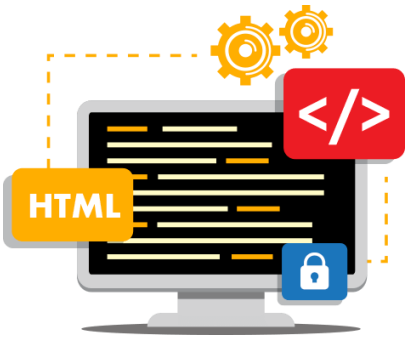
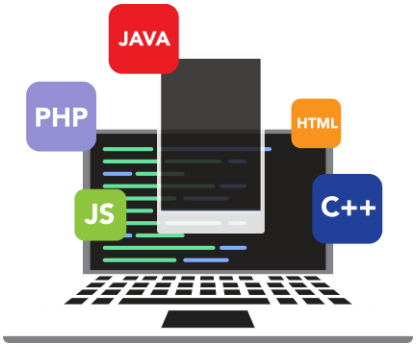
Programación Orientada a Objetos



Grupo: 3203

Alumno: Jesús Navarrete Martínez

Matricula: 202223110



Fecha: 17 de Junio de 2023

Profesora: Yadira Esther Jiménez Pérez

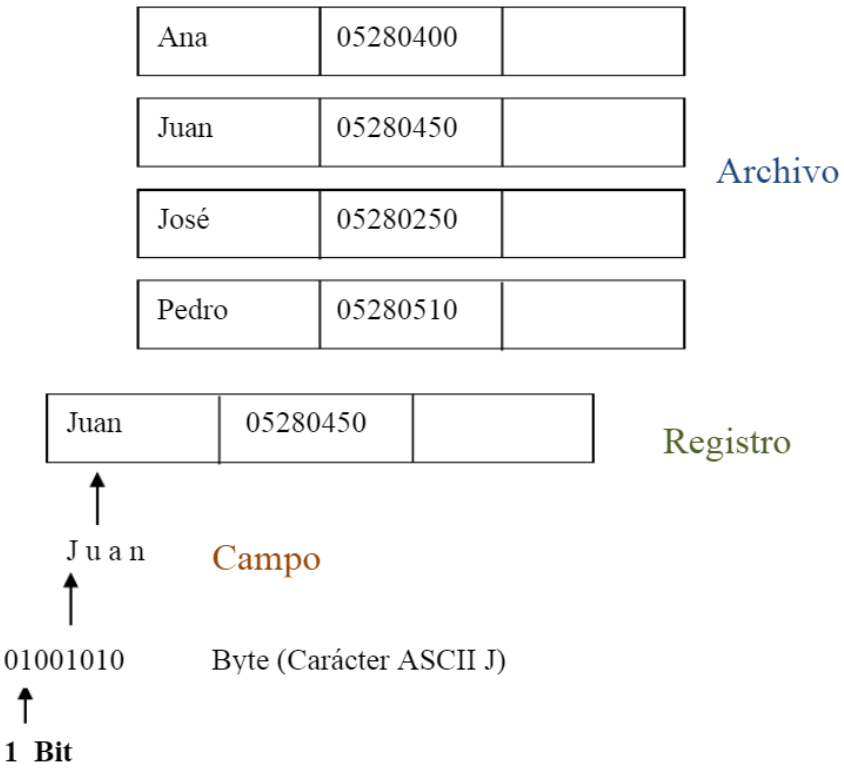
Unidad VI: Flujos y Archivos

¿Qué es un Archivo?

Un archivo es un conjunto de datos o registros relacionados entre sí, los cuales pueden contener información común y organizada para un propósito en específico, por ejemplo: el archivo de una clase escolar contiene registros e información especifica sobre los estudiantes de esa clase.

Un archivo se considera una estructura diseñada para poder contener datos, los cuales deben estar organizados de tal forma que fácilmente puedan recuperarse, borrarse, actualizarse e inclusive almacenarse de nuevo con los cambios realizados en él.

Ejemplo de la estructura de un archivo:



¿Qué es un flujo?

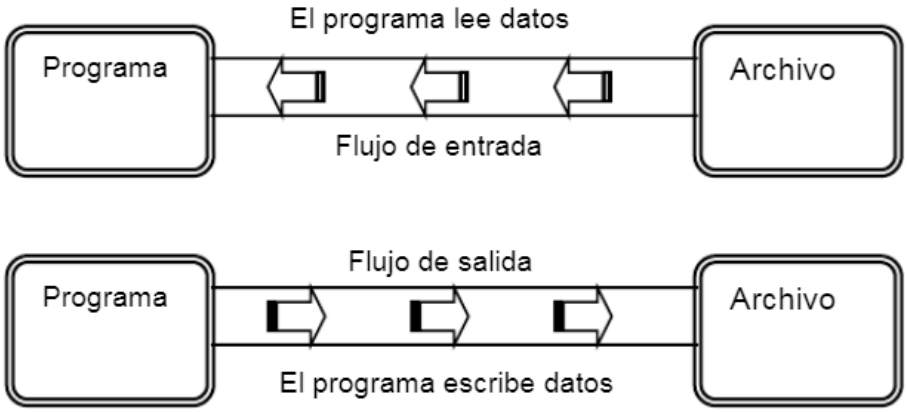
Un flujo de datos nos permite leer o escribir datos con el teclado de nuestra computadora, acceder a un archivo dentro de nuestro disco duro e inclusive generar archivos con datos ingresados por el usuario.

Los flujos pueden ser utilizados para la entrada y salida de datos principalmente, ya sea como bytes o caracteres según la necesidad que exista en el momento.

Tipos de flujos:

- **Flujos basados en bytes:** reciben y envían datos en su formato binario; un char es de dos bytes, un int es de cuatro bytes, un double es de ocho bytes, etcétera.
- **Flujos basados en caracteres:** reciben y envían datos como una secuencia de caracteres en la que cada carácter es de dos bytes; el número de bytes para un valor dado depende del número de caracteres en ese valor. Por ejemplo, el valor 2000000000 requiere 20 bytes (10 caracteres a 2 bytes por carácter), pero el valor 7 requiere sólo dos bytes (1 carácter a dos bytes por carácter).

Los programas en java realizan operaciones de entrada y salida de datos a través de flujos, como la escritura de teclado, la pantalla o los registros de una impresora.



Las diferentes clases de flujos se encuentran agrupadas en el paquete java.io, y en la parte superior de esta jerarquía destacan las clases:

- **OutputStream:** es una clase abstracta que trata flujos de salida de bytes.
- **InputStream:** es una clase abstracta que trata flujos de entrada de bytes.
- **Writer:** es una clase abstracta para escribir caracteres en flujos. Reader, es una clase abstracta para leer caracteres.
- **RandomAccessFile:** permite implementar archivos de acceso directo.

Clasificación: Archivos de texto y Binarios

Archivo de texto: Estos se crean utilizando flujos basados en caracteres, es decir contienen información de tipo textual, sin información sobre su formato, por lo que se dice que es un archivo de texto plano, estos se pueden leer con editores de texto, además se orientan por líneas que facilitan su lectura.

Archivo binario: Se crean utilizando flujos basados en bytes, este archivo contiene información de cualquier tipo la cual esta codificada en un sistema binario, con la finalidad de almacenar y procesar todo tipo de información tal como imágenes e inclusive sonidos. Solo pueden ser leídos por programas que comprenden el archivo y su orden.

Archivo de texto de E/S. Para datos primitivos. Fácil de comprender.
La computadora transforma datos primitivos de formato nativo en un formato de texto legible para archivos.
El lector puede crear o ver archivos de texto con casi cualquier editor de texto.

para salida hacia un archivo de texto:
`PrintWriter`
`FileWriter`
para entrada desde un archivo de texto:
`Scanner`
`FileReader`

Archivo binario de E/S. Para datos primitivos. Eficiente.
Los archivos obtienen datos primitivos en formato nativo.
No es posible crear ni ver archivos binarios con un editor de texto, aunque los archivos binarios son bastante compactos.

para salida hacia un archivo binario:
`DataOutputStream`
`FileOutputStream`
para entrada desde un archivo binario:
`DataInputStream`
`FileInputStream`

Archivo objeto de E/S. Para objetos completos. Fácil de usar.
La computadora descompone objetos en datos primitivos para archivos, aunque también obtiene encabezados descriptivos.
No es posible crear ni ver archivos objeto con un editor de texto, aunque la codificación se minimiza.

para salida hacia un archivo objeto:
`ObjectOutputStream`
`FileOutputStream` <lo mismo que para salida binaria>
para entrada desde un archivo objeto:
`ObjectInputStream`
`FileInputStream` <lo mismo que para entrada binaria>

Tipos de acceso:

Existen dos tipos de acceso (secuencial y directo) , estos se clasifican conforme a las características del soporte empleado y el modo en que se encuentran organizados.

Acceso secuencial: Implica el acceso a un archivo según el orden de almacenamiento de los registros que contiene, uno tras otro.

Acceso directo: Este supone el acceso a un registro determinado, sin que ello involucre la consulta de los registros precedentes, este tipo de acceso solo es posible con soportes direccionales.

Ejemplo de la escritura de un archivo con un acceso secuencial:

```
1  // Fig. 15.3: CrearArchivoTexto.java
2  // Escribir datos en un archivo de texto secuencial mediante la clase Formatter.
3  import java.io.FileNotFoundException;
4  import java.lang.SecurityException;
5  import java.util.Formatter;
6  import java.util.FormatterClosedException;
7  import java.util.NoSuchElementException;
8  import java.util.Scanner;
9
10 public class CrearArchivoTexto
11 {
12    private static Formatter salida; // envía texto a un archivo
13
14    public static void main(String[] args)
15    {
16       abrirArchivo();
17       agregarRegistros();
18       cerrarArchivo();
19    }
20
21    // abre el archivo clientes.txt
22    public static void abrirArchivo()
23    {
24       try
25       {
26          salida = new Formatter("clientes.txt"); // abre el archivo
27       }
28       catch (SecurityException securityException)
29       {
30          System.err.println("Permiso de escritura denegado. Terminando.");
31          System.exit(1); // termina el programa
32       }
33       catch (FileNotFoundException fileNotFoundException)
34       {
35          System.err.println("Error al abrir el archivo. Terminando.");
36          System.exit(1); // termina el programa
37       }
38    }
39
40    // agrega registros al archivo
41    public static void agregarRegistros()
42    {
43       Scanner entrada = new Scanner(System.in);
44       System.out.printf("%s%n%s%n? ",
45          "Escriba numero de cuenta, nombre, apellido y saldo.",
46          "Para terminar la entrada, escriba el indicador de fin de archivo.");
47
48       while (entrada.hasNext()) // itera hasta encontrar el indicador de fin de
                                  archivo
49       {
50          try
51          {
```

Fig. 15.3 | Escritura de datos en un archivo de texto secuencial mediante la clase `Formatter` (parte 1 de 2).

```
52              // escribe el nuevo registro en el archivo; asume una entrada válida
53              salida.format("%d %s %s %.2f%n", entrada.nextInt(),
54                 entrada.next(), entrada.next(), entrada.nextDouble());
55           }
56           catch (FormatterClosedException formatterClosedException)
57           {
58              System.err.println("Error al escribir en el archivo. Terminando.");
59              break;
60           }
61           catch (NoSuchElementException elementException)
62           {
63              System.err.println("Entrada invalida. Intente de nuevo.");
64              entrada.nextLine(); // descarta la entrada para que el usuario
                intente de nuevo
65           }
66
67           System.out.print("? ");
68        } // fin de while
69     } // fin del método agregarRegistros
70
71     // cierra el archivo
72     public static void cerrarArchivo()
73     {
74        if (salida != null)
75           salida.close();
76     }
77  } // fin de la clase CrearArchivoTexto
```

```
Escriba numero de cuenta, nombre, apellido y saldo.
Para terminar la entrada, escriba el indicador de fin de archivo.
? 100 Bob Blue 24.98
? 200 Steve Green -345.67
? 300 Pam White 0.00
? 400 Sam Red -42.16
? 500 Sue Yellow 224.62
? ^Z
```

Fig. 15.3 | Escritura de datos en un archivo de texto secuencial mediante la clase `Formatter` (parte 2 de 2).

Ejemplo de lectura de un archivo con acceso secuencial:

```
1  // Fig. 15.6: LeerArchivoTexto.java
2  // Este programa lee un archivo de texto y muestra cada registro.
3  import java.io.IOException;
4  import java.lang.IllegalStateException;
5  import java.nio.file.Files;
6  import java.nio.file.Path;
7  import java.nio.file.Paths;
8  import java.util.NoSuchElementException;
9  import java.util.Scanner;
10
11 public class LeerArchivoTexto
12 {
13    private static Scanner entrada;
14
15    public static void main(String[] args)
16    {
17       abrirArchivo();
18       leerRegistros();
19       cerrarArchivo();
20    }
21
22    // abre el archivo clientes.txt
23    public static void abrirArchivo()
24    {
25       try
26       {
27          entrada = new Scanner(Paths.get("clientes.txt"));
28       }
```

Fig. 15.6 | Lectura de un archivo secuencial mediante un objeto `Scanner` (parte 1 de 2).

```
29          catch (IOException ioException)
30          {
31             System.err.println("Error al abrir el archivo. Terminando.");
32             System.exit(1);
33          }
34       }
35
36       // lee registro del archivo
37       public static void leerRegistros()
38       {
39          System.out.printf("%-10s%-12s%-12s%10s%n", "Cuenta",
40             "Primer nombre", "Apellido paterno", "Saldo");
41
42          try
43          {
44             while (entrada.hasNext) // mientras haya más qué leer
45             {
46                // muestra el contenido del registro
47                System.out.printf("%-10d%-12s%-12s%10.2f%n", entrada.nextInt(),
48                   entrada.next(), entrada.next(), entrada.nextDouble());
49             }
50          }
51          catch (NoSuchElementException elementException)
52          {
53             System.err.println("El archivo no esta bien formado. Terminando.");
54          }
55          catch (IllegalStateException stateException)
56          {
57             System.err.println("Error al leer del archivo. Terminando.");
58          }
59       } // fin del método leerRegistros
60
61       // cierra el archivo y termina la aplicación
62       public static void cerrarArchivo()
63       {
64          if (entrada != null)
65             entrada.close();
66       }
67    } // fin de la clase LeerArchivoTexto
```

```
Cuenta     Primer nombre     Apellido paterno     Saldo
100        Bob               Blue                 24.98
200        Steve             Green                -345.67
300        Pam               White                0.00
400        Sam               Red                  -42.16
500        Sue               Yellow               224.62
```

Fig. 15.6 | Lectura de un archivo secuencial mediante un objeto `Scanner` (parte 2 de 2).

Operaciones básicas sobre archivos:

a) Consulta: Consiste en la recuperación del contenido de un registro
b) Modificación: Es la alteración de la información contenida en un registro.
c) Inserción: Se le agrega un nuevo registro al archivo
d) Borrado: Supresión de uno o varios elementos del archivo

Nota: Las operaciones sobre archivos se relacionan mediante programas en donde los archivos se identifican por un nombre externo al que están asociados.

La mayoría de los programas ejecutan las siguientes funciones u operaciones al accesar archivos:

- **Creación de archivos:** Antes de que cualquier usuario pueda procesar un archivo es preciso que este haya sido creado previamente. Esta es la primera operación que se realiza, mediante la cual se identifica la información correspondiente al archivo en el soporte de almacenamiento. En términos generales los datos que se requieren son:
    a) Nombre del dispositivo de almacenamiento o ruta de acceso.
    b) Nombre del archivo
    c) Tamaño del archivo
    d) Organización del archivo
    e) Tamaño del bloque o registro.
- **Apertura del archivo:** Es la que permite al usuario tener disponible y acceder los archivos, los datos requeridos generalmente son nombre del archivo, nombre del dispositivo de almacenamiento y nombre del canal de comunicación.
- **Transferencia de datos desde el archivo (lectura) o al archivo (escritura):** Se llama lectura a la recuperación de los datos almacenados en el archivo para su manipulación en memoria principal. La transferencia de datos hacia el archivo se llama escritura y es el almacenamiento de los datos ya procesados en memoria principal.
- **Cerrar el archivo:** El propósito de esta operación es permitir al usuario detener el uso del archivo.

Manejo de objetos persistentes:

Esto significa guardar objetos en un lugar de almacenamiento, como una base de datos o un archivo, para que puedan ser utilizados incluso después de que el programa se haya cerrado. En lugar de perder los datos cuando se cierra el programa, los objetos persistentes se guardan y se pueden recuperar más tarde.

Esto es útil cuando necesitamos almacenar y mantener datos importantes a largo plazo.

Para que un objeto persista una vez que termina la ejecución de una aplicación se debe guardar en un archivo de objetos; por cada objeto que se almacena en un archivo se graban características de la clase y atributos del objeto.

Bibliografía:

- Joyanes. (2011). PROGRAMACION EN JAVA 6: Algoritmos y Programación orientada a objetos (1.ª ed.). Mc Graw Hill Educación.
- Deitel, H. M., & Deitel, P. D. (2016). *Java: como programar* (10.ª ed.). Pearson Educacion.
- Dean, J. S. (2000). *Introducción a la programación con Java*.
- Studocu. (n.d.). *TEMA 6 POO Archivos - 1 6. FLUJOS Y ARCHIVOS 6 Definición. Un archivo es un conjunto de datos - Studocu*. https://www.studocu.com/es-mx/document/instituto-tecnologico-de-toluca/programacion-orientada-a-objetos/tema-6-poo-archivos/25403850
- Guest. (n.d.). Unidad 6 Programacion orientada a objetos Archivos - PDFCOFFEE.COM. *pdfcoffee.com*. https://pdfcoffee.com/unidad-6-programacion-orientada-a-objetos-archivos-2-pdf-free.html