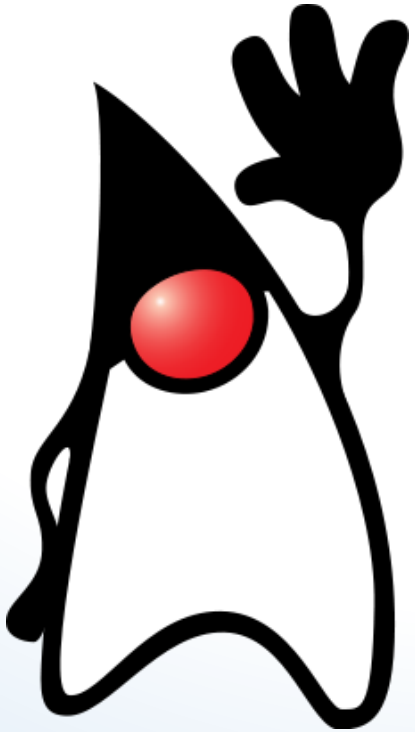
.UBAfiuba
FACULTAD DE INGENIERÍA

Excepciones y persistencia básica



Cátedra: Corsi - Essaya - Maraggi

Definición

Excepción: Una excepción es un **objeto** que se usa para comunicar una situación anómala desde un entorno que la detecta al ámbito desde el cual fue invocado un método.

Se “lanza” cuando en el contexto del método no hay suficiente información para resolver una anomalía, por lo que se debe salir del mismo hacia el módulo que lo invocó informando esta situación, pero **sin provocar la finalización del programa** ni enviar mensajes a un presunto usuario del que no se sabe nada.

Definición: partes

"Una excepción es un objeto que se usa para comunicar una situación anómala desde un entorno que la detecta al ámbito desde el cual fue invocado un método".

1. **Una excepción es un objeto**: en efecto, las excepciones son objetos con identidad, comportamiento y estado. En los lenguajes con clases, son instancias de ciertas clases especiales.
2. **Se envía desde un entorno que la detecta**: habitualmente hay un método que crea y lanza un objeto de excepción porque no puede resolver el problema en su ámbito.
3. **Se envía al ámbito desde el cual fue invocado un método**: el objeto viaja desde el método que detectó el problema hasta el ámbito desde el cual ese método fue invocado. Este ámbito podrá recibir ese objeto y ver de qué manera tratar el problema.
4. **Se usa para comunicar una situación anómala**: su principal uso es la comunicación entre dos ámbitos, el que detecta el problema y el que debe lidiar con el mismo.

Motivación

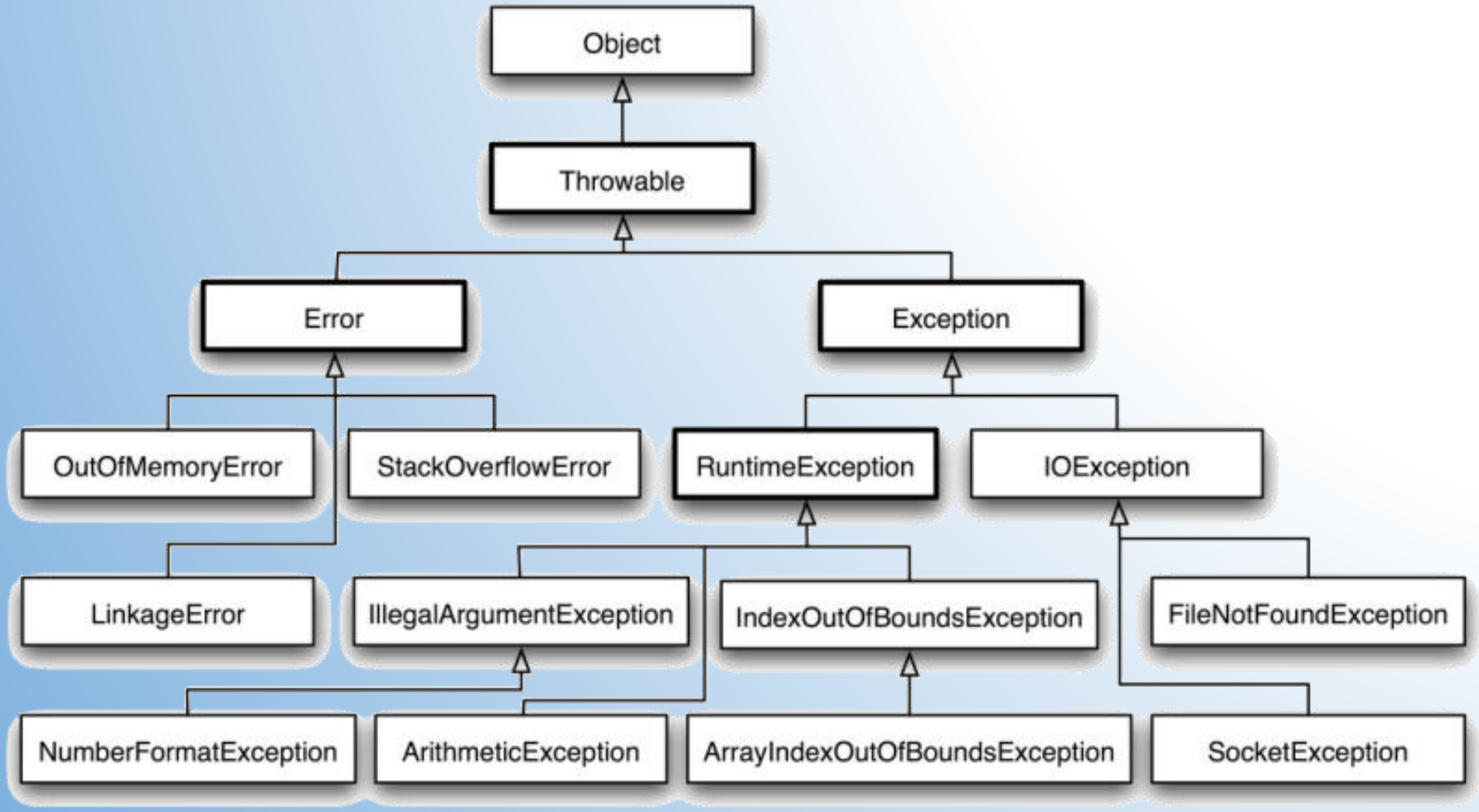
En la programación tradicional, el modo habitual de tratar situaciones de excepción era *devolver un valor especial*, poner un valor en una *variable global* o mediante un *parámetro ad hoc*. Pero esto tenía más de un inconveniente.

En primer lugar, los programadores clientes no siempre chequeaban estas variables o valores especiales (para evitar las *múltiples verificaciones anidadas*).

Por otro lado, chequear las condiciones de error deja el código básico **acoplado** con el que maneja los errores, haciendo que se pierda la claridad del programa.

La idea de trabajar con excepciones es *aislar el código que se usa para tratar problemas* del que podemos llamar código básico (que así queda más limpio).

Jerarquía de clases de excepciones en Java



Uso de excepciones en Java

Existen dos tipos de excepciones: verificadas (*checked*) y no verificadas (*unchecked*). Las excepciones verificadas extienden `Exception` (pero no `RuntimeException`) y, para poder compilar el programa, deben ser declaradas (con `throws`) o atrapadas (con `try..catch..finally`). Las excepciones no verificadas extienden `RuntimeException` o `Error`, y no es obligatorio declararlas ni atraparlas.

En el bloque `try` se encierra el código que podría provocar errores durante la ejecución y lanzar excepciones. En el o los bloque(s) `catch` se encierra el código que se ejecuta cuando se atrapa una excepción del tipo indicado en el parámetro. En el bloque `finally` (que es optativo cuando ya se definió, por lo menos, un bloque `catch`) se encierra el código que debe ejecutarse siempre, se haya o no lanzado y atrapado una excepción.

El bloque `try` debe preceder a, por lo menos, un bloque `catch` o al bloque `finally`.

Si se definen dos o más bloques `catch`, los bloques que atrapan las excepciones más específicas deben aparecer primero y los que atrapan las más generales deben aparecer por último.

```java
package excepciones;

public class AprobadoException extends Exception {

    public AprobadoException(String message) {
        super(message);
    }
}
```

```java
package excepciones;

public class ReprobadoException extends Exception {

    public ReprobadoException(String message) {
        super(message);
    }
}
```

```java
package excepciones;

import java.io.FileNotFoundException;

public class Alumno {

    private String nombre;
    private int nota;

    public Alumno(String nombre, int nota) {
        this.nombre = nombre;
        this.nota = nota;
    }

    public String getNombre() {
        return nombre;
    }

    public int getNota() {
        return nota;
    }

    public void sonreir() {
        int x = 1/0;
    }

    public void llorar() throws FileNotFoundException {
        java.io.FileReader fileReader = new java.io.FileReader("noexiste.fil");
    }
}
```

```java
package excepciones;

public class Main {

    public static void main(String[] args) {

        Alumno a1 = new Alumno("Pedro", );

        Alumno a2 = new Alumno("Pablo", );

        try {
            if (a1.getNota() >= 4) {
                throw new AprobadoException(a1.getNombre() + " ha aprobado!");
            } else {
                throw new ReprobadoException(a1.getNombre() + " fue reprobado!");
            }
        } catch (AprobadoException | ReprobadoException ex) {
            System.out.println(ex.getMessage());
        }

        if (a2.getNota() >= 4) {
            try {
                a2.sonreir();
            } catch (RuntimeException ex) {
                System.out.println("Sonrie " + a2.getNombre() + "...");
            } catch (Exception ex) {
                System.out.println("Se rie " + a2.getNombre() + "...");
            } finally {
                System.out.println("Porque se ha sacado un " + a2.getNota() + "!");
            }
        } else {
            try {
                a2.llorar();
            } catch (Exception ex) {
                System.out.println("Llora " + a2.getNombre() + "...");
            } finally {
                System.out.println("Porque le pusieron un " + a2.getNota() + "!");
            }
        }
    }
}
```

Ejemplo

El sistema está compuesto por las siguientes cuatro clases: `Alumno`, que posee dos atributos (`nombre` y `nota`) con sus correspondientes *getters* y dos comportamientos (`sonreir` y `llorar`), `AprobadoException` y `ReprobadoException`, que son clases de excepciones definidas *ad-hoc* y, por último, `Main`, donde se instancian dos alumnos `a1` y `a2`, y donde, además, según cómo sea la nota de `a1`, se lanza una de las dos excepciones anteriores y según cómo sea la nota de `a2`, se le envía a éste un mensaje `sonreir` o `llorar`.
Estos dos últimos métodos lanzan excepciones estándar de Java, las cuales son atrapadas en `Main`.

Ejercicio: Explique las partes marcadas 1-11.

https://github.com/dcorsi/algo3/tree/main/excepciones

Definición

Persistencia: Se denomina **persistencia** la capacidad de un objeto de trascender el tiempo o el espacio. Permite que un objeto sea usado en diferentes momentos, por el mismo programa o por otros, así como en diferentes instalaciones de hardware.

Un **objeto persistente** es aquel que conserva su estado en un medio de almacenamiento permanente, pudiendo ser reconstruido por el mismo u otro proceso, de modo tal que al reconstruirlo se encuentre en el mismo estado en que se lo guardó. Al objeto no persistente se lo denomina **efímero**.

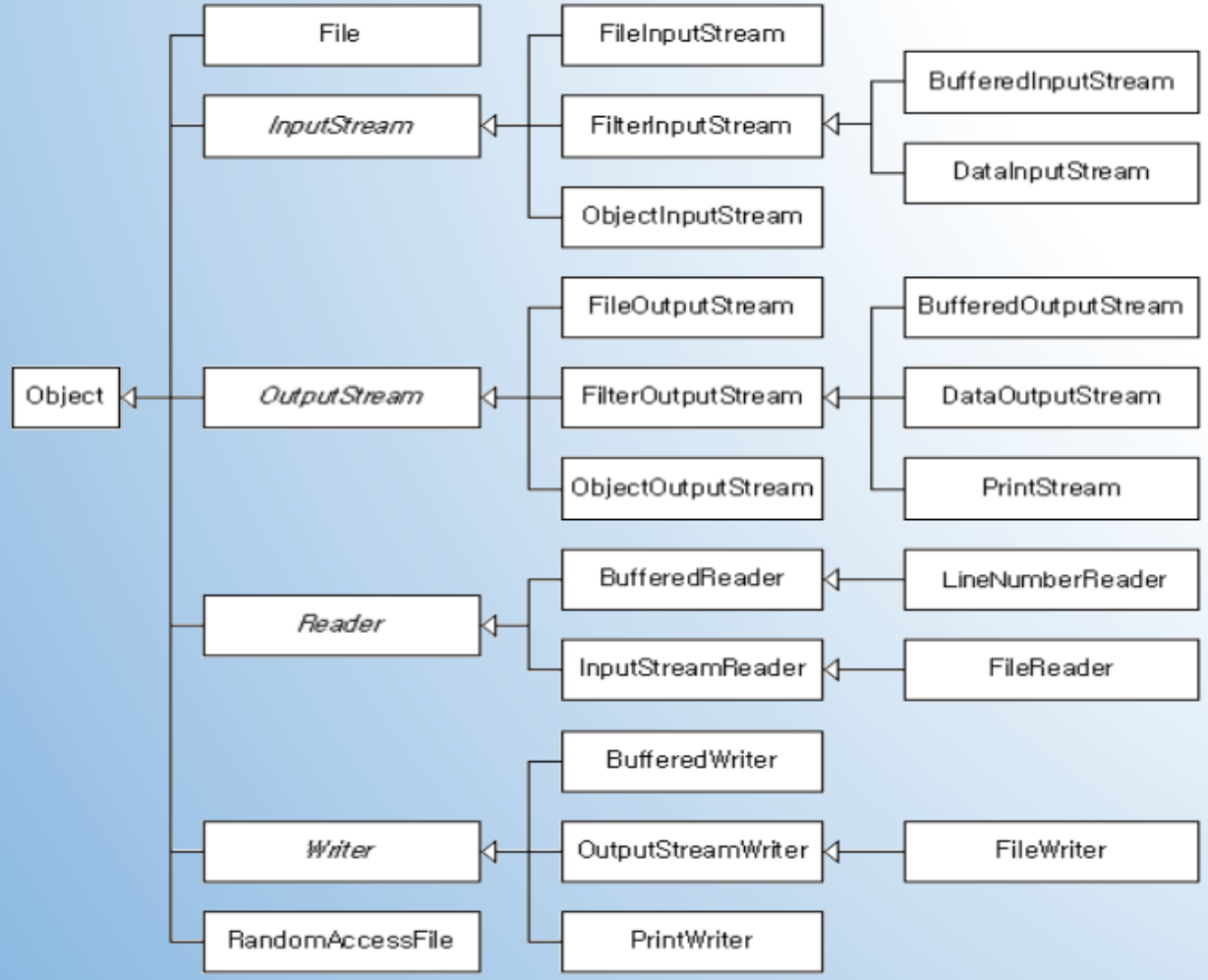
Manejo de archivos en Java

Java proporciona un amplio conjunto de clases e interfaces para manipular archivos.

El paquete `java.io` contiene numerosas clases (más de 50, por ejemplo: `File`), más de 10 interfaces (por ejemplo: `Serializable`), excepciones (más de 10, por ejemplo: `FileNotFoundException`) y un error (`IOError`).

Otros paquetes también proveen clases e interfaces que sirven para trabajar con archivos: `java.util.zip` contiene clases para trabajar con archivos comprimidos (por ejemplo: `ZipInputStream`), `javax.sound.sampled` contiene clases para trabajar con archivos de audio (por ejemplo: `AudioInputStream`), etc.

Jerarquía de clases de `java.io`



El patrón *Decorator* en `java.io`

Los diseñadores del lenguaje pensaron que utilizar la herencia para componer las combinaciones de funcionalidades más comunes requeridas para el manejo de archivos habría resultado en una verdadera *explosión de clases*.

Por eso, diseñaron las clases para el manejo de archivos siguiendo el patrón de diseño **Decorator**.

En lugar de usar la herencia para agregar funcionalidad a las clases, este patrón permite agregar funcionalidad a los objetos individuales (en tiempo de ejecución).

La clase `File`


```
package claseFile;
import java.io.File;
import java.text.DateFormat;
import java.util.Date;
import javax.swing.JFileChooser;

public class Main {
  public static void main(String[] args) {
    String nomElegido = "";
    JFileChooser fc = new JFileChooser("src/clasefile");
    fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
    if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
      nomElegido = fc.getSelectedFile().getPath();
    }
    File f = new File(nomElegido);
    System.out.println("Nombre: " + f.getName());
    System.out.println("Ubicado en: " + f.getParent());
    System.out.println("Última modificación: " +
                       DateFormat.getInstance().format(new Date(f.lastModified())));
    if (f.isFile()) {
      System.out.println("ES UN ARCHIVO");
      System.out.println("Tamaño: " + f.length() + " bytes");
    } else if (f.isDirectory()) {
      System.out.println("ES UN DIRECTORIO");
      System.out.println("Contenido:");
      int cantArchivos = 0;
      long tamArchivos = 0;
      int cantDirectorios = 0;
      for (File elemento : f.listFiles()) {
        System.out.print(DateFormat.getInstance().format(new Date(elemento.lastModified())));
        if (elemento.isFile()) {
          cantArchivos++;
          tamArchivos += elemento.length();
          System.out.printf("     %,14d  ", elemento.length());
        } else if (elemento.isDirectory()) {
                cantDirectorios++;
                System.out.printf("     <DIR>          ");
        }
        System.out.println(elemento.getName());
      }
      System.out.printf(" %,7d archivos  %,14d  bytes\n", cantArchivos, tamArchivos);
      System.out.printf(" %,7d dirs      %,14d  bytes libres\n",
                        cantDirectorios, f.getFreeSpace());
    }
  }
}
```

https://github.com/dcorsi/algo3/tree/main/claseFile

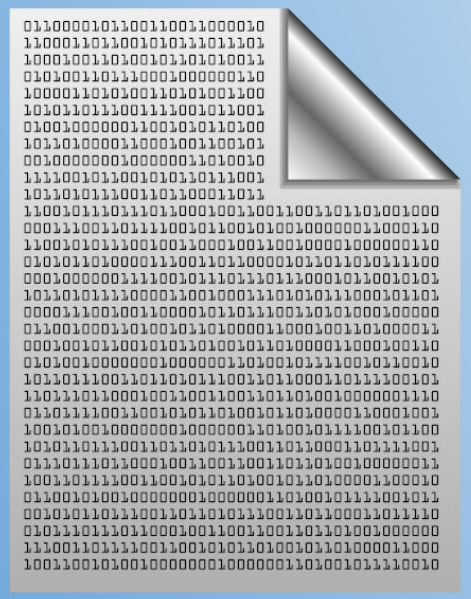
Java

Archivos de Texto Plano (Unicode)


```java
package archivosDeTexto;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.LineNumberReader;
import java.io.PrintWriter;
import javax.swing.JFileChooser;

public class Main {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        String nomArch = "";
        JFileChooser fc = new JFileChooser("src/archivosDeTexto");
        if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
            nomArch = fc.getSelectedFile().getPath();
        }

        FileReader fr = new FileReader(nomArch);
        BufferedReader br = new BufferedReader(fr);
        LineNumberReader lr = new LineNumberReader(br);

        FileWriter fw = new FileWriter(nomArch + ".bak");
        BufferedWriter bw = new BufferedWriter(fw);
        PrintWriter pw = new PrintWriter(bw);

        String renglon;
        while ((renglon = lr.readLine()) != null) {
            pw.println(lr.getLineNumber() + ") " + renglon);
        }

        pw.close();
    }
}
```

https://github.com/dcorsi/algo3/tree/main/archivosDeTexto

Archivos Binarios Secuenciales

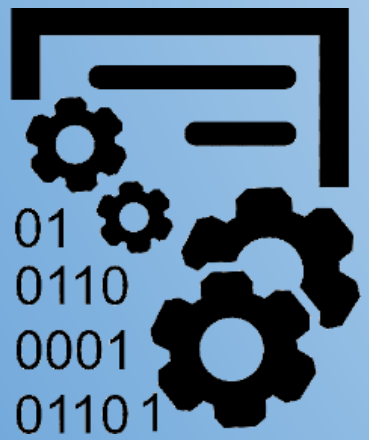


```java
package archivosBinariosSecuenciales;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.swing.JFileChooser;

public class Main {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        String nomArch = "";
        JFileChooser fc = new JFileChooser("src/archivosBinariosSecuenciales");
        if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
            nomArch = fc.getSelectedFile().getPath();
        }

        FileInputStream fis = new FileInputStream(nomArch);
        BufferedInputStream bis = new BufferedInputStream(fis);
        DataInputStream dis = new DataInputStream(bis);

        FileOutputStream fos = new FileOutputStream(nomArch + ".bak");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        DataOutputStream dos = new DataOutputStream(bos);

        int byteLeido;    // byte no permite contener 128..255. Por eso: int
        boolean finDeArchivo = false;
        while (!finDeArchivo) {
            try {
                byteLeido = dis.readByte();
                dos.writeByte(byteLeido);
            } catch (EOFException ex) {
                dos.close();
                finDeArchivo = true;
            }
        }
    }
}
```

https://github.com/dcorsi/algo3/tree/main/archivosBinariosSecuenciales

Java

Archivos Binarios de Acceso Aleatorio




```java
package archivosBinariosDeAccesoAleatorio;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import javax.swing.JFileChooser;

public class Main {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        String nomArch = "";
        JFileChooser fc = new JFileChooser("src/archivosBinariosDeAccesoAleatorio");
        if (fc.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
            nomArch = fc.getSelectedFile().getPath();
        }

        RandomAccessFile raf = new RandomAccessFile(nomArch, "rw");

        long tam = raf.length();
        for (long i = 0; i < tam / 2; i++) {
            raf.seek(i);
            int b1 = raf.read();
            raf.seek(tam - i - 1);
            int b2 = raf.read();
            raf.seek(i);
            raf.write(b2);
            raf.seek(tam - i - 1);
            raf.write(b1);
        }

        raf.close();
    }
}
```

Archivos de objetos (Serialización)

```
package serializacion;
import java.io.IOException;
public class Main {
  public static void main(String[] args) {
      Lista li = new Lista();
      try {
          li = li.deSerializar("lista.txt");
          if (!EntradaSalida.leerBoolean("Ya hay una lista. ¿Desea reutilizarla?")) {
              li = new Lista();
          }
      } catch (IOException | ClassNotFoundException e) {
          EntradaSalida.mostrarString("Lista nueva!");
      }
      do {
          String dato = EntradaSalida.leerString("Ingrese una cadena");
          li.agregar(dato);
      } while (EntradaSalida.leerBoolean("Desea seguir agregando valores?"));
      li.mostrar();
      try {
          li.serializar("lista.txt");
      } catch (Exception e) {
          EntradaSalida.mostrarString(e.getMessage() + "\nERROR AL GRABAR!");
      }
  }
}
```

```
package serializacion;
import java.io.*;
import java.util.ArrayList;
public class Lista implements Serializable {
  private ArrayList<String> valores;
  public Lista() {
    valores = new ArrayList<>();
  }
  public void agregar(String s) {
    valores.add(s);
  }
  public void mostrar() {
    String cadena = "La lista contiene:\n";
    for (String s : valores) {
        cadena += s + "\n";
    }
    EntradaSalida.mostrarString(cadena);
  }
  public Lista deSerializar(String nomArch) throws IOException,ClassNotFoundException {
    ObjectInputStream o =
        new ObjectInputStream(new BufferedInputStream(new FileInputStream(nomArch)));
    Lista j = (Lista) o.readObject();
    o.close();
    return j;
  }
  public void serializar(String nomArch) throws IOException {
    ObjectOutputStream o =
       new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(nomArch)));
    o.writeObject(this);
    o.close();
  }
}
```

```
package serializacion;

import javax.swing.JOptionPane;

public class EntradaSalida {

  public static String leerString(String texto) {
    String st = JOptionPane.showInputDialog(texto);
    return (st == null ? "" : st);
  }

  public static boolean leerBoolean(String texto) {
    int i=JOptionPane.showConfirmDialog(null,texto,"Consulta",JOptionPane.YES_NO_OPTION);
    return i == JOptionPane.YES_OPTION;
  }

  public static void mostrarString(String s) {
    JOptionPane.showMessageDialog(null, s);
  }
}
```

https://github.com/dcorsi/algo3/tree/main/serializacion