Unidad VI

Archivos.

6.1 Definición de Archivos de texto y archivos binarios.

Los archivos de texto plano son aquellos que están compuestos únicamente por texto sin formato, solo caracteres. estos caracteres se pueden codificar de distintos modos dependiendo de la lengua usada. Se les conoce también como archivos de texto llano o texto simple por carecer de información destinada a generar formatos y tipos de letra.

Un archivo binario es una archivo informático que contiene información de cualquier tipo, codificada en forma binaria para el propósito de almacenamiento y procesamiento de ordenadores.

Muchos formatos binarios contienen partes que pueden ser interpretados como texto. Un archivo binario que solo contiene información de tipo textual sin información sobre el formato del mismo, se dice que es un archivo de texto plano. Habitualmente se contraponen los términos archivo binario y archivo de texto de forma que los primeros no contienen solamente texto.

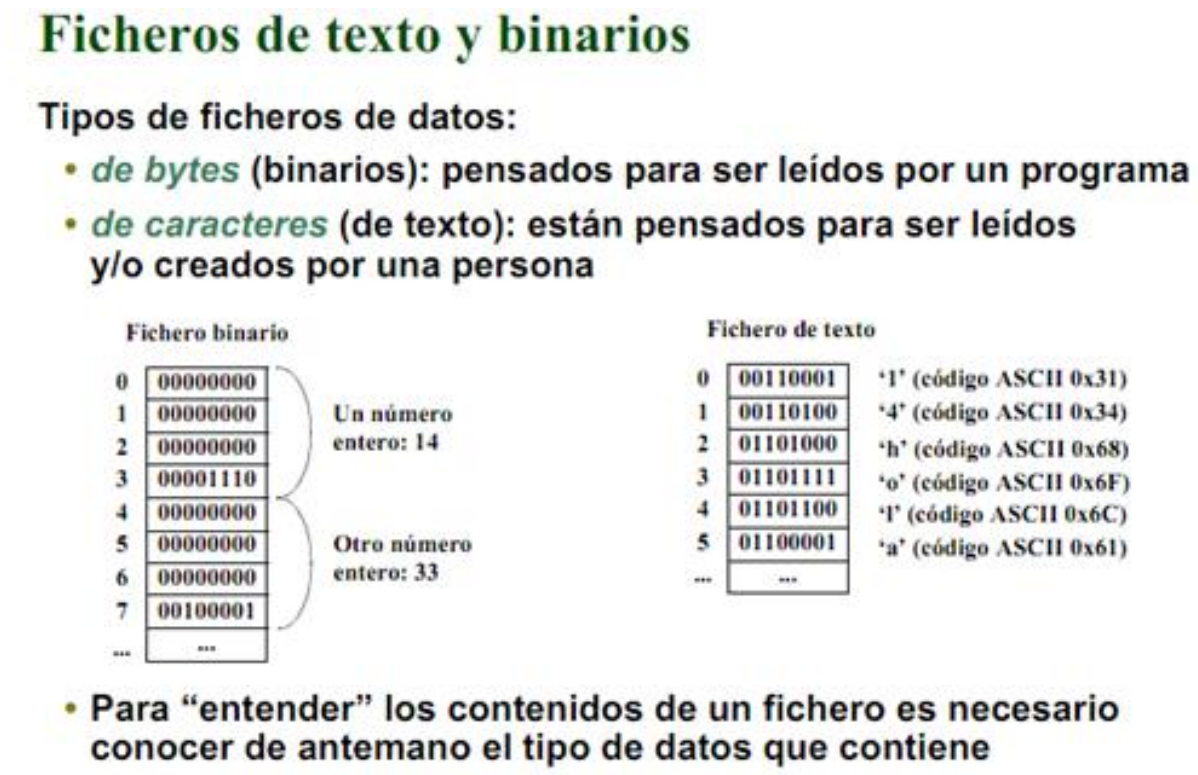
6.2 Operaciones básicas en archivos texto y binario.

Los archivos de texto plano son aquellos que están compuestos únicamente por texto sin formato, solo caracteres. estos caracteres se pueden codificar de distintos modos dependiendo de la lengua usada. Se les conoce también como archivos de texto llano o texto simple por carecer de información destinada a generar formatos y tipos de letra.

Un archivo binario es una archivo informático que contiene información de cualquier tipo, codificada en forma binaria para el propósito de almacenamiento y procesamiento de ordenadores.

Muchos formatos binarios contienen partes que pueden ser interpretados como texto. Un archivo binario que solo contiene información de tipo textual sin información sobre el formato del mismo, se dice que es un archivo de texto plano.

Habitualmente se contraponen los términos archivo binario y archivo de texto de forma que los primeros no contienen solamente texto.



EJEMPLO:

clase 1.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class Demostracionfile extends JFrame{
    private JTextArea areaSalida;
    private JScrollPane panelDespl;
```

```
    //constructor de la interfaz
    public Demostracionfile(){
            super("Ejemplo de archivos");

            areaSalida= new JTextArea();
            panelDespl=new JScrollPane(areaSalida);
            add(panelDespl,BorderLayout.CENTER);
           setSize(400,400); //establece el tamaño de la interfaz
            setVisible(true);//muestra la interfaz GUI8

            analizarRuta(); //crear y analizar un objeto File
    } // fin del contructor

    private File obtenerArchivo(){
    JFileChooser selectorArchivos=new JFileChooser();

selectorArchivos.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES
);
    int resultado = selectorArchivos.showOpenDialog(this);

    if(resultado == JFileChooser.CANCEL_OPTION)
        System.exit(1);
    File nombreArchivo = selectorArchivos.getSelectedFile(); //obtiene el nombre del
archivo

    if ((nombreArchivo==null)||(nombreArchivo.getName().equals(""))) {
        JOptionPane.showMessageDialog(this, "nombre del archivo
invalido","nombre del archivo invalido", JOptionPane.ERROR_MESSAGE );
        System.exit(1);
    }//fin del IF
    return nombreArchivo;
    }
    //yet..another class
    public void analizarRuta(){
    File nombre = obtenerArchivo();
    if (nombre.exists())
    {

areaSalida.setText(String.format("%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n
%s%s\n%s%s",nombre.getName(),"existe",
                (nombre.isFile()?"es un archivo":"no es un archivo"),
                (nombre.isDirectory() ? "no es directorio":"no es directorio"),
```

```
            (nombre.isAbsolute() ? "es una ruta absoluta":"no es un a ruta
absoluta"),
            "ultima modifecacion: ",nombre.lastModified(), "tamaño:
",nombre.length(),
            "Ruta: ",nombre.getPath(), "Ruta absoluta: ",
            nombre.getAbsolutePath(),"Padre: ",nombre.getParent() ));
        if(nombre.isDirectory())//imprime el listado del directorio
        {
            String directorio[] = nombre.list();
            areaSalida.append("\n\nContenido del directorio: \n");

            for(String nombreDirectorio:directorio)
                areaSalida.append(nombreDirectorio+"\n");
        }

    }

    else
    {
        JOptionPane.showMessageDialog(this, nombre + "no existe",
"ERROR",JOptionPane.ERROR_MESSAGE);

        }
    }


    }
clase 2.

 import javax.swing.JFrame;
public class pruebademostracionfile {
    public static void main(String args[]){
    Demostracionfile aplicacion = new Demostracionfile();
    aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

}
```

6.3 Manejo de excepciones en archivos

Una excepción en términos de lenguaje de programación es la indicación de un problema que ocurre durante la ejecución de un programa. Sin embargo la palabra excepción se refiere que este problema ocurre con poca frecuencia generalmente cuando existe algún dato o instrucción que no se apega al funcionamiento del programa por lo que se produce un error. El manejo de excepciones permite al usuario crear aplicaciones tolerantes a fallas y robustos (resistentes a errores) para controlar estas excepciones y que pueda seguir ejecutando el programa sin verse afectado por el problema. En lenguaje java estas excepciones pueden manejarse con las clases que extienden el paquete Throwable de manera directa o indirecta, pero existen diversos tipos de excepciones y formas para manejarlas.

Uso del manejo de excepciones

El manejo de excepciones ayuda al programador a remover el código para manejo de errores de la línea principal de ejecución, además se puede elegir entre manejar todas las excepciones, las de cierto tipo o de las de grupos relacionados, esto hace que la probabilidad de pasar por alto los errores se reduzca y a la vez hace los programas más robustos. Pero es importante utilizar un lenguaje de programación que soporte este manejo, de lo contrario el procesamiento de errores no estará incluido y hará el programa más vulnerable. Este manejo está diseñado para procesar errores que ocurren cuando se ejecuta una instrucción, algunos ejemplos son: desbordamiento aritmético, división entre cero, parámetros inválidos de método y asignación fallida en la memoria. Sin embargo no está diseñado para procesar problemas con eventos independientes al programa como son pulsar una tecla o clic al mouse. Las excepciones se dividen en verificadas y no verificadas. Es importante esta división porque el compilador implementa requerimientos de atrapar o declarar para las verificadas lo que hará que se detecten las excepciones automáticamente y de acuerdo al lenguaje de programación utilizado se utilizará un método para corregirlas. Sin embargo para las no verificadas se producirá un error indicando que deben atraparse y declararse. Por eso el programador debe pensar en los problemas que pueden ocurrir cuando se llama a un método y definir excepciones para verificarse cuando sean importantes. Las clases de excepciones pueden derivarse de una superclase común, por lo que con un manejador para atrapar objetos de la superclase, también se pueden atrapar todos los objetos de las subclases de esa clase. Pero también, se pueden atrapar a cada uno de los tipos de las subclases de manera individual si estas requieren ser procesadas diferente. A cada celula se le conoce como compiladora de distintos.

Limpieza de pila

En ocasiones cuando se hace lanza una excepción, pero no se atrapa en un enlace específico, la pila de llamadas se limpia y el programa intenta volverlo a atrapar en el siguiente bloque, esto se conoce como limpia de pila. Este proceso hace que el método en el que no se atrapó la excepción termina, todas sus

variables quedan fuera del enlace y el control regresa a la instrucción que originalmente la invocó. La limpieza de pila de repetirá hasta que la excepción pueda ser atrapada porque de lo contrario se producirá un error a la hora de compilar.

Aserciones[editar · editar código]

Las aserciones ayudan a asegurar la validez del programa al atrapar los errores potenciales e identificar los posibles errores lógicos del desarrollo. Estas pueden escribirse como comentarios para apoyar a la persona que desarrolla el programa. Algunos ejemplos son: Precondiciones y pos condiciones Estas características son utilizadas por los programadores para hacer un análisis de lo esperado del programa antes y después de su ejecución. Son importantes porque gracias a ellas se pueden detectar posibles fallas en el programa y corregirlas. Las precondiciones son verdaderas cuando se invoca a un método, estas describen las características del método y las expectativas que se tienen en el estado actual del programa. Si no se cumplen las precondiciones el comportamiento del método es indefinido por lo que se lanza una excepción que esté preparada o continuar con el programa esperando el error. Las pos condiciones describen las restricciones en el entorno y cualquier efecto secundario del método. Es recomendable escribirlas para saber que esperar en un futuro si es que se hacen modificaciones.

Conclusión

El manejo de excepciones ayuda a lidiar con los errores de una aplicación por medio de la manipulación del código para hacer programas más robustos. Además existen herramientas que ayudan a manejarlas tal es el caso de los bloques **try** (intentar) que encierran el código que puede lanzar una excepción y los bloques **catch** (atrapar) que lidian con las excepciones que surjan. También existen técnicas que el programador utiliza para conocer el posible funcionamiento del programa y detectar los errores que pueda contener.

Ejemplo de manejo de excepción en Java:

```
import java.io.IOException;

// ...

public static void main(String[] args) {
   try {
      // Se ejecuta algo que puede producir una excepción
   } catch (IOException e) {
      // manejo de una excepción de entrada/salida
   } catch (Exception e) {
      // manejo de una excepción cualquiera
   } finally {
      // código a ejecutar haya o no excepción
```

```
    }
}
```