MANEJO DE FICHEROS OBJETOS

Javier García-Retamero Redondo

¿QUÉ SON?

- **Flujo o Stream:**
- Canal de comunicación que se establece para cualquier envío o recepción de información entre una fuente y un destino.

- Trabajaremos con dos tipos:
  - Flujos de 16 bits: Operaciones E/S de **caracteres** (Unicode).
    - Clases descendientes de **Reader** y **Writer**.
  - Flujos de 8 bits: Operaciones E/S de **bytes**.
    - Clases descendientes de **InputStream** y **OutputStream.**
    - **No son legibles directamente**
    - **Ocupan menos espacio en disco**

TRABAJANDO CON ARCHIVOS BINARIOS

¿QUÉ PODEMOS HACER?

- La clase **OutputStream** nos permite:
  - **Escribir** bytes en distintas fuentes: array de bytes, un fichero.
  - La subclase más utilizada es **FileOutputStream:** Escribir información en un fichero.
    - Ideal para escribir datos binarios simples, como imágenes, archivos binarios, o texto convertido a bytes.
    - No tiene métodos para escribir tipos de datos primitivos directamente (como int, boolean, double, etc.).

ESCRITURA PASOS

- **Abrimos el fichero:**

```
File ficheroPrueba = new File(".\\archivos_pruebas\\ejemploprueba.dat");
```

- **Creamos el flujo de escritura:**
  - **Opción 1:**

```
FileOutputStream ficheroOut = new FileOutputStream(ficheroPrueba);
```

Si el fichero no existe lo crea.
Si existe lo destruye y lo vuelve a crear.

  - **Opción 2:**

```
FileOutputStream ficheroOut = new FileOutputStream(ficheroPrueba, true);
```

Añadimos datos y no destruimos el fichero ya creado.

Escritura……

- **Cerramos el flujo:**

```
ficheroOut.close();
```

TRABAJANDO CON ARCHIVOS BINARIOS

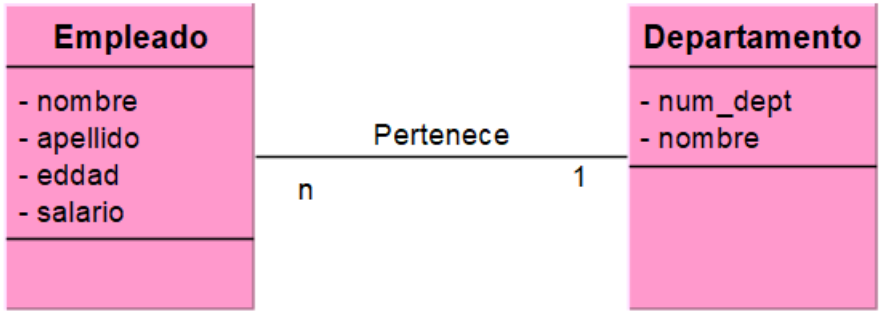
OBJETOS

¿QUÉ VAMOS A AÑADIR?

- Para realizar operaciones de L/E de objetos vamos a necesitar la clase **ObjectOutputStream.**
- Esta clase se utiliza para serializar objetos, es decir, para escribir objetos en una secuencia de bytes que puede almacenarse en un archivo o enviarse por una red.
- Extiende: OutputStream
- Usada junto con: FileOutputStream para escribir en archivos

CLASES

- Tenemos que crear las clases que van a representar a los objetos que vamos a almacenar en el archivo.



SERIALIZACIÓN

- Serializar objetos es el proceso de convertir un objeto en una secuencia de bytes o en un formato que pueda ser almacenado (por ejemplo, en un archivo o base de datos) o transmitido (por ejemplo, a través de una red), y luego reconstruido (deserializado) más adelante.

```
public class Empleado implements Serializable {

public class Departamento implements Serializable {
```

ESCRITURA

Escritura mediante ObjectOutputStream:

```
FileOutputStream ficheroOut = new FileOutputStream(getRuta());
ObjectOutputStream = new ObjectOutputStream(ficheroOut);

datosOut.writeObject(objeto);
```

EVITAR ERRORES AL GUARDAR OBJETOS

- Cada vez que se crea una nueva instancia de **ObjectOutputStream**, se escribe una **cabecera** al inicio del flujo. Si haces esto varias veces sobre el mismo archivo (por ejemplo, para añadir más objetos), se escriben múltiples cabeceras, lo que provoca errores al deserializar con ObjectInputStream

- Para evitar este problema, se puede crear una clase personalizada que evite escribir la cabecera si el archivo ya contiene datos. Esto se hace sobrescribiendo el método **writeStreamHeader()**.

EVITAR ERRORES AL GUARDAR OBJETOS

```
public class MiObjectOutputStream extends ObjectOutputStream {

    public MiObjectOutputStream(OutputStream out) throws IOException {
        super(out);
    }
    protected MiObjectOutputStream() throws IOException, SecurityException {
        super();
    }

   @Override
    protected void writeStreamHeader() throws IOException {
        reset();
    }
}
```

ESCRITURA

Escritura mediante ObjectOutputStream escribiendo sólo una cabecera:

```
FileOutputStream ficheroOut = null;
ObjectOutputStream datosOut = null;

 if (existeFichero()){
     ficheroOut = new FileOutputStream(getRuta(), true);
     datosOut = new MiObjectOutputStream(ficheroOut);
 } else {
     // Si el archivo es nuevo, utilizamos ObjectOutputStream para que cree la primera cabecera.
     ficheroOut = new FileOutputStream(getRuta());
     datosOut = new ObjectOutputStream(ficheroOut);
 }
 datosOut.writeObject(objeto);
```

LECTURA

Lectura mediante ObjectInputStream:

```
ArrayList <Object> retornoList = new ArrayList();
Object retorno = null;

FileInputStream ficheroIn = new FileInputStream(getRuta());
ObjectInputStream datosIn = new ObjectInputStream(ficheroIn);

while (ficheroIn.available()>0){
        retorno =  datosIn.readObject();
        retornoList.add(retorno);
}
```

LECTURA

Conversión de un ArrayList de Object a un ArrayList de una clase específica:

```
public static <T> ArrayList<T> convertirArrayList(ArrayList<Object> listaOriginal,
        Class<T> claseDestino) {
    return listaOriginal.stream()
        .filter(claseDestino::isInstance)
        .map(claseDestino::cast)
        .collect(Collectors.toCollection(ArrayList::new));
}
```

LECTURA

Donde:

- **filter(claseDestino::isInstance):** Filtra los elementos que son instancia de claseDestino.
  - Por ejemplo, si claseDestino es String.class, se quedarán solo los elementos que sean String.

- **map(claseDestino::cast):** Convierte cada elemento filtrado al tipo deseado.

- **collect(Collectors.toCollection(ArrayList::new)):** Recolecta los elementos en un nuevo ArrayList.