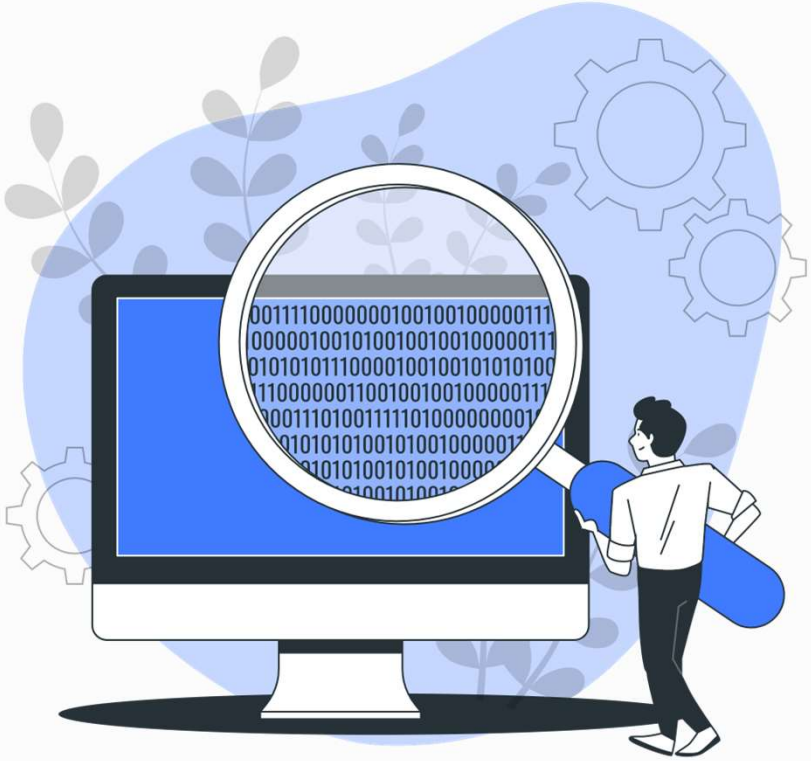
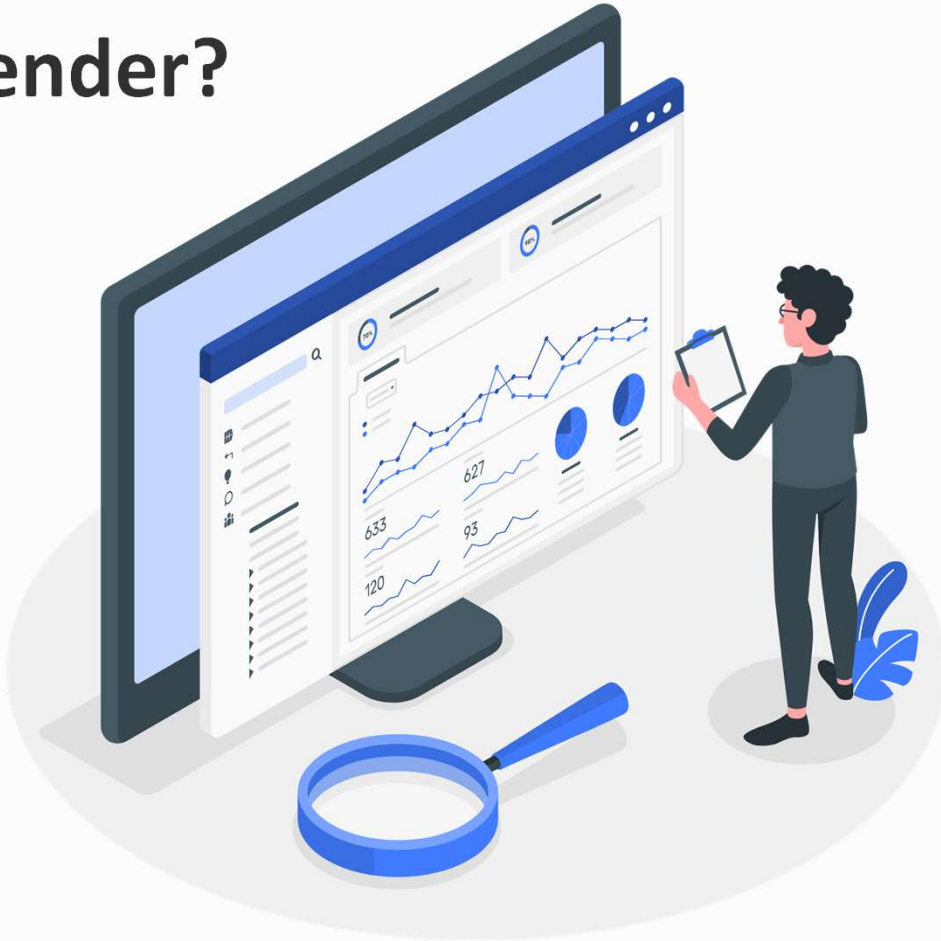
#Lectura y escritura de información

Tema 11



¿Qué vamos a #aprender?

- Definición flujo de comunicación.
- Flujos predeterminados.
  - System.out
  - System.err
  - System.in
- Tipos de Flujos
  - Según la dirección.
  - Según la forma.
  - Según el acceso.



#Introducción

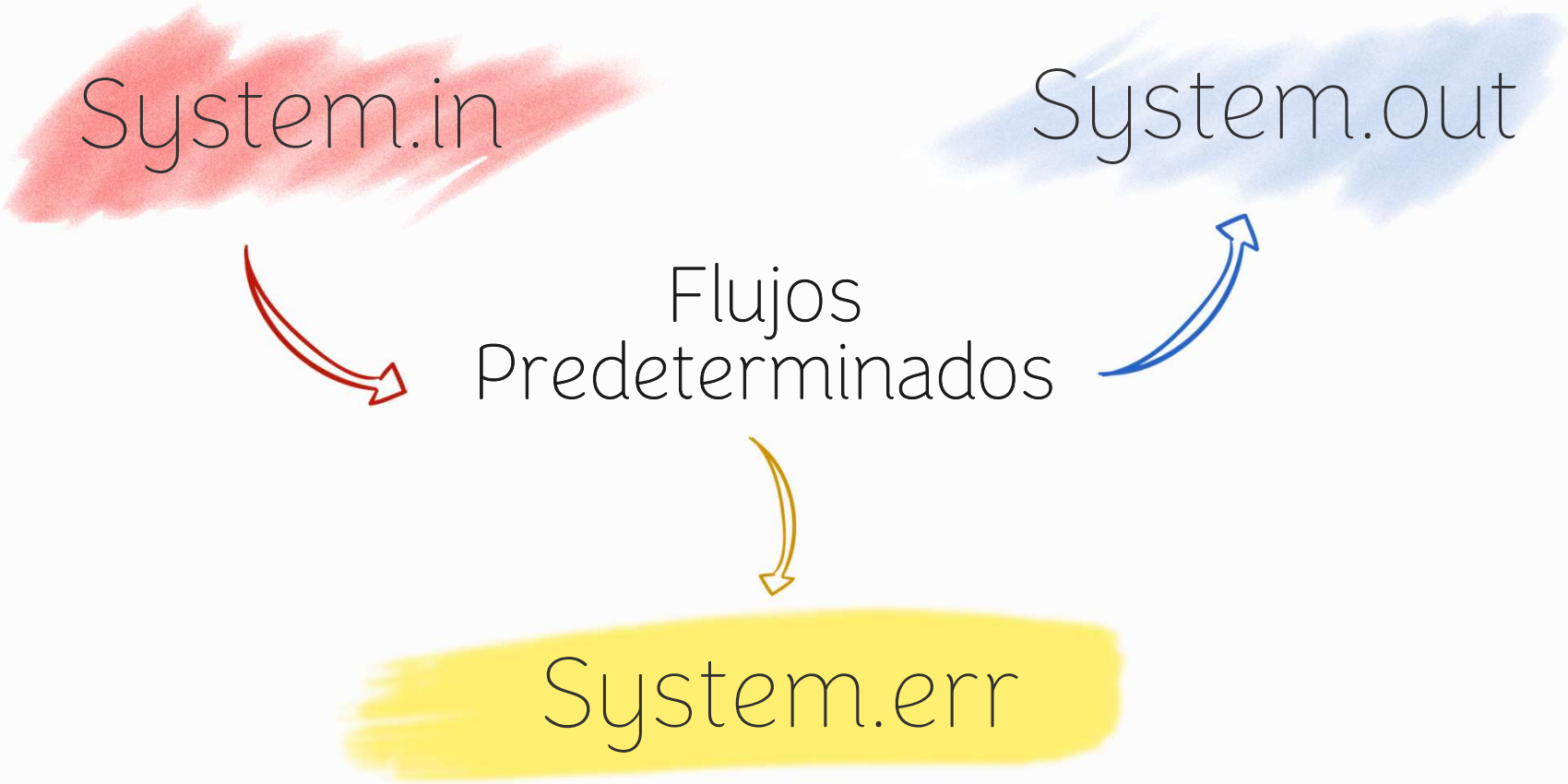
Interacción usuario – programa.

Información **no** se pierda → dispositivo de memoria no volátil.

Flujo de comunicación o *steam*	Conjunto datos de una fuente o destino

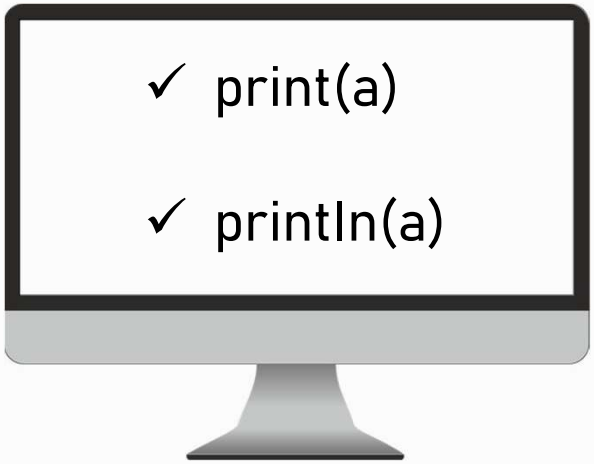
Entrada/Salida (E/S)	Conjunto operaciones de flujo de información del programa con el exterior

- Estándar
- A través de fichero


System.in
System.out
Flujos
Predeterminados
System.err

System.out

Implementa la salida estándar.



Por otro lado, el método *print* imprimirá el mensaje en la misma línea.

```
System.out.print("Hola Mundo");
System.out.print("Mi nombre es Codi");
```

Salida

```
Hola MundoMi nombre es Codi
```

Con el método *println* el mensaje se imprimirá en consola con un salto de línea.

```
System.out.println("Hola Mundo");
System.out.println("Mi nombre es Codi");
```

Salida

```
Hola Mundo
Mi nombre es Codi
```

System.err

Implementa la salida en caso de error.

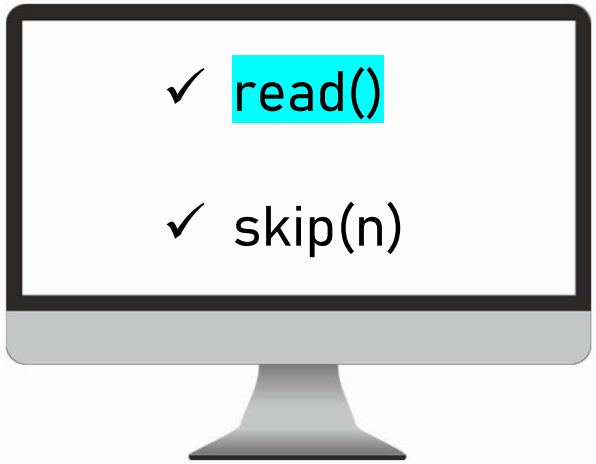
Similar a System.out

```
public class EjemploSystemErr {
    public static void main(String[] args) {
        int dividendo = 10;
        int divisor = 0;

        try {
            int resultado = dividendo / divisor;
            System.out.println("El resultado es: " + resultado);
        } catch (ArithmeticException e) {
            System.err.println("Error: no se puede dividir por cero.");
        }
    }
}
```

```
Error: no se puede dividir por cero.
```

System.in

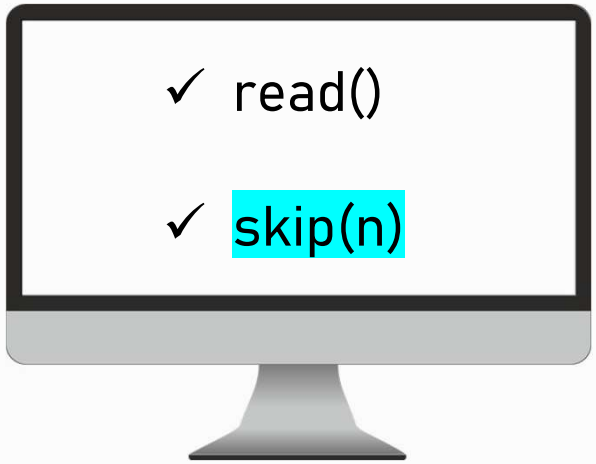
Implementa la entrada estándar.




```java
import java.io.*;

public class ReadExample {
   public static void main(String[] args) throws IOException {
      // crear un flujo de entrada
      InputStream input = System.in;

      // leer un byte de entrada
      int ch = input.read();

      // imprimir el byte leído
      System.out.println("Byte leído: " + ch);
   }
}
```

System.in

Implementa la entrada estándar.

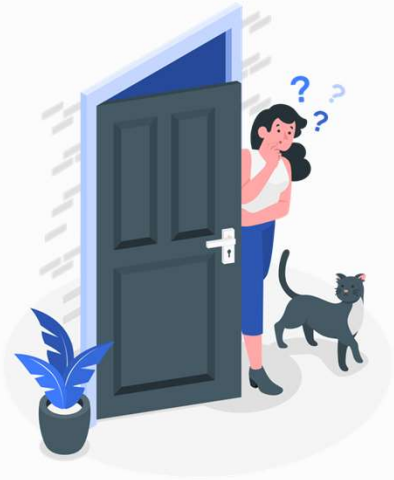



```java
import java.io.*;

public class SkipExample {
    public static void main(String[] args) throws IOException {
        // crear un flujo de entrada
        InputStream input = System.in;

        // saltar los primeros 5 bytes
        long skipped = input.skip(5);

        // leer el siguiente carácter
        int ch = input.read();

        // imprimir el carácter leído y los bytes saltados
        System.out.println("Carácter leído: " + (char) ch);
        System.out.println("Bytes saltados: " + skipped);
    }
}
```

Dirección del flujo

Forma del flujo

Acceso al flujo



Dirección del flujo

- Flujos de entrada
- Flujos de salida
- Flujos de entrada/salida

Forma del flujo

Bytes

Clases conversoras

Caracteres

Acceso al flujo

Acceso secuencial

Métodos
- Int read()
- Long skip(n)
- Void close()

FileReader (String path)

Lee caracteres de un archivo de texto (no codifica/decodifica)

```java
import java.io.FileReader;
import java.io.IOException;

public class EjemploFileReader {
    public static void main(String[] args) {
        String rutaArchivo = "/home/usuario/documentos/datos.txt";

        try (FileReader fr = new FileReader(rutaArchivo)) {
            char[] buffer = new char[1024];
            int charsRead = fr.read(buffer);
            while (charsRead != -1) {
                System.out.print(new String(buffer, 0, charsRead));
                charsRead = fr.read(buffer);
            }
        } catch (IOException e) {
            System.err.println("Error al leer el archivo: " + e.getMessage()
        }
    }
}
```

Acceso al flujo

FileInputStream(String path)

Lee bytes en bruto de un archivo
(no codifica/decodifica)

Acceso secuencial

Métodos

- Int read()
- Long skip(n)
- Void close()

```java
import java.io.*;

public class EjemploFileInputStream {
    public static void main(String[] args) {
        try {
            // Abrir un archivo para lectura
            FileInputStream fis = new FileInputStream("ejemplo.txt");

            // Leer datos del archivo
            int byteLeido;
            while ((byteLeido = fis.read()) != -1) {
                // Procesar el byte leído
                System.out.print((char) byteLeido);
            }

            // Cerrar el archivo
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

Acceso secuencial

Métodos
- Int read()
- Long skip(n)
- Void close()

DataInputStream

↓

Lee datos primitivos

```java
import java.io.*;

public class EjemploDataInputStream {
    public static void main(String[] args) {
        try {
            // Crear un flujo de entrada de archivo
            FileInputStream fis = new FileInputStream("datos.bin");
            DataInputStream dis = new DataInputStream(fis);

            // Leer datos primitivos del flujo de entrada
            int entero = dis.readInt();
            float flotante = dis.readFloat();
            double doble = dis.readDouble();
            boolean booleano = dis.readBoolean();
            String cadena = dis.readUTF();

            // Imprimir los datos leídos
            System.out.println("Entero: " + entero);
            System.out.println("Flotante: " + flotante);
            System.out.println("Doble: " + doble);
            System.out.println("Booleano: " + booleano);
            System.out.println("Cadena: " + cadena);

            // Cerrar el flujo de entrada
            dis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

BufferedReader newBufferedReader (Path path [, Charset cs][, OpenOption opcion1 …, OpenOption opcionN])

Lee caracteres

```java
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class BufferedReaderEjemplo {
    public static void main(String[] args) {
        String rutaArchivo = "ruta/del/archivo.txt"; // Ruta del archivo de

        try {
            Path path = Paths.get(rutaArchivo);

            // Crear el BufferedReader
            BufferedReader bufferedReader = Files.newBufferedReader(path);

            // Leer caracteres del archivo y mostrarlos en la consola
            int caracter; // Variable para almacenar cada caracter leido
            while ((caracter = bufferedReader.read()) != -1) {
                System.out.print((char) caracter);
            }

            bufferedReader.close(); // Cerrar el BufferedReader
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

FileWriter(String path [, boolean append])

Escribe caracteres

Métodos

- Void write(int c)
- Void write(String cadena)
- Void write(char [] array)
- Void flush()
- Void close()

Acceso secuencial

```java
import java.io.*;

public class FileWriterEjemplo {
    public static void main(String[] args) {
        String rutaArchivo = "ruta/del/archivo.txt"; // Ruta del archivo de

        try {
            // Crear un objeto FileWriter con la ruta del archivo
            FileWriter fileWriter = new FileWriter(rutaArchivo);

            // Escribir caracteres en el archivo
            fileWriter.write("Hola, mundo!"); // Escribir una cadena de cara
            fileWriter.write('\n'); // Escribir un salto de línea
            fileWriter.write("¡Hola desde Java!"); // Escribir otra cadena

            fileWriter.close(); // Cerrar el FileWriter
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

Acceso secuencial

FileOutputStream (String path [, boolean append])

Escribe caracteres en modo binario

Métodos

- Void write(int b)
- Void write(byte[] a)
- Void flush()
- Void close()

```java
import java.io.*;

public class FileOutputStreamSimpleEjemplo {
    public static void main(String[] args) {
        String rutaArchivo = "ruta/del/archivo.bin"; // Ruta del archivo bina

        try {
            // Crear un objeto FileOutputStream con la ruta del archivo
            FileOutputStream fileOutputStream = new FileOutputStream(rutaArch

            // Escribir un byte en el archivo
            int dato = 65; // Representa el valor ASCII de la letra 'A'
            fileOutputStream.write(dato); // Escribir el byte en el archivo

            fileOutputStream.close(); // Cerrar el FileOutputStream
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

Métodos generales

OutputStream newOutputStream (Path path [, OpenOption opcion1 …, OpenOption opcionN])

↓

Escribe bytes

```java
import java.io.*;

public class OutputStreamEjemplo {
    public static void main(String[] args) {
        String rutaArchivo = "ruta/del/archivo.bin"; // Ruta del archivo bin

        try {
            // Crear un nuevo flujo de salida (OutputStream) para el archivo
            OutputStream outputStream = new FileOutputStream(rutaArchivo);

            // Escribir datos en el flujo de salida
            byte[] bytes = {72, 111, 108, 97}; // Bytes a escribir en el arc
            outputStream.write(bytes); // Escribir los bytes en el flujo de

            outputStream.close(); // Cerrar el flujo de salida
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

Métodos generales

DataOutputStream

Escribe datos primitivos

```java
public class DataOutputStreamEjemplo7 {
    public static void main(String[] args) throws IOException {
        OutputStream outputStream = new FileOutputStream("datos.bin");
        DataOutputStream dataOutputStream = new DataOutputStream(outputStream);

        String nombre = "Juan";
        int edad = 30;
        double salario = 2500.50;

        // Escribir datos en el flujo de salida
        dataOutputStream.writeUTF(nombre);
        dataOutputStream.writeInt(edad);
        dataOutputStream.writeDouble(salario);

        // Cerrar el flujo de salida de datos
        dataOutputStream.close();

        System.out.println("Datos escritos en el archivo datos.bin exitosamente.");
    }
}
```

Acceso al flujo

Métodos generales

ObjectInputStream

Deserializar objetos desde flujo de entrada - Convertir Objetos serializados (datos binarios) → objetos Java

```java
import java.io.*;

no usages
public class EjemploObjectInputStream {
    public static void main(String[] args) {
        try {
            // Crear un flujo de entrada desde un archivo
            FileInputStream fileInputStream = new FileInputStream("objeto_serializado.ser");

            // Crear un objeto ObjectInputStream para leer desde el flujo de entrada
            ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);

            // Leer el objeto serializado
            Object objeto = objectInputStream.readObject();

            // Realizar las operaciones necesarias con el objeto deserializado
            // ...

            // Cerrar el flujo de entrada
            objectInputStream.close();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Acceso al flujo

Métodos generales

ObjectOutputStream

Escribir objetos en forma de flujo de bytes en un flujo de salida

```java
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.io.IOException;
// Clase que será serializada
2 usages
class MiObjeto implements Serializable {
    no usages
    private static final long serialVersionUID = 1L; // Identificador de versión para la serialización
    1 usage
    private String nombre;
    1 usage
    private int edad;

    1 usage
    public MiObjeto(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Getters y setters (omitidos para simplificar el ejemplo)
}
no usages
public class EjemploObjectOutputStream {
    public static void main(String[] args) {
        // Crear un objeto para serializar
        MiObjeto objeto = new MiObjeto( nombre: "Juan", edad: 30);

        // Crear un flujo de salida
        try (FileOutputStream archivoSalida = new FileOutputStream("objeto_serializado.bin")) {
            // Crear un ObjectOutputStream
            ObjectOutputStream objetoSalida = new ObjectOutputStream(archivoSalida);

            // Escribir el objeto en el flujo de salida
            objetoSalida.writeObject(objeto);

            // Cerrar el flujo de salida
            objetoSalida.close();

            System.out.println("El objeto ha sido serializado y guardado en objeto_serializado.bin");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Acceso al flujo

Acceso directo

RandomAccessFile ➡ Acceso aleatorio de archivos.

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public class RandomAccessFileExample {
    public static void main(String[] args) {
        String filePath = "archivo.txt";
        String mode = "rw";
        try (RandomAccessFile file = new RandomAccessFile(filePath, mode)) {
            String contenido = "Este es el contenido del archivo.";
            file.write(contenido.getBytes());
            System.out.println("Se escribió el contenido en el archivo.");
        } catch (IOException e) {
            System.err.format("Error al escribir en el archivo: %s%n", e);
        }
    }
}
```

Constructores

New RandomAccessFile(File path, String modo)

New RandomAccessFile(String path, String modo)

**ejemplo*

Acceso al flujo

Acceso directo

RandomAccessFile → Acceso aleatorio de archivos.

Fichero de entrada o salida binario con acceso aleatorio

Tipo fichero que permite el acceso a cualquier registro del archivo de forma aleatoria

Acceso al flujo

Acceso directo

Métodos de acceso directo

```
public final void writeDouble(double v) throws IOException
```

```
public final void writeFloat(float v) throws IOException
```

```
public final String readUTF() throws IOException
```

```
public final long length() throws IOException
```

```
public final int readInt() throws IOException
```

#Clase File

- Se utiliza para trabajar con archivos.
- Esta clase proporciona métodos para crear, leer, escribir, eliminar y renombrar archivos y directorios.

```
File archivo = new File("ruta/al/archivo.txt"); // ruta relativa
File archivo = new File("C:/Users/usuario/archivo.txt"); // ruta absoluta
```

Para trabajar con un archivo, primero se crea una instancia de la clase File. Esto se puede hacer utilizando una ruta relativa o absoluta.

#Metodos

- Algunos de los métodos más comunes son:

- **exists():** devuelve true si el archivo o directorio existe, y false en caso contrario.
- **createNewFile():** crea un nuevo archivo en el sistema de archivos.
- **delete():** elimina el archivo o directorio.
- **getName():** devuelve el nombre del archivo o directorio.
- **isDirectory():** devuelve true si el objeto File representa un directorio, y false si representa un archivo.
- **listFiles():** devuelve un arreglo de objetos File que representan los archivos y subdirectorios de un directorio.
- **Y otros como: isFile(),renameTo(),canRead(),canWrite(),getPath(),getAbsolutePath(), getParent(),mkdir(),list().**

#Ejemplos

- Ejemplo 1:

```
File archivo = new File("ruta/al/archivo.txt");
try {
    if (archivo.createNewFile()) {
        FileWriter writer = new FileWriter(archivo);
        writer.write("Este es el contenido del archivo");
        writer.close();
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

En este ejemplo, se crea un nuevo archivo en la ruta especificada y se escribe el texto "Este es el contenido del archivo" en él. Si el archivo ya existía, no se crea uno nuevo.

- Ejemplo 2:

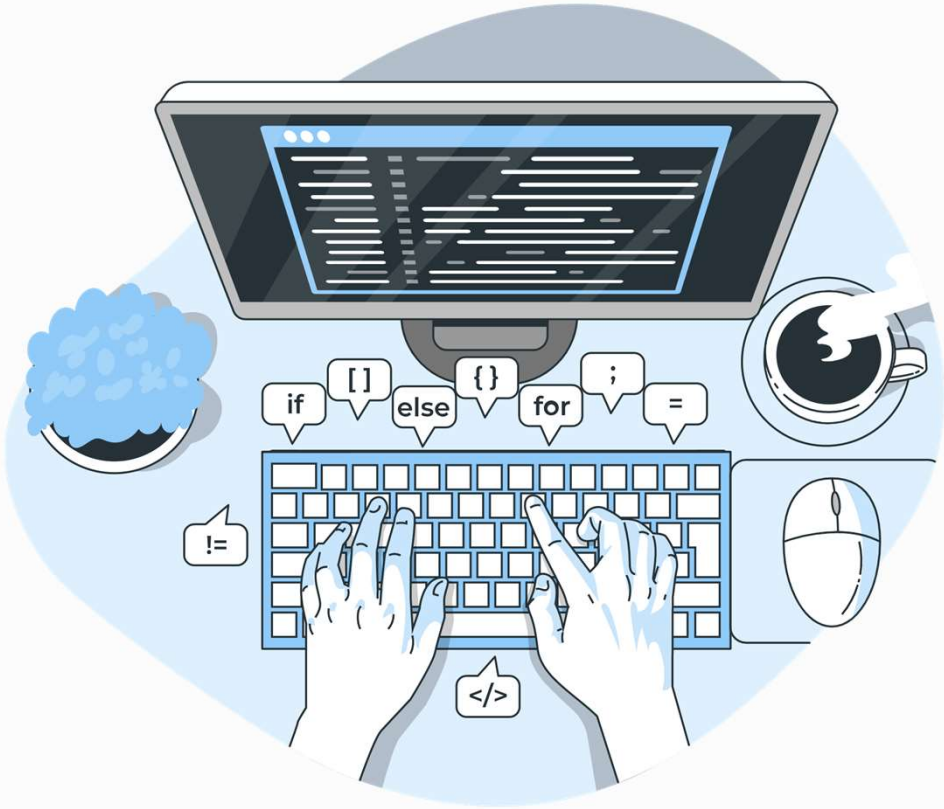
```java
import java.io.File;
import java.io.IOException;

no usages
public class CrearArchivo {
    public static void main(String[] args) {
        // Nombre del archivo a crear
        String nombreArchivo = "nuevo_archivo.txt";

        // Crear un objeto de la clase File para representar el archivo
        File archivo = new File(nombreArchivo);

        try {
            // Intentar crear el archivo con el método createNewFile()
            if (archivo.createNewFile()) {
                System.out.println("El archivo " + nombreArchivo + " ha sido creado exitosamente.");
            } else {
                System.out.println("El archivo " + nombreArchivo + " ya existe.");
            }
        } catch (IOException e) {
            System.out.println("Ha ocurrido un error al crear el archivo.");
            e.printStackTrace();
        }
    }
}
```

#Conclusiones

- Los flujos en Java son una herramienta esencial para el manejo de entrada y salida de datos.
- Es fundamental comprender su funcionamiento y cómo utilizarlos correctamente en el desarrollo de aplicaciones Java.

¡IMPORTANTE!

Como programadores, es importante mantenerse **actualizados** y poder desarrollar soluciones cada vez más avanzadas y efectivas.



#¡Vamos a practicar!