusta para conseguir el objetivo del sprint. Esta es la clave para conseguir el éxito en proyectos complejos, donde los requisitos son cambiantes o poco definidos y en donde la adaptación, la innovación, la complejidad y flexibilidad son fundamentales.

Al estar enmarcada dentro de las metodologías agile, Scrum se basa en aspectos como:

La flexibilidad en la adopción de cambios y nuevos requisitos durante un proyecto complejo.

- El factor humano.
- La colaboración e interacción con el cliente.
- El desarrollo iterativo como forma de asegurar buenos resultados.

La manera en que funciona la metodología Scrum es muy sencilla: una vez que se han creado los equipos se reparten el trabajo en una lista de pequeños entregables con un orden de prioridad; los tiempos de entrega se dividen en ciclos conocidos como sprints, que por lo regular representan una semana.

FASES DE LA METODOLOGÍA SCRUM

El desarrollo de producto tiene un ciclo de vida en la metodología Scrum. Estas son fases en las que se divide un proceso Scrum:

¿Qué y quién? El producto que queremos conseguir una vez terminemos el Sprint, y los roles de equipo con sus tareas asignadas.

¿Dónde y cuándo? El plazo y el contenido del Sprint.

¿Por qué y cómo? Las distintas herramientas para aplicar esta metodología ágil.

Todos los integrantes colaboran en función de sus conocimientos individuales y el trabajo se optimiza a través de diferentes reuniones que se tienen al término de cada sprint.

Ventajas de la metodología Scrum

- Scrum es muy fácil de aprender: los roles, hitos y herramientas son claros y tienen un objetivo por lo que es un método muy relacionado con nuestra manera diaria de trabajar.
- Se agiliza el proceso, ya que la entrega de valor es muy frecuente

Desarrollo e Implementación

Para acceder de forma aleatoria a los datos contenidos en el fichero, la clase `RandomAccessFile` dispone de varios métodos. Entre ellos:

long getFilePointer();

Devuelve la posición actual del puntero del fichero. Indica la posición (en bytes) donde se va a leer o escribir.

long length();

Devuelve la longitud del fichero en bytes.

void seek(long pos);

Coloca el puntero del fichero en una posición *pos* determinada. La posición se da como un desplazamiento en bytes desde el comienzo del fichero. La posición 0 indica el principio del fichero. La posición *length()* indica el final del fichero.

Además dispone de métodos de lectura/escritura:

public int read();

Devuelve el byte leído en la posición marcada por el puntero. Devuelve -1 si alcanza el final del fichero. Se debe utilizar este método para leer los caracteres de un fichero de texto.

public final String readLine();

Devuelve la cadena de caracteres que se lee, desde la posición marcada por el puntero, hasta el siguiente salto de línea que se encuentre.

public xxx readXxx();

Hay un método `read` para cada tipo de dato básico: **`readChar`**, **`readInt`**, **`readDouble`**, **`readBoolean`**, etc.

public void write(int b);

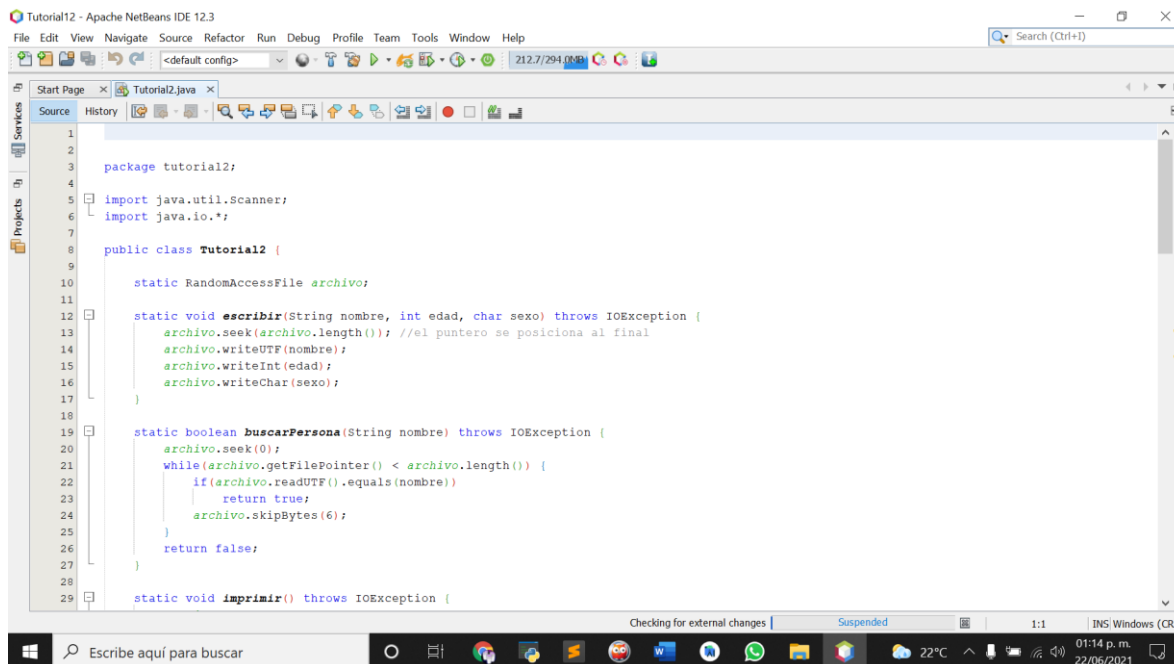
Escribe en el fichero el byte indicado por parámetro. Se debe utilizar este método para escribir caracteres en un fichero de texto.

public final void writeBytes(String s);

Escribe en el fichero la cadena de caracteres indicada por parámetro.

public final void writeXxx(argumento);

También existe un método `write` para cada tipo de dato básico: **`writeChar`**, **`writeInt`**, **`writeDouble`**, **`writeBoolean`**, etc.

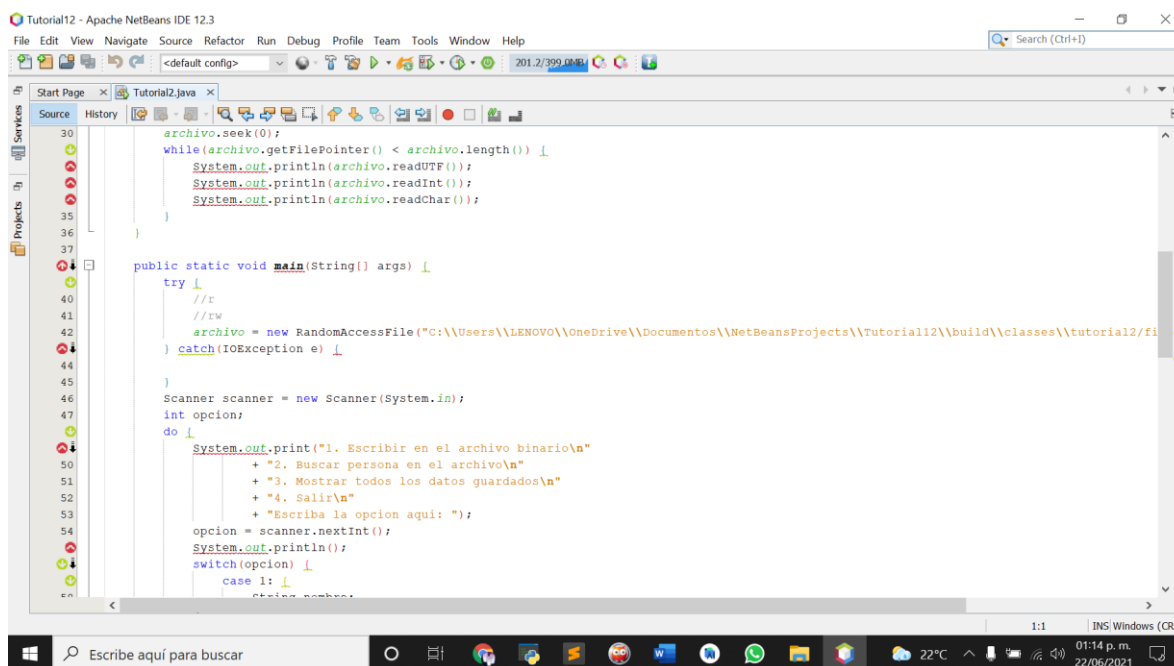


```

1 package tutorial2;
2
3
4
5 import java.util.Scanner;
6 import java.io.*;
7
8 public class Tutorial2 {
9
10     static RandomAccessFile archivo;
11
12     static void escribir(String nombre, int edad, char sexo) throws IOException {
13         archivo.seek(archivo.length()); //el puntero se posiciona al final
14         archivo.writeUTF(nombre);
15         archivo.writeInt(edad);
16         archivo.writeChar(sexo);
17     }
18
19     static boolean buscarPersona(String nombre) throws IOException {
20         archivo.seek(0);
21         while(archivo.getFilePointer() < archivo.length()) {
22             if(archivo.readUTF().equals(nombre)) {
23                 return true;
24             }
25             archivo.skipBytes(6);
26         }
27         return false;
28     }
29
30     static void imprimir() throws IOException {

```

Ilustración 6 Desarrollo de código

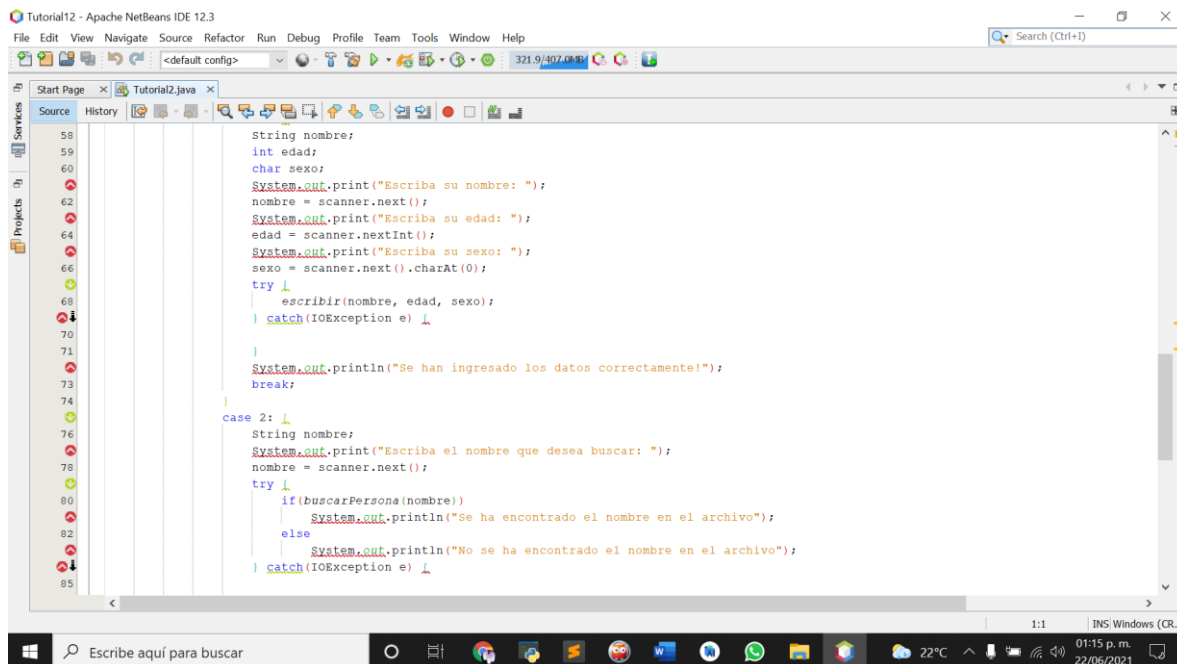


```

30     archivo.seek(0);
31     while(archivo.getFilePointer() < archivo.length()) {
32         System.out.println(archivo.readUTF());
33         System.out.println(archivo.readInt());
34         System.out.println(archivo.readChar());
35     }
36
37
38
39
40     public static void main(String[] args) {
41         try {
42             //r
43             //rw
44             archivo = new RandomAccessFile("C:\\Users\\LENOVO\\OneDrive\\Documents\\NetBeansProjects\\Tutorial12\\build\\classes\\tutorial2\\fi
45         } catch (IOException e) {
46
47         }
48         Scanner scanner = new Scanner(System.in);
49         int opcion;
50         do {
51             System.out.print("1. Escribir en el archivo binario\n"
52                 + "2. Buscar persona en el archivo\n"
53                 + "3. Mostrar todos los datos guardados\n"
54                 + "4. Salir\n"
55                 + "Escriba la opcion aqui: ");
56             opcion = scanner.nextInt();
57             System.out.println();
58             switch(opcion) {
59                 case 1:
60                     escribir(nombre, edad, sexo);
61                 case 2:
62                     buscarPersona(nombre);
63                 case 3:
64                     imprimir();
65                 case 4:
66                     return;
67             }
68         } while(opcion != 4);
69     }

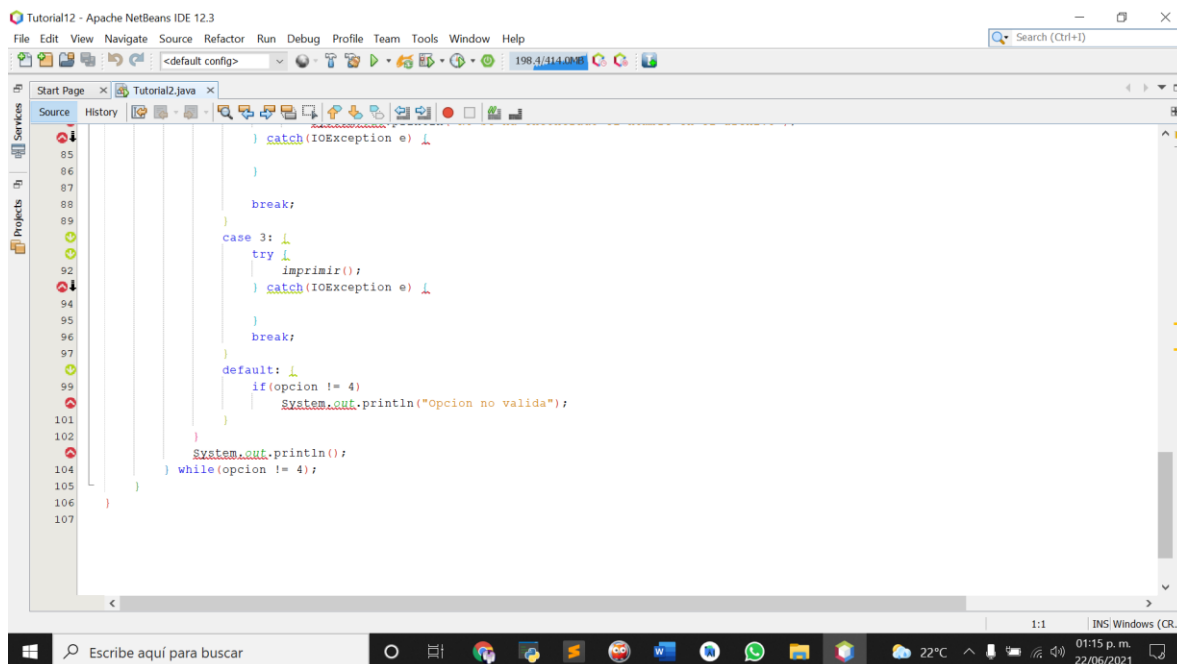
```

Ilustración 7 Desarrollo de código



```
58 String nombre;
59 int edad;
60 char sexo;
61 System.out.print("Escriba su nombre: ");
62 nombre = scanner.next();
63 System.out.print("Escriba su edad: ");
64 edad = scanner.nextInt();
65 System.out.print("Escriba su sexo: ");
66 sexo = scanner.next().charAt(0);
67 try {
68     escribir(nombre, edad, sexo);
69 } catch (IOException e) {
70 }
71
72 System.out.println("Se han ingresado los datos correctamente!");
73 break;
74
75 case 2: {
76     String nombre;
77     System.out.print("Escriba el nombre que desea buscar: ");
78     nombre = scanner.next();
79     try {
80         if (buscarPersona(nombre))
81             System.out.println("Se ha encontrado el nombre en el archivo");
82         else
83             System.out.println("No se ha encontrado el nombre en el archivo");
84     } catch (IOException e) {
85     }
```

Ilustración 8 Desarrollo de código



```
85 } catch (IOException e) {
86 }
87
88 break;
89
90 case 3: {
91     try {
92         imprimir();
93     } catch (IOException e) {
94     }
95     break;
96 }
97
98 default: {
99     if (opcion != 4)
100         System.out.println("Opcion no valida");
101 }
102
103 System.out.println();
104 } while (opcion != 4);
105
106 }
107 }
```

Ilustración 9 Desarrollo de código

Resultados

Aquí se guarda en un archivo de extensión .DAT

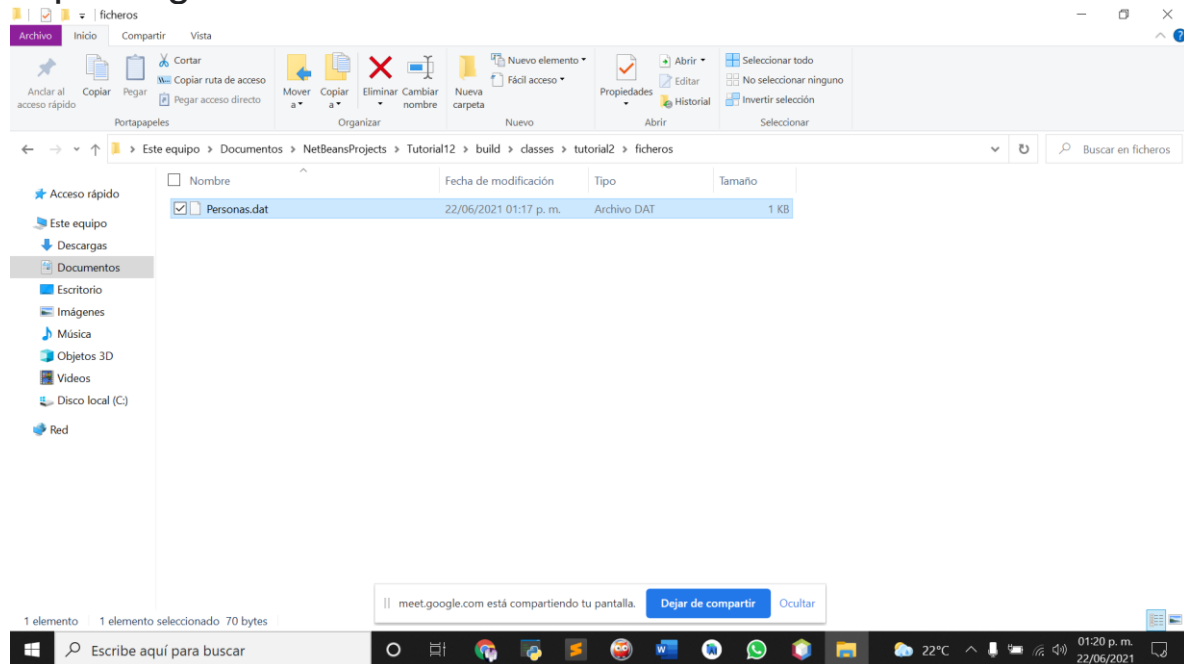


Ilustración 10

Buscamos una persona en el archivo DAT

Mostramos todos los datos guardados

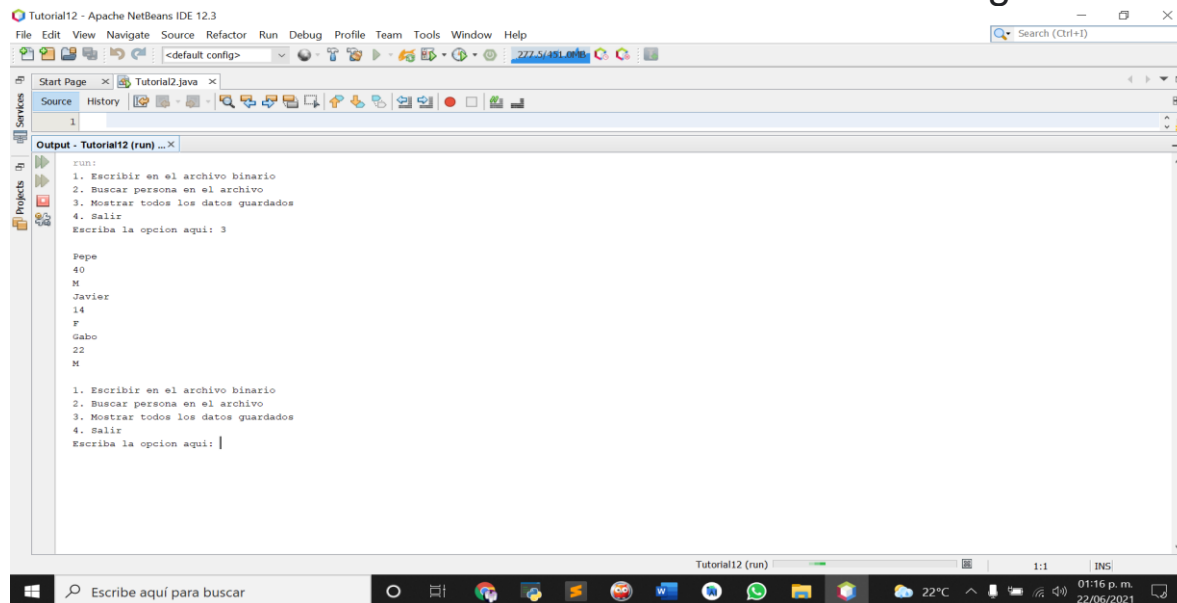


Ilustración 11

Vemos el archivo guardado “Se ha seleccionado el nombre en el archivo”

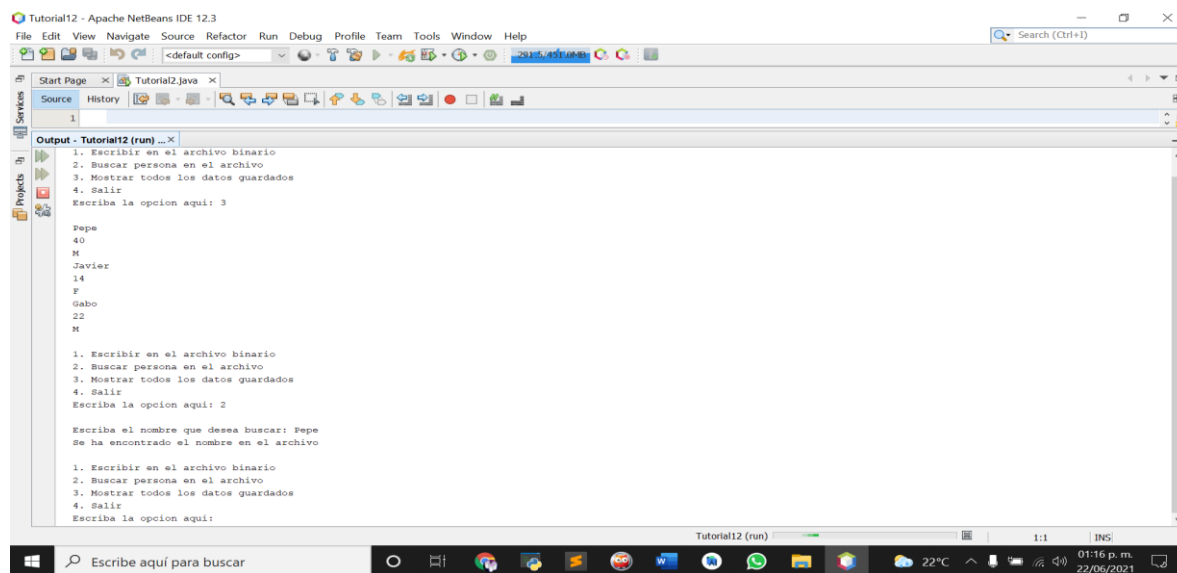
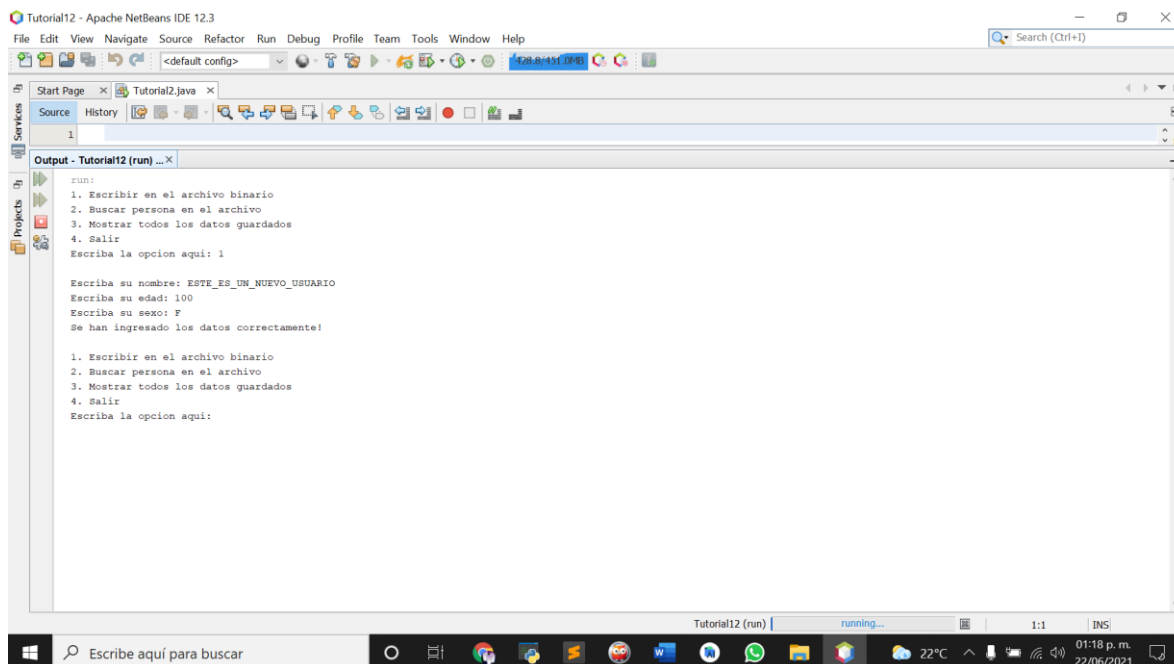


Ilustración 12

Agregamos un nuevo nombre
“ESTE_ES_UN_NUEVO_USUARIO” y lo guardamos en el
archivo de memoria DAT



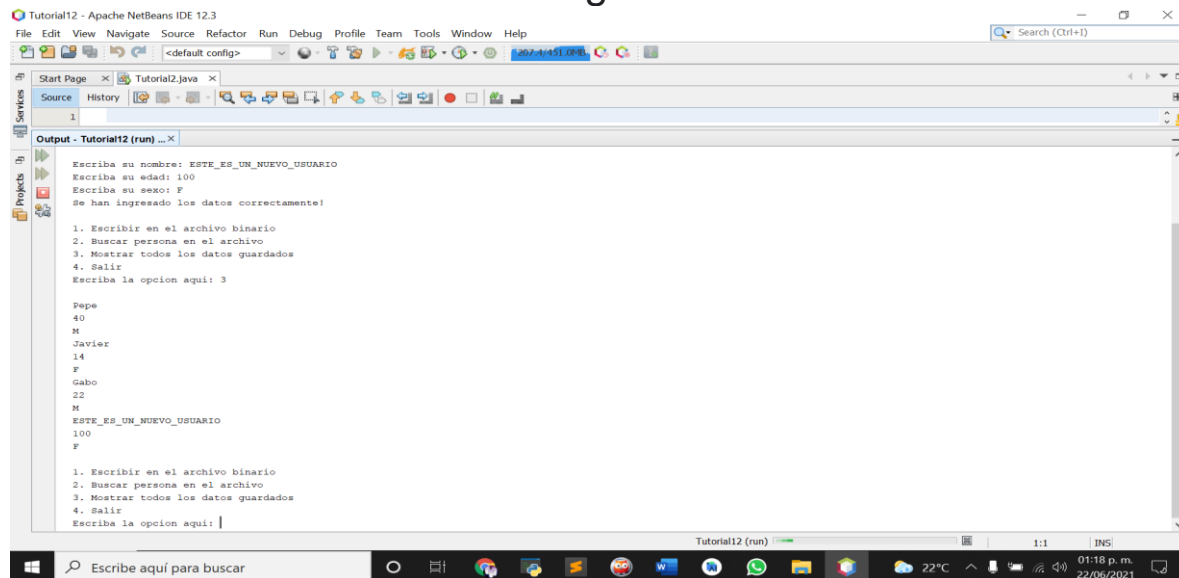
```
run:
1. Escribir en el archivo binario
2. Buscar persona en el archivo
3. Mostrar todos los datos guardados
4. Salir
Escriba la opción aquí: 1

Escriba su nombre: ESTE_ES_UN_NUEVO_USUARIO
Escriba su edad: 100
Escriba su sexo: F
Se han ingresado los datos correctamente!

1. Escribir en el archivo binario
2. Buscar persona en el archivo
3. Mostrar todos los datos guardados
4. Salir
Escriba la opción aquí:
```

Ilustración 13

Mostramos todos los datos guardados en el archivo DAT



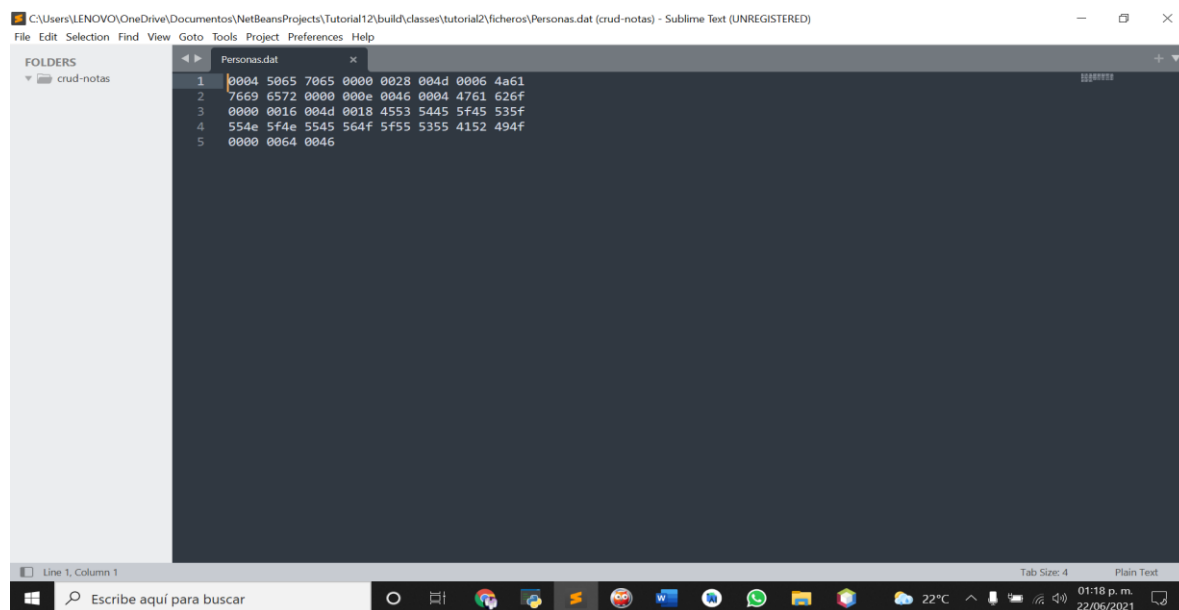
```
Output - Tutorial12 (run) ...x
Escriba su nombre: ESTE_ES_UN_NUEVO_USUARIO
Escriba su edad: 100
Escriba su sexo: F
Se han ingresado los datos correctamente!

1. Escribir en el archivo binario
2. Buscar persona en el archivo
3. Mostrar todos los datos guardados
4. Salir
Escriba la opcion aqui: 3

Pepe
40
M
Javier
14
F
Gabo
22
M
ESTE_ES_UN_NUEVO_USUARIO
100
F

1. Escribir en el archivo binario
2. Buscar persona en el archivo
3. Mostrar todos los datos guardados
4. Salir
Escriba la opcion aqui: |
```

Ilustración 14



```
C:\Users\LENOVO\OneDrive\Documentos\NetBeansProjects\Tutorial12\build\classes\tutorial2\ ficheros\Personas.dat (crud-notas) - Sublime Text (UNREGISTERED)
FOLDERS
  crud-notas
Personas.dat
1 0004 5065 7065 0000 0028 004d 0006 4a61
2 7669 6572 0000 000e 0046 0004 4761 626f
3 0000 0016 004d 0018 4553 5445 5f45 535f
4 554e 5f4e 5545 564f 5f55 5355 4152 494f
5 0000 0064 0046
```

Ilustración 15

Abrimos el archivo DAT y rectificamos que se guardan los datos en el archivo de memoria DAT



CONCLUSION:

Para concluir tenemos que un modelo computacional es un modelo matemático en las ciencias de la computación que requiere extensos recursos computacionales para estudiar el comportamiento de un sistema complejo por medio de la simulación por computadora y que la máquina de acceso aleatorio (RAM) es una máquina abstracta en la clase general de máquinas de registro.

La RAM es muy similar a la máquina contadora, pero con la capacidad adicional de "direccionamiento indirecto" de sus registros. Al igual que la máquina de contador, la RAM tiene sus instrucciones en la parte de estado finito de la máquina. El equivalente de RAM de la máquina universal de Turing, con su programa en los registros y sus datos, se denomina máquina de programa almacenado de acceso aleatorio o RASP.

Nuestro modelo de cómputo secuencial es la máquina de acceso aleatorio (RAM, Random Access Machine) y que su funcionamiento se compone por:

1. Una fase de lectura, durante la cuál el procesador lee un dato desde una posición arbitraria de la memoria en uno de sus registros internos
2. Una fase de cómputo, durante la cuál el procesador realiza una operación básica sobre los contenidos de uno o dos de sus registros
3. Una fase de escritura, durante la cual el procesador escribe el contenido de un registro en una posición de memoria arbitraria.



Fuentes de información

1. Volver arriba↑ Nachum Dershowitz & Yuri Gurevich (2008). «A natural axiomatization of computability and proof of Church's Thesis». *Bulletin of Symbolic Logic* 14 (3). ISSN 10798986, 299-350.

- Sipser, Michael (2005). *Introduction to the Theory of Computation* (2 edición). Course Technology. ISBN 978-0534950972.

- Kelley, Dean (1995). *Teoría de Autómatas y Lenguajes Formales*. Prentice Hall. ISBN 978-0-691-13382-9.

- Boolos, George; Burgess, John; & Jeffrey, Richard (2007). *Computability and logic*. Cambridge. ISBN 978-0-521-70146-4.

- S. Barry Cooper (2004). *Computability theory*. Chapman & Hall/CRC. ISBN 1-58488-237-9.

- Sección 68Qxx, *Theory of computing* de American Mathematical Society. «2010 Mathematics Subject Classification.». Consultado el 7 de marzo de 2010.

Calvin Elgot y Abraham Robinson (1964), Máquinas de programa almacenado de acceso aleatorio, una aproximación a los lenguajes de programación , *Revista de la Asociación de Maquinaria de Computación*, vol. 11, núm. 4 (octubre de 1964), págs. 365–399. J. Hartmanis (1971), "Complejidad computacional de las máquinas de programas almacenados de acceso aleatorio", *Teoría de sistemas matemáticos* 5, 3 (1971) págs. 232–245. Máquina de acceso aleatorio - https://es.xcv.wiki/wiki/Random-access_machine#Footnotes