**Benemérita Universidad Autónoma de Puebla**



**EJERCICIO 9:**

**“Archivo”**

**Fac. de Cs. Computación**

**Asignatura:**

**Programación II**

**Profesora: Carmen Cerón Garnica**

**Equipo 4:**

**Emmanuel Flores Navarro 201929142**

**Erick Brian Ramírez Morales 201955520**

**07 / Mayo / 2021**

**INTRODUCCIÓN**

**Archivos y flujos**

El manejo de archivos en Java se hace mediante el concepto de flujo (*stream*) o canal, también denominado secuencia; éste conduce los datos entre el programa y los dispositivos externos, además puede estar abierto o cerrado. Con las clases y sus métodos proporcionadas por el paquete de clases de entrada y salida (*java.io*) se pueden tratar archivos secuenciales, de acceso directo, archivos indexados, etcétera.

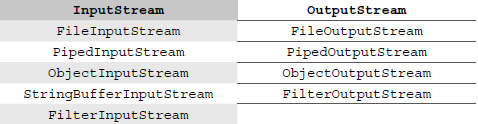
Los archivos tienen como finalidad guardar datos de forma permanente; una vez terminada la aplicación los datos almacenados quedan disponibles para que otra pueda recuperarlos, así como para consulta o modificación. Los archivos son dispositivos externos, en contraposición de los datos que se guardan en arreglos, listas, o árboles que están en memoria interna y, por tanto, desaparecen al acabar la ejecución del programa.

* **Archivo**

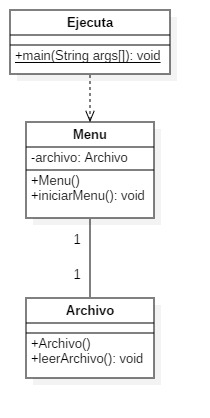
Un fichero, archivo de datos o simplemente archivo es una colección de registros relacionados entre sí con aspectos comunes y organizados para un propósito específico. Un archivo es una estructura diseñada para contener datos que están organizados de tal modo que puedan recuperarse fácilmente, borrarse, actualizarse o almacenarse de nuevo en el archivo con todos los cambios realizados.

* **Flujos y jerarquía de clases**

Todo el proceso de entrada y salida en Java se hace a través de flujos; en los programas hay que crear objetos *stream* y en muchas ocasiones hacer uso de los objetos *in*, *out* de la clase *System*; los flujos de datos, caracteres o bytes pueden ser de entrada y salida. En consonancia, Java declara dos clases que derivan directamente de la clase *Object*: *InputStream* y *OutputStream*; ambas son abstractas y declaran métodos que deben redefinirse en sus clases derivadas; la primera es la base de todas las clases definidas para flujos de entrada; la segunda es la base de todas las clases definidas para flujos de salida.



**DIAGRAMA UML**



**CÓDIGO DEL PROGRAMA**

**Archivo.java**

// Fecha: 07 / Mayo / 2021

// Objetivo: Mostrar la extension de un archivo.

// Elaboro Equipo 4:

// 1) Flores Navarro Emmanuel

// 2) Ramírez Morales Erick Brian

//Inicio de la clase

import java.util.GregorianCalendar;

import javax.swing.JOptionPane;

import java.util.Calendar;

import java.util.Date;

import java.io.\*;

public class Archivo {

//Método constructor

public Archivo() {}

//Metódo definido por el programador

public void leerArchivo() {

//Variables de instancia

File f = new File("Archivo.txt");

Date d = new Date(f.lastModified());

Calendar c = new GregorianCalendar();

//Acceder a un método

c.setTime(d);

//Mostrar ventana

JOptionPane.showMessageDialog(null,"Nombre: "+f.getName()

+"\nFecha: "+Integer.toString(c.get(Calendar.DATE))+"/"

+Integer.toString(c.get(Calendar.MONTH))+"/"

+Integer.toString(c.get(Calendar.YEAR))+" - "

+Integer.toString(c.get(Calendar.HOUR\_OF\_DAY))+":"

+Integer.toString(c.get(Calendar.MINUTE))

+"\nLongitud: "+f.length()

+"\nRuta: "+f.getAbsolutePath(),"ARCHIVO",JOptionPane.INFORMATION\_MESSAGE);

}

}

//Fin de la clase

**Ejecuta.java**

// Fecha: 07 / Mayo / 2021

// Objetivo: Mostrar la extension de un archivo.

// Elaboro Equipo 4:

// 1) Flores Navarro Emmanuel

// 2) Ramírez Morales Erick Brian

//Inicio de la clase

public class Ejecuta {

public static void main(String args[]) {

//Variable de instancia

Menu menu = new Menu();

//Acceder a un método

menu.iniciarMenu();

}

}

//Fin de la clase

**Menu.java**

// Fecha: 07 / Mayo / 2021

// Objetivo: Mostrar la extension de un archivo.

// Elaboro Equipo 4:

// 1) Flores Navarro Emmanuel

// 2) Ramírez Morales Erick Brian

//Inicio de la clase

import javax.swing.JOptionPane;

public class Menu {

//Variable de instancia

private Archivo archivo = new Archivo();

//Método constructor

public Menu() {}

//Metódo definido por el programador

public void iniciarMenu() {

//Atributos

String panel;

int opc;

//Inicio Menu

do{

panel = JOptionPane.showInputDialog(null,"Ingrese el numero de la opcion a ejecutar:"

+"\n 1) Inicio"

+"\n 2) Salir","MENU",JOptionPane.QUESTION\_MESSAGE);

//Iniciar excepcion

try {

//Ingresar un valor

opc = Integer.parseInt(panel);

} catch(Exception e) {

//Capturar y mostrar error

JOptionPane.showMessageDialog(null,"ERROR..."+e.getMessage(),"ERROR",JOptionPane.WARNING\_MESSAGE);

opc = 0;

}

switch(opc) {

case 2 -> {

//Mostrar mensaje y cerrar panel

JOptionPane.showMessageDialog(null,"Gracias por usar este programa, hasta luego!!!");

System.exit(0);

}

case 1 -> {

//Acceder a un método

archivo.leerArchivo();

opc = 2;

}

default -> {

//Mostrar mensaje

JOptionPane.showMessageDialog(null,"Opcion no valida","ERROR",JOptionPane.WARNING\_MESSAGE);

}

}

}while(opc != 2);

//Fin Menu

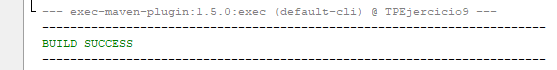
}

}

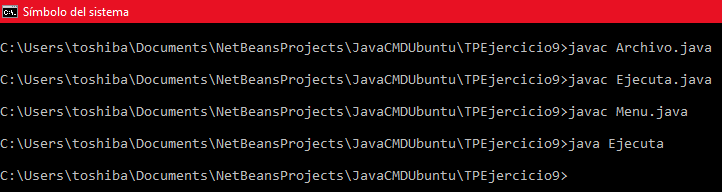
//Fin de la clase

**CAPTURAS DE PANTALLA**

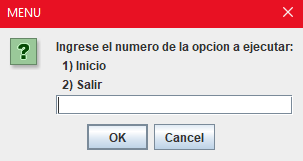
**NetBeans**

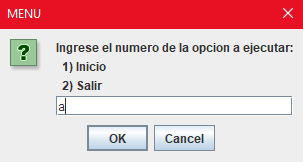


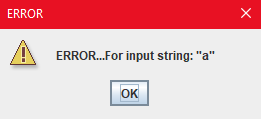
**CMD**

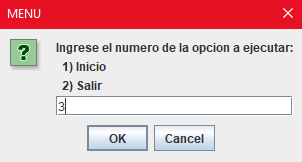


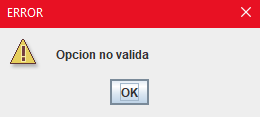
**Ventanas de ejecución**

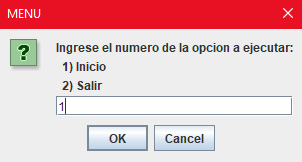


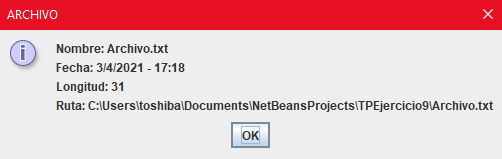


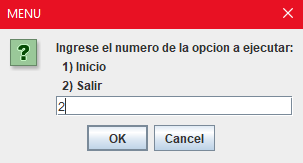


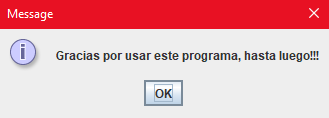




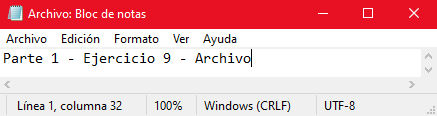








**Archivo.txt**



**CONCLUSIÓN**

Un archivo es una estructura diseñada que nos ayuda a contener datos que están organizados, de tal modo que puedan recuperarse fácilmente, borrarse, actualizarse o almacenarse de nuevo en el archivo con todos los cambios realizados.

# **BIBLIOGRAFÍA**

Céron, C. (27 de Abril de 2021). *Unidad4\_FlujosES*. Obtenido de Blackboard.

Joyanes, L., & Zahonero, I. (2011). Programación en Java 6. En L. Joyanes, & I. Zahonero, *Programación en Java 6* (págs. 407-410). México, D.F: Mc Graw Hill.

**PROGRAMACIÓN II**

**RÚBRICA HOJA DE EVALUACIÓN DE PRÁCTICAS**

MATRICULA: 201955520 FECHA: 07 / Mayo / 2021

NOMBRE: Ramírez Morales Erick Brian No. PRACTICA: Ejercicio 9 INDIVIDUAL ( ) COLABORATIVA ( X )

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Deficiente** | **Suficiente** | **Bueno** | **Excelente** | **Calificación Obtenida** | |
| **CRITERIOS** | **1-5.9** | **6‐7.9** | **8‐9** | **9.1‐10** |
| **%** | **Puntos** |
| **CONOCIMIENTO TEORICO**  **20%** | Conocimiento deficiente de los fundamentos teóricos de POO y no puede aplicarlos en el diseño de clases. | Conocimiento confuso de los fundamentos teóricos de POO y el diseño de las clases y relaciones es incompleto. | Conocimiento claro de los fundamentos teóricos POO pero requiere mejorar el modelado de las clases, sus métodos y sus relaciones entre clases. | Dominio del Conocimiento de los fundamentos teóricos POO y puede aplicarlos de forma completa en el modelado de las clases, métodos y todas las relaciones entre clase. |  |  |
| **EJECUCCIÓN DE LA PRACTICA**  **30%** | No puede realizar la práctica ya que desconoce el entorno de trabajo y desarrollo de la práctica en lenguaje UML y Java. | Realiza la práctica de forma incompleta ya que desconoce el entorno de trabajo del lenguaje UML y Java | Realiza la práctica, pero requiere mejorar en el manejo del entorno de trabajo del lenguaje de programación (sintaxis y  semántica) | Realiza la práctica de forma correcta y completa, demuestra dominio del entorno de trabajo del lenguaje de programación (sintaxis y semántica). |  |  |
| **SOLUCIÓN DE LA PRACTICA**  **40%** | No puede generar las soluciones o programas a los problemas planteados ya que no posee el dominio teórico y práctico del modelado y el lenguaje de programación. | Propone soluciones confusas o programas incompletos a los problemas planteados, ya que carece del dominio del modelado y lenguaje de programación | Genera soluciones con poca profundidad y los programas no están orientados de acuerdo a los problemas solicitados, por lo cual no tiene un dominio profundo de la temática y del Lenguaje. | Genera soluciones con profundidad y los programas son correctos de acuerdo a los problemas planteados, por lo cual demuestra un dominio de la temática y del Lenguaje de Programación. |  |  |
| **ACTITUD DE APRENDER Y COLABORAR EN EQUIPO DE TRABAJO**  **10%** | No posee una actitud proactiva para un aprendizaje autónomo y no le gusta participar y trabajar en equipo. | Posee una actitud propositiva para un aprendizaje autónomo, participa pero no le gusta trabajar en equipo. | Posee una actitud propositiva logrando un aprendizaje autónomo, colaborativo, le gusta trabajar en equipo pero requiere mejorar su participación y portaciones de forma profunda. | Posee una actitud proactiva logrando un aprendizaje autónomo, participa con propuestas concretas y profundas, le gusta trabajar en equipo y asume su responsabilidad dentro para lograr éxito del equipo |  |  |
| **Total**  Equipo 4 | | | | |  |  |

****

**Firma del Alumn@**

**Vo. Bo. Docente**