

Módulo Profesional 06:
Acceso a datos

Actividad UF1

CICLO FORMATIVO DE GRADO SUPERIOR EN

**DESARROLLO DE APLICACIONES
MULTIPLATAFORMA**

MODALIDAD ONLINE

Enrique Verea



Persistencia en ficheros de texto/binarios

Objetivos

Repasar la POO

Leer y escribir ficheros de texto (FileReader y FileWriter) y ficheros binarios (FileInputStream y FileOutputStream)

Competencias asociadas:

- Programación OO
- Leer/escribir ficheros de texto/binarios

Metodología

Entrega

Preparación individual

10 octubre 2023, en PDF + ZIP

Dedicación estimada

Documentos de referencia

8 horas

Resultados de aprendizaje

- Programación básica (Java)
- Acceder a ficheros
- Leer/escribir ficheros de texto
- Leer/escribir ficheros binarios

Criterios de evaluación

- Escribir y leer ficheros de texto y binarios usando POO
- Incluye comentarios en tu código para facilitar su mantenimiento en un futuro hipotético

Desarrollo de la actividad

¡IMPORTANTE!

Para cada uno de los ejercicios, se deberá:

- explicar el ejercicio para que quede claro qué está haciendo y porqué (escribirlo en este documento, que luego guardaréis como pdf)
- documentar con **capturas de pantalla** los pasos más significativos del funcionamiento del programa, intercalándoles con el punto anterior
- documentar los **resultados** obtenidos al finalizar la ejecución del programa
- incluir también **comentarios en el código** para explicar los pasos más importantes
- copiar el **código completo** (ponerlo al final de este documento, en un apartado llamado **Anexo**, indicando el número de ejercicio al que pertenece)

Para **presentar** la actividad:

- se deberán subir, en el único intento habilitado disponible, los **2 archivos**: 1 pdf y 1 zip
 - o el pdf incluirá este documento con toda la resolución de los ejercicios (con lo comentado anteriormente)
 - o el zip incluirá sólo los archivos zip (con el código de cada uno de los ejercicios). Es decir, el zip englobará a los otros zips (1 por actividad)
- Por favor, no pongáis el pdf dentro del zip 😊

PARTE 1 – Repaso a POO

Ejercicio1 – Crear la clase persona

Crea la clase **persona**, con sus constructores (vacío + el que le pases el nombre + el que le pases todos los campos). También crea todos los *getters*, *setters* y un mostrar todos los campos. Las propiedades serán:

- Nombre
- Apellido
- Ciudad
- Nacionalidad
- Edad

Cuando lo tengas, realiza un pequeño *main* para probarlo (donde tengas que informar de estos campos para 3 personas) e incluye las capturas de pantalla del resultado y el código para que quede bien documentada tu respuesta.

Recuerda que, en el ámbito profesional, tanto un código claro, modular y que cumpla los requerimientos funcionales, como una documentación completa son importantes para cualquier tipo de proyecto.

PARTE 2 – Persistencia en ficheros de Texto

En la siguiente práctica vamos a trabajar con ficheros de texto, ficheros binarios y objetos. A lo largo de los diferentes ejercicios se irá creando un menú.

FICHEROS DE TEXTO

Ejercicio2 - Almacenar personas en un fichero texto.

Vamos a utilizar 1 fichero de texto para almacenar, de forma secuencial, datos de 3 personas (Nombre, Apellido, Ciudad, Nacionalidad y Edad). Pueden ser diferentes del ejercicio anterior.

El programa sólo deberá pedir los datos de 3 personas (campo a campo) y guardarlos en un fichero de texto, usando las clases de Java vistas en esta UF para ficheros de texto. El nombre del fichero de texto y la ubicación de este serán los primeros parámetros que el usuario del programa deberá indicar.

En casos de que la ruta del fichero no exista, deberá saltar un mensaje de error indicando de que el *path* es inexistente. Si el fichero ya existe en esta ruta, deberá avisar al usuario si quiere o bien sobre escribirlo o bien añadir la información al final. Tener en cuenta todas estas casuísticas...

Por lo tanto, como decía, en caso de que el fichero ya exista, deberá añadir los datos de las 3 personas al final de este, para conservar los datos originales. Y avisar al usuario que se están añadiendo datos a un fichero ya existente. Si el fichero no existe, se avisará que el fichero se ha creado correctamente.

Utiliza la clase persona para recolectar los datos, y un fichero de texto para almacenarlos. Al acabar, muestra el archivo creado en tu *filesystem* y el contenido de este.

Recuerda cumplir con los requisitos informados en el punto “Desarrollo de la actividad”.

Ejercicio3 - Leer personas de un fichero de texto.

Vamos a ampliar ahora las opciones de este programa para que el usuario también pueda leer ficheros, usando las clases de Java vistas en esta UF para ficheros de texto.

Crea un menú para ampliar el programa anterior. El menú deberá tener 2 opciones: Escribir (que invocará el código del ejercicio anterior) y Leer, objetivo de este ejercicio.

Aunque vas a aprovechar el código del ejercicio anterior para la opción de Leer, debes hacerlo en un proyecto nuevo, para conservar tanto el código del ejercicio anterior tal y como lo tenías, como el de ahora, de forma independiente. En el ámbito profesional, es importante guardar los códigos en sus diferentes estatus, para poder usarlos en caso necesario (por esto existen los repositorios que conservan distintas versiones...)

Al seleccionar cualquiera de las opciones, el programa deberá preguntar al usuario la ubicación del fichero y el nombre (recuerda gestionar las excepciones en caso de que el *path* y/o el nombre no existan, este último caso si se quiere leer...).

A continuación, deberá mostrar 2 sub-opciones más: leer todo el archivo y leer una persona. En este caso, sólo deberá mostrar todos los datos de la persona que tenga el mismo Nombre que el indicado por el usuario. Si hay más usuarios con el mismo Nombre, los deberá mostrar también.

Recuerda cumplir con los requisitos informados en el punto "Desarrollo de la actividad".

PARTE 3 – Persistencia en ficheros Binarios

FICHEROS BINARIOS

Ejercicio4 - Almacenar personas en un fichero binario.

Amplía el programa anterior con una opción más en el menú. Esta nueva opción (ya es la tercera) será para que guarde (escribir) los datos de “N” personas en un fichero binario, usando las clases de Java vistas en esta UF para ficheros binarios.

“N” será un campo que el usuario decidirá/informará al ejecutarse esta opción, y siempre será menor o igual a 3. Habrá que informar que el máximo de usuarios a insertar en el archivo binario **de una sola vez** es de 3. Si ya había datos en el fichero, simplemente avisar al usuario que se van a sobrescribir (perder) los datos que había previamente en el fichero (no haremos *append* en binario).

Atención: en binario, debéis escribirlo todo siempre dentro una misma/sola conexión, para que luego se pueda leer sin problemas (cuando se cierra y se vuelve a abrir una conexión a un fichero binario, se generan unos bits adicionales entre los datos de las distintas conexiones que complican la lectura). Esto se podría gestionar, pero no lo vamos a pedir en este ejercicio.

Recuerda cumplir con los requisitos informados en el punto “Desarrollo de la actividad”.

Ejercicio5 - Leer personas de un fichero binario.

Amplía el programa anterior (opción 4: Leer fichero binario) para que, usando las clases de Java vistas en esta UF para ficheros binarios, nuestro programa permita recuperar los datos de una o varias personas y los muestre por pantalla. Ídem (misma funcionalidad requerida) que el ejercicio 3, pero ahora leer de un fichero binario, no de texto).

Recuerda cumplir con los requisitos informados en el punto “Desarrollo de la actividad”.

Anexo

Ejercicio 1

Clase Persona

JavaBean con atributos nombre, apellido, ciudad y nacionalidad de tipo *String* y edad de tipo *int*.

Sobreescribe el método *toString()* de la clase *Object* para producir una *String* con todos sus atributos y sus nombres.

Sobreescribe el método *equal()* de la clase *Object* comparar instancias de *Persona* entre sí.

Constructores

```

1  package ifp.kikeverea.persona;
2
3  import ifp.kikeverea.io.DatosNoContienenPersonasException;
4
5  import java.io.Serializable;
6
7  public class Persona implements Serializable {
8
9      private String nombre;
10     private String apellido;
11     private String ciudad;
12     private String nacionalidad;
13     private int edad;
14
15     /**
16      * Constructor por defecto
17      */
18     public Persona() {}
19
20     /**
21      * Constructor con el parámetro "nombre"
22      * @param nombre el nombre de la persona
23      */
24     public Persona(String nombre) { this.nombre = nombre; }
25
26     /**
27      * Constructor con los parámetros: nombre, apellido, ciudad, nacionalidad, edad
28      * @param nombre el nombre de la persona
29      * @param apellido el apellido de la persona
30      * @param ciudad la ciudad de nacimiento de la persona
31      * @param nacionalidad la nacionalidad de la persona
32      * @param edad la edad en años de la persona
33      */
34     public Persona(String nombre, String apellido, String ciudad, String nacionalidad, int edad) {
35         this.nombre = nombre;
36         this.apellido = apellido;
37         this.ciudad = ciudad;
38         this.nacionalidad = nacionalidad;
39         this.edad = edad;
40     }
41 }
42
43 
```

Getters and Setters

```

41
42 public String getNombre() { return nombre; }
45
46 public void setNombre(String nombre) { this.nombre = nombre; }
49
50 public String getApellido() { return apellido; }
53
54 public void setApellido(String apellido) { this.apellido = apellido; }
57
58 public String getCiudad() { return ciudad; }
61
62 public void setCiudad(String ciudad) { this.ciudad = ciudad; }
65
66 public String getNacionalidad() { return nacionalidad; }
69
70 public void setNacionalidad(String nacionalidad) { this.nacionalidad = nacionalidad; }
73
74 public int getEdad() { return edad; }
77
78 public void setEdad(int edad) { this.edad = edad; }

```

toString()

```

95
96 /**
97  * Convierte esta Persona en String
98  * @return Una String con los atributos de esta Persona
99  */
100 @Override
101 public String toString() {
102     return "Nombre: " + nombre + ", " +
103           "Apellido: " + apellido + ", " +
104           "Ciudad: " + ciudad + ", " +
105           "Nacionalidad: " + nacionalidad + ", " +
106           "Edad: " + edad + " ";
107 }
108

```

main()

Se crean 3 instancias de persona y se pasan como argumento al método `println()` de `System.out`, concatenadas a una `String` que indica el número de persona. El método `println()` invoca automáticamente el método `toString()` de todos los objetos que son pasados como argumento. El método `toString()` de `Persona` produce una `String` con todos sus parámetros.

```

1  package ifp.kikeverea.main;
2
3  import ifp.kikeverea.persona.Persona;
4
5  public class Ejercicio1 {
6  public static void main(String[] args) {
7      Persona persona1 = new Persona("Elsa", "Pato", "Zapata", "España", 34);
8      Persona persona2 = new Persona("Susana", "Oria", "California", "Mexico", 25);
9      Persona persona3 = new Persona("Elmer", "Cado", "Marrakech", "Marruecos", 29);
10
11     System.out.println("Informe:");
12     System.out.println("Persona 1: " + persona1);
13     System.out.println("Persona 2: " + persona2);
14     System.out.println("Persona 3: " + persona3);
15 }
16 }

```

Run: Ejercicio1 x

```

/home/kike/jdks/corretto-11/bin/java -javaagent:/home/kike/bin/idea-IC-222.4167.29/lib/idea_rt.jar=41235:/t
Informe:
Persona 1: Nombre: 'Elsa', Apellido: 'Pato', Ciudad: 'Zapata', Nacionalidad: 'España', Edad: '34'
Persona 2: Nombre: 'Susana', Apellido: 'Oria', Ciudad: 'California', Nacionalidad: 'Mexico', Edad: '25'
Persona 3: Nombre: 'Elmer', Apellido: 'Cado', Ciudad: 'Marrakech', Nacionalidad: 'Marruecos', Edad: '29'

Process finished with exit code 0

```

Todos los Ejercicios - Clases *util*

Clase InputUsuario

Clase que facilita la obtención y validación de datos del usuario.

```
public class InputUsuario {

    private final Scanner scanner;

    public InputUsuario(Scanner scanner) { this.scanner = scanner; }

    /**
     * Solicita una entrada de texto al usuario
     * @param mensaje El mensaje que se imprime al usuario al pedir la entrada
     * @return El texto proporcionado por el usuario
     */
    public String solicitarTexto(String mensaje) {
        System.out.print(mensaje);
        return scanner.nextLine();
    }

    /**
     * Solicita una entrada de entero al usuario
     * @param mensaje El mensaje que se imprime al usuario al pedir la entrada
     * @return El entero proporcionado por el usuario
     */
    public int solicitarEntero(String mensaje) {
        return solicitarEntero(mensaje, ValidadorNumeros.sinValidacion());
    }

    /**
     * Solicita una entrada de entero al usuario, hasta que ésta sea válida
     * @param mensaje El mensaje que se imprime al usuario al pedir la entrada
     * @param validador el objeto que dará validez a la entrada del usuario
     * @return El entero proporcionado por el usuario, una vez validado
     */
    public int solicitarEntero(String mensaje, ValidadorNumeros validador) {
        while (true) {
            try {
                // imprime el mensaje al usuario en pantalla y lee el siguiente entero introducido
                System.out.print(mensaje);
                int entero = scanner.nextInt();

                // consume el resto de la línea
                scanner.nextLine();

                // comprueba validez
                if (!validador.validarNumero(entero)) {
                    System.out.println(validador.mensajeError());
                    continue;
                }

                return entero;
            }
            catch (InputMismatchException e) {
                resolverInputInvalido("Por favor, introducir un número entero.");
            }
        }
    }
}
```

Interfaz ValidadorNumeros

Interfaz que facilita la validación de números. Expone métodos de fábrica estáticos para la creación de objetos de clases que implementan esta interfaz.

```
public interface ValidadorNumeros {

    boolean validarNumero(double numero);
    String mensajeError();

    static ValidadorNumeros soloPositivos() { return new ValidadorPositivos(); }
    static ValidadorNumeros sinValidacion() { return new NoValidador(); }
    static ValidadorNumeros enIntervalo(int start, int end) { return new ValidadorEnIntervalo(start, end); }

    /**
     * Clase que valida números solo si son positivos
     */
    class ValidadorPositivos implements ValidadorNumeros {
        @Override
        public boolean validarNumero(double numero) { return numero >= 0; }

        @Override
        public String mensajeError() { return "Por favor, introducir un número positivo"; }
    }

    /**
     * Clase que valida números solo si se encuentran dentro del intervalo establecido
     */
    class ValidadorEnIntervalo implements ValidadorNumeros {

        private final int start;
        private final int end;

        public ValidadorEnIntervalo(int start, int end) {
            this.start = start;
            this.end = end;
        }

        @Override
        public boolean validarNumero(double numero) { return numero >= start && numero <= end; }

        @Override
        public String mensajeError() { return "Por favor, introducir un número entre " + start + " y " + end; }
    }
}
```

Interfaz IOFichero

Interfaz para la entrada y salida de colecciones de objetos en ficheros (*File*). Consiste en dos métodos de lectura (con y sin filtro) y dos métodos de escritura (con y sin opción a *append*).

```
public interface IOFichero<T> {
    Collection<T> leerContenido(File fichero) throws IOException, DatosNoContienenPersonasException;
    Collection<T> leerContenido(File fichero, FiltroLectura<Persona> filtro) throws IOException, DatosNoContienenPersonasException;
    void escribirEnFichero(File fichero, Collection<T> objects) throws IOException;
    void escribirEnFichero(File fichero, Collection<T> objects, boolean anadir) throws IOException;
}
```

Clase utility ProgramaProveedorFicheros

Programa que genera instancias de FicheroPersona. La ruta del fichero se pide al usuario y es validada por FicheroPersona. El programa también valida si el fichero existe, de ser necesario. Si la ruta es inválida, se seguirá pidiendo una ruta al usuario hasta que éste proporcione una ruta válida.

```

1  package ifp.kikeverea.main.programas;
2
3  import ifp.kikeverea.io.FicheroPersonas;
4  import ifp.kikeverea.io.IOFichero;
5  import ifp.kikeverea.persona.Persona;
6  import ifp.kikeverea.util.InputUsuario;
7
8  public class ProgramaProveedorFicheros {
9
10 @
11     public static FicheroPersonas solicitarFichero(IOFichero<Persona> io, InputUsuario input) {
12         FicheroPersonas fichero = new FicheroPersonas(io);
13         establecerRutaDelFichero(input, fichero);
14
15         return fichero;
16     }
17
18 @
19     public static FicheroPersonas solicitarFicheroExistente(IOFichero<Persona> io, InputUsuario input) {
20         do {
21             FicheroPersonas fichero = solicitarFichero(io, input);
22
23             if (fichero.existe())
24                 return fichero;
25
26             else System.out.println("El fichero no existe");
27         }
28         while (true);
29     }
30
31 @
32     private static void establecerRutaDelFichero(InputUsuario input, FicheroPersonas fichero) {
33         String ruta;
34         do {
35             ruta = input.solicitarTexto("Ruta y/o nombre del fichero: ");
36             ruta = ruta.strip(); // elimina los espacios en blanco al principio y final del String
37         }
38         while (!fichero.establecerRuta(ruta));
39     }
40 }

```

Ejercicio 2

Clase FicheroPersona

Recibe una instancia de IOFichero con tipo de parámetro Persona, a la que delega la responsabilidad de escritura del fichero.

Expone el método *establecerRuta(String)*, que valida la ruta recibida y genera la instancia de *File* con la que trabajará esta clase.

Expone el método *anadirPersona(Persona)*, que añade objetos de Persona al buffer interno de esta clase.

```
/**
 * Clase que encapsula la creación y validación de ficheros y facilita la lectura y escritura de instancias de Persona
 * a través de la interfaz IOFichero.
 */
public class FicheroPersonas {

    private File fichero;
    private final List<Persona> buffer = new ArrayList<>(3);
    private final IOFichero<Persona> io;

    /**
     * Constructor
     * @param io Clase encargada de la entrada y salida de datos del fichero
     */
    public FicheroPersonas(IOFichero<Persona> io) { this.io = io; }

    /**
     * Valida y establece la ruta recibida como la ruta con la que trabajará esta clase
     * @param ruta La ruta a validar y establecer
     * @return true si la ruta es válida, false si es inválida
     */
    public boolean establecerRuta(String ruta) {
        fichero = new File(ruta);

        if (!rutaDeDirectorioValida()) {
            System.out.println("La ruta del directorio de trabajo no existe");
            return false;
        }

        return true;
    }

    private boolean rutaDeDirectorioValida() {
        String rutaDirectorioDeTrabajo = fichero.getParent();
        return rutaDirectorioDeTrabajo == null || new File(rutaDirectorioDeTrabajo).exists();
    }
}
```

```
/**
 * Añade una persona al buffer de esta clase
 * @param persona Persona que se añade al buffer
 */
public void añadirPersona(Persona persona) { buffer.add(persona); }

/**
 * Escribe el contenido del buffer en la ruta establecida
 * @param anadir Si es true, añade el contenido al final del fichero, si no, sobrescribe el contenido del fichero
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 */
public void escribirFichero(boolean anadir) throws IOException {
    io.escribirEnFichero(fichero, buffer, anadir);
}
```

Clase IOFicheroTextoPersona

Implementa la interfaz IOFichero. Encapsula la escritura de instancias de Persona en ficheros de texto. El texto a escribir es producido por el método *toString()* de la clase Persona.

```
/**
 * Escribe contenido a un fichero. Si el fichero ya tenía contenido, este será eliminado en el proceso de escritura
 * @param fichero El fichero en el que escribirá el contenido
 * @param personas Las Personas a escribir en el fichero
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 */
@Override
public void escribirEnFichero(File fichero, Collection<Persona> personas) throws IOException {
    escribirEnFichero(fichero, personas, false);
}

/**
 * Escribe contenido a un fichero
 * @param fichero El fichero en el que escribirá el contenido
 * @param personas Las Personas a escribir en el fichero
 * @param anadir Si es true, añade el contenido al final del archivo. Si es false los datos existentes en el
 * fichero serán eliminados durante la escritura
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 */
@Override
public void escribirEnFichero(File fichero, Collection<Persona> personas, boolean anadir) throws IOException {
    try (FileOutputStream fos = new FileOutputStream(fichero, anadir);
        OutputStreamWriter sw = new OutputStreamWriter(fos);
        BufferedWriter writer = new BufferedWriter(sw)) // Crea un nuevo escritor con buffer para el fichero
    {
        for (Persona persona : personas) {
            writer.write(persona.toString()); // escribe una String con los atributos de Persona en el fichero
            writer.newLine(); // escribe un salto de línea en el fichero
        }
    }
}
```


Enum AccionEscritura

Enum que encapsula el número y nombre de la acción de escritura a presentar en el menú del programa de escritura, y el mensaje final a mostrar al usuario al terminar la acción.

```
public enum AccionEscritura {
    SOBREESCRIBIR_FICHERO(1, "Sobreescribir fichero", "Datos escritos al fichero"),
    ANADIR_A_FICHERO(2, "Añadir contenido al final", "Datos añadidos al final del fichero"),
    CREAR_FICHERO(3, "Crear fichero", "Fichero creado correctamente"),
    CANCELAR(0, "Cancelar", "Programa finalizado");

    private final int numero;
    private final String nombre;
    private final String mensajeFinal;

    AccionEscritura(int numero, String nombre, String mensajeFinal) {
        this.numero = numero;
        this.nombre = nombre;
        this.mensajeFinal = mensajeFinal;
    }

    public int numero() { return numero; }

    public String nombre() { return nombre; }

    public String mensajeFinal() { return mensajeFinal; }

    /OptionalGetWithoutIsPresent/
    static AccionEscritura determinarAccion(int numero) {
        return Arrays.stream(AccionEscritura.values()) Stream<AccionEscritura>
            .filter(accionEscritura -> accionEscritura.numero == numero)
            .findFirst() Optional<AccionEscritura>
            .get();
    }
}
```

Clase utility ProgramaEscritura

Programa encargado de coordinar la escritura de instancias de Persona en ficheros de texto con los inputs al usuario. Delega todas las responsabilidades a las clases expuestas anteriormente.

Ejecución: determina el tipo de acción de escritura según si el fichero existe o según la elección del usuario. Si la acción es CANCELAR, termina el programa. Si no, solicita un número de Personas determinado por el usuario (máximo 3). Si la acción es ANADIR_A_FICHERO, añade las Personas al final del fichero. Si es SOBREESCRIBIR_FICHERO, elimina el contenido del fichero al escribir los nuevos datos. Termina el programa imprimiendo el *mensaje final* de la acción.

```

public class ProgramaEscritura {

    private static final int MAX_PERSONAS = 3;

    private static final String MENU_ACCION_ESCRITURA =
        "El fichero ya existe. Elija una acción:\n" +
        SOBREESCRIBIR_FICHERO.numero() + "- " + SOBREESCRIBIR_FICHERO.nombre() + "\n" +
        ANADIR_A_FICHERO.numero() + "- " + ANADIR_A_FICHERO.nombre() + "\n" +
        CANCELAR.numero() + "- " + CANCELAR.nombre() + "\n" +
        "Acción: ";

    public static void ejecutar(FicheroPersonas fichero, InputUsuario input) {
        AccionEscritura accion = solicitarAccionEscritura(fichero, input);

        if (accion == CANCELAR) {
            System.out.println(CANCELAR.mensajeFinal());
            System.exit(0); // termina el programa
        }

        int numeroDePersonas = input.solicitarEntero(
            "Número de personas que desea escribir (máx. " + MAX_PERSONAS + "): ",
            ValidadorNumeros.enIntervalo(1, MAX_PERSONAS)
        );

        solicitarPersonas(numeroDePersonas, input, fichero);

        try {
            boolean anadirAlFinal = accion == ANADIR_A_FICHERO;
            fichero.escribirFichero(anadirAlFinal);
            System.out.println(accion.mensajeFinal());
        }
        catch (IOException e) {
            System.out.println("Error: no se ha podido escribir datos en el fichero");
            System.out.println("Causa: " + e.getMessage());
        }
    }

    private static AccionEscritura solicitarAccionEscritura(FicheroPersonas fichero, InputUsuario input) {
        if (fichero.existe())
            return CREAM_FICHERO;

        ValidadorNumeros validadorAccion = ValidadorNumeros.enIntervalo(0,2);
        int numeroAccion = input.solicitarEntero(MENU_ACCION_ESCRITURA, validadorAccion);
        return AccionEscritura.determinarAccion(numeroAccion);
    }

    private static void solicitarPersonas(int numeroDePersonas, InputUsuario input, FicheroPersonas ficheroPersonas) {
        for (int i = 0; i < numeroDePersonas; i++) {
            System.out.println("***** Persona" + (numeroDePersonas > 1 ? " " + (i + 1) : "") + " *****");
            String nombre = input.solicitarTexto("Nombre: ");
            String apellido = input.solicitarTexto("Apellido: ");
            String ciudad = input.solicitarTexto("Ciudad: ");
            String nacionalidad = input.solicitarTexto("Nacionalidad: ");
            int edad = input.solicitarEntero("Edad: ");
            ficheroPersonas.anadirPersona(new Persona(nombre, apellido, ciudad, nacionalidad, edad));
        }
    }
}

```

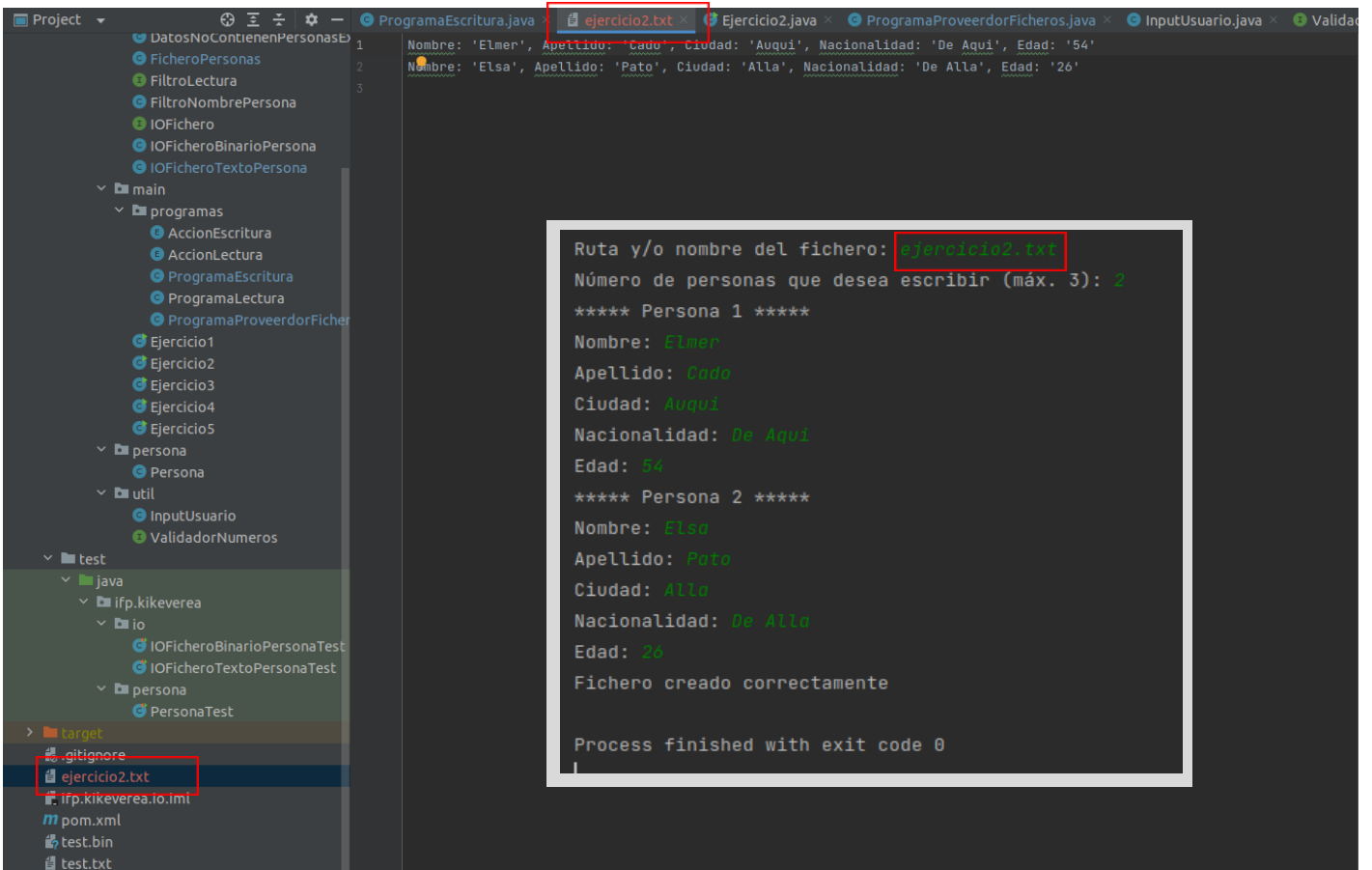
main()

Instancia un Scanner con la entrada estándar del sistema y un InputUsuario con este objeto de Scanner. Genera una instancia de FicheroPersonas mediante el ProgramaProveedorFicheros. Ejecuta el ProgramaEscritura, con la instancia de FicheroPersonas y de InputUsuario.

```
public class Ejercicio2 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        InputUsuario input = new InputUsuario(scanner);

        FicheroPersonas fichero = ProgramaProveedorFicheros.solicitarFichero(new IOFicheroTextoPersona(), input);
        ProgramaEscritura.ejecutar(fichero, input);
    }
}
```



The screenshot shows an IDE with the following components:

- Project Explorer (Left):** Displays the project structure. The `ejercicio2.txt` file is highlighted in the `target` directory.
- Editor (Top Right):** Shows the `Ejercicio2.java` file with the `main` method. The input data is visible: `Nombre: 'Elmer', Apellido: 'Cado', Ciudad: 'Aguil', Nacionalidad: 'De Agui', Edad: '54'` and `Nombre: 'Elsa', Apellido: 'Pato', Ciudad: 'Alla', Nacionalidad: 'De Alla', Edad: '26'`.
- Output Console (Bottom Right):** Displays the execution output of the program. The file path `ejercicio2.txt` is highlighted in red.

Execution Output:

```
Ruta y/o nombre del fichero: ejercicio2.txt
Número de personas que desea escribir (máx. 3): 2
***** Persona 1 *****
Nombre: Elmer
Apellido: Cado
Ciudad: Aguil
Nacionalidad: De Agui
Edad: 54
***** Persona 2 *****
Nombre: Elsa
Apellido: Pato
Ciudad: Alla
Nacionalidad: De Alla
Edad: 26
Fichero creado correctamente

Process finished with exit code 0
```

Ejercicio 3

Clase Persona

La clase Persona también expone un método *fromString(String)* que reconstruye una instancia de Persona a partir de una *String*. Si el formato de la *String* no corresponde al formato que produce *toString()*, se produce una excepción tipo *DatosNoContienenPersonasException*.

```
/**
 * Convierte una String en una Persona. La operación es el opuesto simétrico a {@link #toString() toString}
 * @param raw String con la info para crear una nueva persona. Asume que su formato viene dado por {@link #toString() toString}
 * @return Una nueva persona con los atributos encontrados en la String 'raw', o null si el formato de la String es inválido
 * @throws DatosNoContienenPersonasException Si el formato de la String no corresponde al formato que produce 'toString()'
 */
public static Persona fromString(String raw) throws DatosNoContienenPersonasException {
    if (!formatoValido(raw))
        throw new DatosNoContienenPersonasException("El formato de la String no se corresponde al formato " +
            "que produce 'toString()'. No se ha podido extraer ninguna instancia de Persona");

    String[] atributos = raw.split(",");

    return new Persona(
        extraerAtributo("Nombre", atributos[0]),
        extraerAtributo("Apellido", atributos[1]),
        extraerAtributo("Ciudad", atributos[2]),
        extraerAtributo("Nacionalidad", atributos[3]),
        Integer.parseInt(extraerAtributo("Edad", atributos[4]))
    );
}

private static boolean formatoValido(String raw) { return FORMATO.matcher(raw).matches(); }

private static String extraerAtributo(String atributo, String raw) {
    // longitud del atributo + los 3 caracteres siguientes (:)
    int longitudNombreAtributo = atributo.length() + 3;

    return raw.substring(longitudNombreAtributo, raw.length() - 1);
}

private static final Pattern FORMATO = Pattern.compile(
    "^Nombre: '.,+', " +
    "Apellido: '.,+', " +
    "Ciudad: '.,+', " +
    "Nacionalidad: '.,+', " +
    "Edad: '[0-9]+'$");
```

Clase FicheroPersona

La clase FicheroPersona también encapsula operaciones de lectura de ficheros, que delega a su instancia miembro de IOFichero.

```
/**
 * Lee el contenido de Personas del fichero en la ruta establecida
 * @return Una colección de Personas que representan el contenido del fichero
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 * @throws DatosNoContienenPersonasException si el fichero no contiene personas
 */
public Collection<Persona> leerFichero() throws IOException, DatosNoContienenPersonasException {
    return io.leerContenido(fichero);
}

/**
 * Lee el contenido de Personas con un nombre determinado del fichero en la ruta establecida
 * @param nombre El nombre de la(s) Persona(s) que se quiere obtener
 * @return Una colección de Personas con el nombre proporcionado
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 * @throws DatosNoContienenPersonasException si el fichero no contiene personas
 */
public Collection<Persona> leerConNombre(String nombre) throws IOException, DatosNoContienenPersonasException {
    return io.leerContenido(fichero, new FiltroNombrePersona(nombre));
}
}
```

Clase FiltroNombrePersona

Implementa la *interfaz funcional* FiltroLectura y es la clase utilizada para filtrar las lecturas de ficheros según un nombre de Persona.

```
/**
 * Clase que prueba si una persona tiene un determinado nombre
 */
public class FiltroNombrePersona implements FiltroLectura<Persona> {

    private final String nombre;

    public FiltroNombrePersona(String nombre) { this.nombre = nombre; }

    /**
     * Indica el nombre de la Persona y el nombre con que fue instanciada esta clase son iguales
     * @param persona La Persona cuyo nombre se va a revisar
     * @return true si los nombres coinciden, de lo contrario false
     */
    @Override
    public boolean pasaFiltro(Persona persona) { return persona.getNombre().equals(nombre); }
}
```

Clase IOFicheroTextoPersona

Esta clase también encapsula la lectura de instancias de Persona en ficheros de texto.

```

* Lee objetos tipo Persona de un fichero de texto. Asume que el fichero existe. Asume que el formato del texto
* es igual al formato utilizado por los métodos de escritura de esta clase
* @param fichero Fichero del cual se lee los objetos de Persona
* @return Colección de Personas contenidas en el fichero
* @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
* @throws DatosNoContienenPersonasException Si el fichero contiene instancias de Persona
*/
public Collection<Persona> leerContenido(File fichero) throws IOException, DatosNoContienenPersonasException {
    return leerContenido(fichero, null);
}

/**
* Lee objetos tipo Persona de un fichero de texto. Asume que el fichero existe. Asume que el formato del texto
* es igual al formato utilizado por los métodos de escritura de esta clase
* @param fichero Fichero del cual se lee los objetos de Persona
* @param filtro Filtro que se aplica a la lectura del fichero
* @return Colección de Personas contenidas en el fichero, o null si el formato del contenido es incorrecto
* @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
* @throws DatosNoContienenPersonasException Si el fichero contiene instancias de Persona
*/
public Collection<Persona> leerContenido(File fichero, FiltroLectura<Persona> filtro) throws IOException,
    DatosNoContienenPersonasException
{
    List<Persona> personas = new ArrayList<>();

    // Crea un nuevo flujo de entrada (BufferedReader)
    try (FileInputStream fis = new FileInputStream(fichero);
        InputStreamReader is = new InputStreamReader(fis);
        BufferedReader reader = new BufferedReader(is))
    {
        String linea;
        while ((linea = reader.readLine()) != null) { // itera hasta llegar al final del archivo (linea == null)

            Persona persona = Persona.fromString(linea); // reconstruye una Persona a partir de la línea

            if (filtro == null || filtro.pasaFiltro(persona))
                personas.add(persona);
        }
    }

    return personas;
}

```

Enum AccionLectura

Encapsula el número de la acción de lectura a presentar en el menú del programa de lectura.

```

public enum AccionLectura {
    LEER_TODO(1),
    LEER_NOMBRE(2);

    private final int numero;

    AccionLectura(int numero) { this.numero = numero; }

    public static AccionLectura determinarAccion(int numero) {
        return numero == LEER_TODO.numero ? LEER_TODO : LEER_NOMBRE;
    }
}

```

Clase utility ProgramaLectura

Programa encargado de coordinar la lectura de instancias de Persona de ficheros de texto. Delega todas las responsabilidades a las clases expuestas anteriormente.

Ejecución: solicita el tipo de acción deseada al usuario. Si la acción es *LEER_TODO* imprime todo el contenido del fichero. Si es *LEER_NOMBRE*, solicita un nombre al usuario con el que se filtra la lectura del fichero y luego imprime en pantalla las Personas cuyo nombre coincide con el filtro.

```
public class ProgramaLectura {

    private static final String MENU_ACCION_LECTURA =
        "Lectura. Elije una acción de lectura:\n" +
        "1- Leer todo el archivo:\n" +
        "2- Leer una persona:\n" +
        "Acción: ";

    public static void ejecutar(FicheroPersonas fichero, InputUsuario input) {
        AccionLectura accion = determinarAccionLectura(input);

        try {
            if (accion == AccionLectura.LEER_TODO) {
                imprimirPersonas(fichero.leerFichero());
            }
            else {
                String nombre = input.solicitarTexto("Nombre de la persona: ");
                imprimirPersonas(fichero.leerConNombre(nombre));
            }
        }
        catch (DatosNoContienenPersonasException | IOException e) {
            System.out.println("Error: no se ha podido leer el fichero");
            System.out.println("Causa: " + e.getMessage());
        }
    }

    private static AccionLectura determinarAccionLectura(InputUsuario input) {
        ValidadorNumeros validadorAccion = ValidadorNumeros.enIntervalo(1,2);
        int numeroAccion = input.solicitarEntero(MENU_ACCION_LECTURA, validadorAccion);

        return AccionLectura.determinarAccion(numeroAccion);
    }

    private static void imprimirPersonas(Collection<Persona> personas) {
        System.out.println("Personas:");
        personas.forEach(System.out::println);
    }
}
```

main()

Instancia InputUsuario al igual que el ejercicio anterior. Solicita una opción de entrada/salida al usuario. Si la opción es ESCRIBIR, genera un FicheroPersonas y ejecuta el ProgramaEscritura. Si la opción es LEER, genera un FicheroPersonas **existente** y ejecuta el ProgramaLectura. Si es SALIR, termina el programa. Los FicheroPersonas se crean con instancias de **IOFicheroTextoPersona**.

```
public class Ejercicio3 {

    private static final int ESCRIBIR = 1;
    private static final int LEER = 2;
    private static final int SALIR = 0;

    private static final String MENU =
        "Entrada/Salida de datos. Elige una opción:\n" +
        "1- Escribir\n" +
        "2- Leer\n" +
        "0- Salir\n" +
        "Opción: ";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        InputUsuario input = new InputUsuario(scanner);

        int programa = input.solicitarEntero(MENU, ValidadorNumeros.enIntervalo(0, 2));
        FicheroPersonas fichero;

        switch (programa) {
            case ESCRIBIR:
                fichero = ProgramaProveedorFicheros.solicitarFichero(new IOFicheroTextoPersona(), input);
                ProgramaEscritura.ejecutar(fichero, input);
                break;
            case LEER :
                fichero = ProgramaProveedorFicheros.solicitarFicheroExistente(new IOFicheroTextoPersona(), input);
                ProgramaLectura.ejecutar(fichero, input);
                break;
            case SALIR:
                System.out.println("Programa finalizado");
                System.exit(0);
                break;
        }
    }
}
```

```
1- Escribir
2- Leer
0- Salir
Opción: 2
Ruta y/o nombre del fichero: ejercicio2.txt
Elige una acción de lectura:
1- Leer todo el archivo:
2- Leer una persona:
Acción: 1
Personas:
Nombre: 'Elmer', Apellido: 'Cado', Ciudad: 'Auqui', Nacionalidad: 'De Aqui', Edad: '54'
Nombre: 'Elsa', Apellido: 'Pato', Ciudad: 'Alla', Nacionalidad: 'De Alla', Edad: '26'

Process finished with exit code 0
```


*** 2 personas fueron añadidas manualmente al archivo de texto para esta demostración*

```
ejercicio2.txt x ProgramaLectura.java x Ejercicio3.java x ProgramaEscritura.java x
1 Nombre: 'Elmer', Apellido: 'Cado', Ciudad: 'Auqui', Nacionalidad: 'De Aqui', Edad: '54'
2 Nombre: 'Elsa', Apellido: 'Pato', Ciudad: 'Alla', Nacionalidad: 'De Alla', Edad: '26'
3 Nombre: 'Elmer', Apellido: 'Curio', Ciudad: 'Espacio', Nacionalidad: 'Alien', Edad: '800'
4 Nombre: 'Elsa', Apellido: 'Pito', Ciudad: 'Charco', Nacionalidad: 'Teria', Edad: '3'
```

```
1- Escribir
2- Leer
0- Salir
Opción: 2
Ruta y/o nombre del fichero: ejercicio2.txt
Elije una acción de lectura:
1- Leer todo el archivo:
2- Leer una persona:
Acción: 2
Nombre de la persona: Elmer
Personas:
Nombre: 'Elmer', Apellido: 'Cado', Ciudad: 'Auqui', Nacionalidad: 'De Aqui', Edad: '54'
Nombre: 'Elmer', Apellido: 'Curio', Ciudad: 'Espacio', Nacionalidad: 'Alien', Edad: '800'

Process finished with exit code 0
```

Ejercicio 4

Clase Persona

La clase Persona implementa la interfaz Serializable para poder ser serializada y escrita en binario.

```
public class Persona implements Serializable {
```

Clase IOFicheroBinarioPersona

Implementa la interfaz IOFichero. Encapsula la escritura de instancias de Persona en ficheros binarios. No da soporte a escrituras al final del fichero (*append*).

```
/**
 * Escribe instancias de Persona en un fichero binario
 * @param fichero Fichero del cual se lee el contenido
 * @param personas Las instancias de Persona a escribir en el fichero
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 */
@Override
public void escribirEnFichero(File fichero, Collection<Persona> personas) throws IOException {
    try (FileOutputStream fos = new FileOutputStream(fichero);
        BufferedOutputStream bs = new BufferedOutputStream(fos);
        ObjectOutputStream writer = new ObjectOutputStream(bs)) // crea un flujo de salida de objetos al fichero
    {
        writer.writeObject(personas); // escribe un objeto de Persona serializado en el fichero
    }
}

/**
 * Escribe instancias de Persona en un fichero binario
 * @param fichero Fichero del cual se lee el contenido
 * @param personas Las instancias de Persona a escribir en el fichero
 * @param anadir Este parámetro será ignorado
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 */
@Override
public void escribirEnFichero(File fichero, Collection<Persona> personas, boolean anadir) throws IOException {
    // esta clase no da soporte a 'append'. Convoca la versión de este método sin append
    escribirEnFichero(fichero, personas);
}
}
```

main()

Igual que el ejercicio anterior, pero también ofrece la opción ESCRIBIR_BINARIO, que genera un FicheroPersonas con instancia de **IOFicheroBinarioPersona** y ejecuta el ProgramaEscritura.

```
public class Ejercicio4 {

    private static final int ESCRIBIR = 1;
    private static final int LEER = 2;
    private static final int ESCRIBIR_BINARIO = 3;
    private static final int SALIR = 0;

    private static final String MENU =
        "Entrada/Salida de datos. Elige una opción:\n" +
        "1- Escribir\n" +
        "2- Leer\n" +
        "3- Escribir binario\n" +
        "0- Salir\n" +
        "Opción: ";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        InputUsuario input = new InputUsuario(scanner);

        int programa = input.solicitarEntero(MENU, ValidadorNumeros.enIntervalo(0, 3));
        FicheroPersonas fichero;

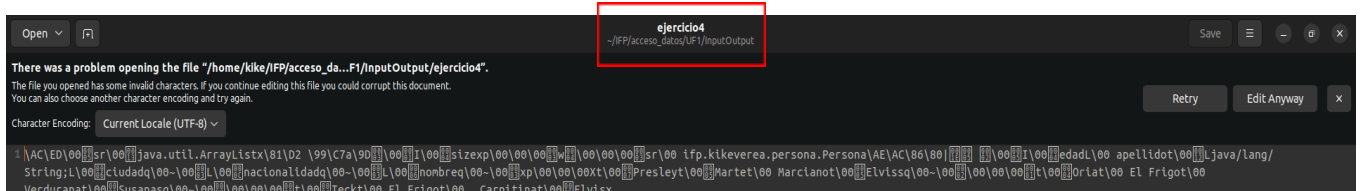
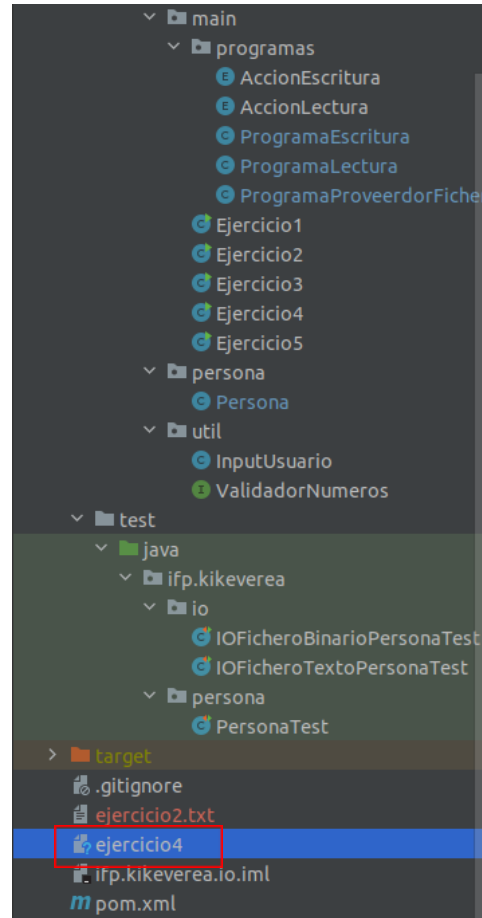
        switch (programa) {
            case ESCRIBIR:
                fichero = ProgramaProveedorFicheros.solicitarFichero(new IOFicheroTextoPersona(), input);
                ProgramaEscritura.ejecutar(fichero, input);
                break;
            case LEER :
                fichero = ProgramaProveedorFicheros.solicitarFicheroExistente(new IOFicheroTextoPersona(), input);
                ProgramaLectura.ejecutar(fichero, input);
                break;
            case ESCRIBIR_BINARIO:
                fichero = ProgramaProveedorFicheros.solicitarFichero(new IOFicheroBinarioPersona(), input);
                ProgramaEscritura.ejecutar(fichero, input);
                break;
            case SALIR:
                System.out.println("Programa finalizado");
                System.exit(0);
                break;
        }
    }
}
```

```

Entrada/Salida de datos. Elige una opción:
1- Escribir
2- Leer
3- Escribir binario
0- Salir
Opción: 3
Ruta y/o nombre del fichero: ejercicio4
Número de personas que desea escribir (máx. 3): 3
***** Persona 1 *****
Nombre: Elvia
Apellido: Presley
Ciudad: Marte
Nacionalidad: Marciano
Edad: 88
***** Persona 2 *****
Nombre: Susana
Apellido: Orta
Ciudad: El Frigo
Nacionalidad: Verdurana
Edad: 1
***** Persona 3 *****
Nombre: Elvia
Apellido: Teck
Ciudad: El Frigo
Nacionalidad: Carnitina
Edad: 2
Fichero creado correctamente

Process finished with exit code 0

```



Ejercicio 5

Clase IOFicheroBinarioPersona

También encapsula la lectura de instancias de Persona de ficheros binarios. Si el fichero no contiene instancias de Persona, la lectura produce un error de tipo *DatosNoContienenPersonasException*.

```
@Override
/unchecked/
public Collection<Persona> leerContenido(File fichero) throws IOException, DatosNoContienenPersonasException {
    try (FileInputStream fis = new FileInputStream(fichero);
        BufferedInputStream bs = new BufferedInputStream(fis);
        ObjectInputStream reader = new ObjectInputStream(bs))
    {
        return (Collection<Persona>) reader.readObject(); // lee y reconstruye un objeto de Persona del fichero
    }
    catch (ClassNotFoundException e) {
        throw new DatosNoContienenPersonasException("El fichero no contiene objetos de Persona");
    }
}

/**
 * Lee objetos tipo Persona de un fichero binario, que pasen el filtro dado. Asume que el fichero existe
 * @param fichero Fichero del cual se lee el contenido
 * @param filtro El filtro a aplicar a la lectura
 * @return Colección de Personas contenidas en el fichero, que pasan el filtro
 * @throws IOException Si el fichero no existe, o hay excepciones de tipo input/output
 * @throws DatosNoContienenPersonasException Si el fichero no contiene instancias de Persona
 */
@Override
public Collection<Persona> leerContenido(File fichero, FiltroLectura<Persona> filtro) throws IOException,
    DatosNoContienenPersonasException
{
    return leerContenido(fichero) Collection<Persona>
        .stream() Stream<Persona>
        .filter(filtro::pasaFiltro)
        .collect(Collectors.toList());
}
```

main()

Igual que el ejercicio anterior, pero también ofrece la opción LEER_BINARIO, que genera un FicheroPersonas **existente** con instancia de **IOFicheroBinarioPersona** y ejecuta el ProgramaLectura.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    InputUsuario input = new InputUsuario(scanner);

    int programa = input.solicitarEntero(MENU, ValidadorNumeros.enIntervalo(0, 4));
    FicheroPersonas fichero;

    switch (programa) {
        case ESCRIBIR:
            fichero = ProgramaProveedorFicheros.solicitarFichero(new IOFicheroTextoPersona(), input);
            ProgramaEscritura.ejecutar(fichero, input);
            break;
        case LEER :
            fichero = ProgramaProveedorFicheros.solicitarFicheroExistente(new IOFicheroTextoPersona(), input);
            ProgramaLectura.ejecutar(fichero, input);
            break;
        case ESCRIBIR_BINARIO:
            fichero = ProgramaProveedorFicheros.solicitarFichero(new IOFicheroBinarioPersona(), input);
            ProgramaEscritura.ejecutar(fichero, input);
            break;
        case LEER_BINARIO:
            fichero = ProgramaProveedorFicheros.solicitarFicheroExistente(new IOFicheroBinarioPersona(), input);
            ProgramaLectura.ejecutar(fichero, input);
            break;
        case SALIR:
            System.out.println("Programa finalizado");
            System.exit(0);
            break;
    }
}
```

```
Entrada/Salida de datos. Elige una opción:
1- Escribir
2- Leer
3- Escribir binario
4- Leer binario
0- Salir
Opción: 4
Ruta y/o nombre del fichero: ejercicio4
Elige una acción de lectura:
1- Leer todo el archivo:
2- Leer una persona:
Acción: 1
Personas:
Nombre: 'Elvis', Apellido: 'Presley', Ciudad: 'Marte', Nacionalidad: 'Marciano', Edad: '88'
Nombre: 'Susana', Apellido: 'Oria', Ciudad: 'El Frigo', Nacionalidad: 'Verdurana', Edad: '1'
Nombre: 'Elvis', Apellido: 'Teck', Ciudad: 'El Frigo', Nacionalidad: 'Carnitina', Edad: '2'

Process finished with exit code 0
```

```
Entrada/Salida de datos. Elige una opción:
1- Escribir
2- Leer
3- Escribir binario
4- Leer binario
0- Salir
Opción: 4
Ruta y/o nombre del fichero: ejercicio4
Elige una acción de lectura:
1- Leer todo el archivo:
2- Leer una persona:
Acción: 2
Nombre de la persona: Elvis
Personas:
Nombre: 'Elvis', Apellido: 'Presley', Ciudad: 'Marte', Nacionalidad: 'Marciano', Edad: '88'
Nombre: 'Elvis', Apellido: 'Teck', Ciudad: 'El Frigo', Nacionalidad: 'Carnitina', Edad: '2'

Process finished with exit code 0
```