CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MODELO EXAMEN 2

M06. ACCESO A DATOS

UF1: Persistencia en ficheros

1) Indica qué dos tipos de ficheros existen atendiendo a su codificación (Como está almacenada la información en ellos) y explícalos brevemente. ¿Cuáles son las operaciones básicas que podemos hacer sobre ellos? (3 Pts).

Archivos de texto: son generados por un editor y se suele utilizar de forma genérica un tipo de formato como ASCII, UNICODE o UTF8 que sirven para almacenar los caracteres.
Al trabajar con estos formatos se utilizan las clases **FileReader** para leer estos caracteres y **FileWriter** para escribirlos. Para trabajar con estos archivos y poder leerlos o escribir con ellos, se hará siempre dentro de la excepción **try-catch**.

Archivos binarios: almacenan secuencias de digitos binarios que no son legibles directamente por el usuario como ocurría con los ficheros de texto. Tienen la ventaja de que ocupan menos espacio en disco.
Las dos clases que se pueden usar para realizar el trabajo con archivos binarios en Java son:

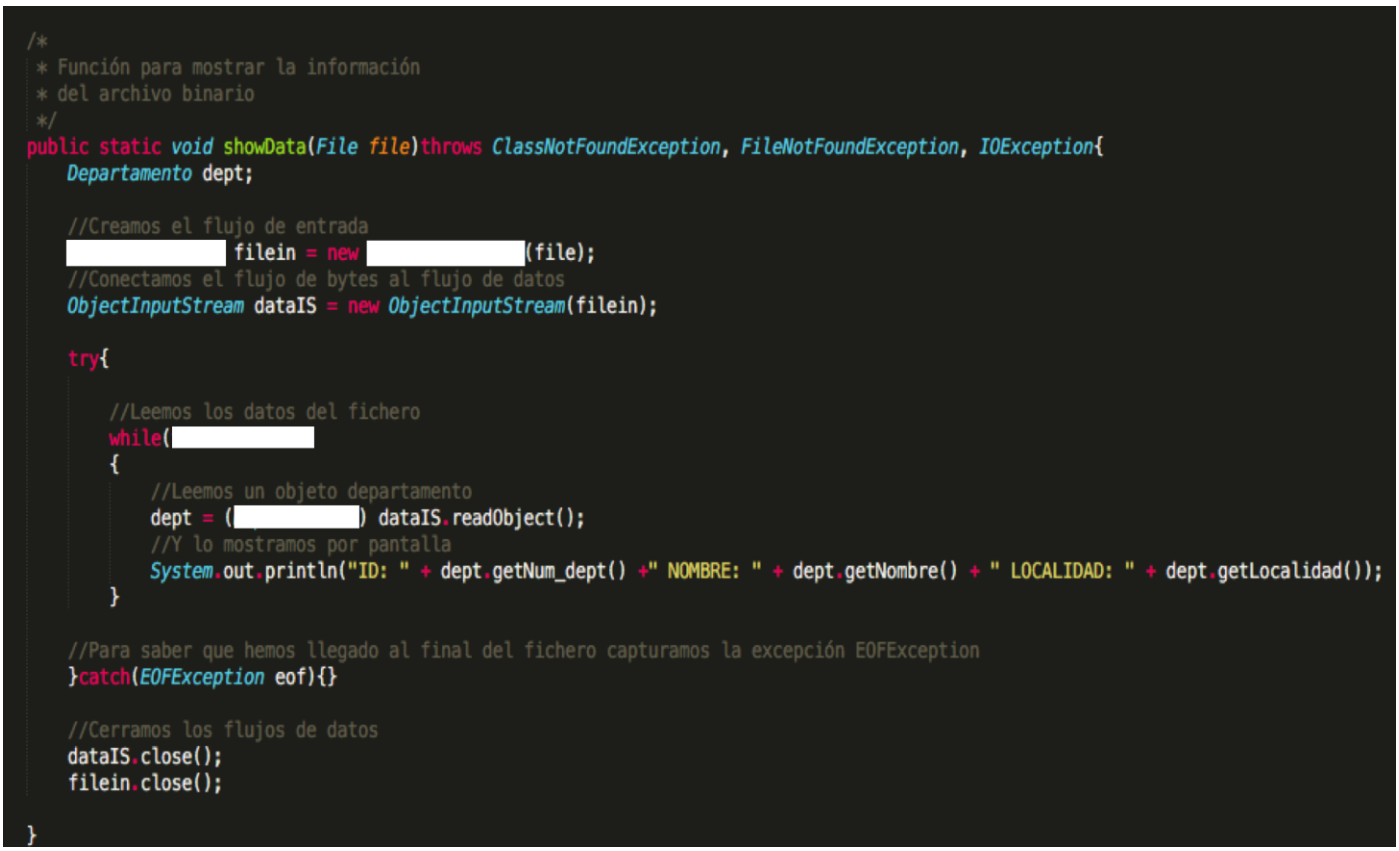
1. FileInputStream (para entrada).

2. FileOutputStream (para salida).

2) Explica brevemente que hace el siguiente código (Tanto a nivel global como cada línea en particular) y complétalo con lo necesario para que el programa funcionase. (7 Pts) Respondido en modelo 2 de examen

```
/*
 * Función de creación/apertura del archivo
 *
 */
public static RandomAccessFile createFile(String file) throws [ ]NotFoundException{
    //Creamos un fichero en la ruta que pasamos por parametro.
    File fichero=new File(file);

    //Creamos y retornamos un nuevo fichero de acceso aleatorio generado a partir del fichero
    return new [ ](fichero, "rw");
}
```



```
/*
 * Función para mostrar la información
 * del archivo binario
 */
public static void showData(File file)throws ClassNotFoundException, FileNotFoundException, IOException{
    Departamento dept;

    //Creamos el flujo de entrada
    [redacted] filein = new [redacted](file);
    //Conectamos el flujo de bytes al flujo de datos
    ObjectInputStream dataIS = new ObjectInputStream(filein);

    try{

        //Leemos los datos del fichero
        while([redacted])
        {
            //Leemos un objeto departamento
            dept = ([redacted]) dataIS.readObject();
            //Y lo mostramos por pantalla
            System.out.println("ID: " + dept.getNum_dept() +" NOMBRE: " + dept.getNombre() + " LOCALIDAD: " + dept.getLocalidad());
        }

    //Para saber que hemos llegado al final del fichero capturamos la excepción EOFException
    }catch(EOFException eof){}

    //Cerramos los flujos de datos
    dataIS.close();
    filein.close();

}
```

UF2: Persistencia en Bases de datos R, O-R, OO

1) Explica las principales similitudes y diferencias entre ODBC y JDBC. (3 Pts)

Similitudes:

- Ambos son normas de conexión a una base de datos SQL.
- Ambos definen una API que puede usar las aplicaciones para abrir una conexión a una base de datos SQL.

Diferencias:

- La API OBDC usa una interfaz escrita en el lenguaje C y no es apropiada para su uso directo desde Java (las llamadas desde Java a código C nativo tienen un número de inconvenientes en la seguridad,

implementación, robustez y portabilidad de las aplicaciones), mientras que la JDBC interfaz está escrita en Java y, por tanto, es ideal para dar acceso a los datos desde aplicaciones Java.

2) Explica brevemente que hace el siguiente código (Tanto a nivel global como cada línea en particular) y complétalo con lo necesario para que el programa funcionase. (6 Pts) Ver solución en modelo 1.

```
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost/ud2";
static final String USER = "ejemplo";
static final String PASS = "ejemplo";
static Connection conn = null;
static Statement stmt = null;


public void conectar(){
    try {
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection(                );
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

//Función encargada de cerrar la conexión
public void desconectar(){
    try {
     conn.close();
 } catch (SQLException e) {
     e.printStackTrace();
 }
}
```

```
public static boolean insertCliente(String nombre, String direc, String pobla, String tele
    try {
        stmt = conn.
        String sql;
        sql = "insert into clientes (clientes.Nombre, clientes.Direccion, clientes.Poblaci
        values('"+nombre+"','"+direc+"','"+pobla+"','"+telef+"','"+nif+"');";
        int result= stmt.          (sql);
        stmt.close();
        if(result>0) return true;
        else return false;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}
```

3) ¿Qué son las clases persistentes de Hibernate? Explica su utilidad/función. (1Pt)

Ver solución a modelo 1

UF3: Persistencia en Bases de datos XML

1) ¿Qué es eXist? Descríbelo brevemente y nombre sus principales características y sus ventajas e inconvenientes. (3 Pts)

eXist es un SGBD libre de código abierto en el que se almacenan datos XML con un motor de base de datos escrito en Java y que es capaz de soportar los estándares de consulta XPath, XQuery y XSLT, además de indexar documentos. Asimismo, sirve para una gran cantidad de protocolos como SOAP, XML-RPC, WebDav y REST.

- Ventajas:
    1. eXist es flexible, permitiendo almacenar documentos sin especificar ningún schema.

2. Portabilidad de las consultas a otros productos que proporcionen procesador de XQuery y/o XLST.
3. Búsquedas estructuradas.
4. A diferencia de cualquier otra base de datos NoSQL, eXist proporciona soporte para XForms (formularios Web escritos en XML).
5. Cuando eXist se utiliza como servidor, permite desarrollar aplicaciones fácilmente.

- Desventajas:
  1. eXist no ofrece soporte para que el usuario pueda tener control de transacciones a nivel de la base de datos.
  2. eXist no ofrece soporte para fragmentación automática de datos.

2) Explica brevemente que hace el siguiente código (Tanto a nivel global como cada línea en particular) y complétalo con lo necesario para que el programa funcionase. (6 Pts) Ver solución a modelo 1

```
static String driver = "org.exist.xmldb.DatabaseImpl";
static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc/db/ColeccionPruebas";
static String usu = "admin";
static String usuPwd = "root";
static Collection col = null;
public static Collection conectar() {
    try {
        Class<?> cl = Class.forName(      );
        Database database = (Database) cl.newInstance();

        DatabaseManager.registerDatabase(database);
        col = DatabaseManager.getCollection(                );
        return col;
    } catch (XMLDBException e) {
        System.out.println("Error al inicializar la BD eXist.");
         e.printStackTrace();
    } catch (ClassNotFoundException e) {
        System.out.println("Error en el driver.");
        e.printStackTrace();
    } catch (InstantiationException e) {
        System.out.println("Error al instanciar la BD.");
         e.printStackTrace();
    } catch (IllegalAccessException e) {
        System.out.println("Error al instanciar la BD.");
         e.printStackTrace();
    }
    return null;
}
```

```
private static void listarDep() {
    if (conectar() != null) {
        try {
            XPathQueryService servicio;
            servicio = (XPathQueryService) col.[redacted]("XPathQueryService", "1.0");
            ResourceSet result = servicio
                    .query("for $dep in doc('departamentos.xml')/departamentos/DEP_ROW return $dep");
            ResourceIterator i;
            i = result.[redacted];
            if (!i.hasMoreResources()) {
                System.out.println(" LA CONSULTA NO DEVUELVE NADA O ESTÁ MAL ESCRITA");
            }
            while (i.hasMoreResources()) {
                Resource r = i.[redacted]();
                System.out.println("----------------------------------------");
                System.out.println((String) r.getContent());
            }
            col.[redacted];
        } catch (XMLDBException e) {
            System.out.println(" ERROR AL CONSULTAR DOCUMENTO.");
            e.printStackTrace();
        }
    } else {
        System.out.println("Error en la conexión. Comprueba datos.");
    }

}
```

UF4: Componentes de acceso a datos

1) Explica brevemente el patrón MVC, sus características, ventajas e inconvenientes (3 Pts).

El patrón MVC (Model-View-Controller) es un patrón de diseño que se utiliza como guía para el diseño de arquitecturas software que ofrecen una fuerte interactividad con el usuario y donde se requiere una separación de conceptos para que el desarrollo se realice más eficientemente facilitando la programación en diferentes capas de manera paralela e independiente. Este patrón organiza la aplicación en 3 bloques cada cual especializado en una tarea:

- El Modelo: representa los datos de la aplicación y sus reglas de negocio.
- La Vista: es la representación del modelo de forma gráfica para interactuar con el usuario, un ejemplo son los formularios de entrada y salida de información o páginas HTML con contenido dinámico.
- El Controlado: interpreta los datos que recibe del usuario analizando la petición, coordinando la vista y el modelo para que la aplicación produzca los resultados esperados.

Las ventajas:

- Separación de los datos de la representación visual de los mismos.
- Diseño de aplicaciones modulares.
- Reutilización de código.
- Facilidad para probar las unidades por separado.

- Facilita el mantenimiento y la detección de errores.

Entre las desventajas cabe destacar la complejidad que se agrega al sistema al separar los conceptos en capas o la cantidad de ficheros a desarrollar que se incrementa considerablemente.

2) ¿Qué es un componente? Indica sus principales características explicándolas brevemente. (1 Pts).

- Ver solución de modelo 1

3) ¿Qué características debe cumplir un JavaBean? Explica brevemente cada una de ellas. (1 Pts).

- Ver solución de modelo 1

4) ¿Qué debe contener el fichero Manifest.mf de un .jar? (1 Pts).

- Ver solución de modelo 1

5) Explica brevemente que hace el siguiente código (Tanto a nivel global como cada línea en particular) y complétalo con lo necesario para que el programa funcionase. (4 Pts) Ver solución de modelo 1.

```
int idproducto = 6;
int cantidad = 89;
ODB odb = ODBFactory.[redacted]("Producto_Ped.BD");
IQuery query = new [redacted](Producto.class, Where.equal("idproducto", idproducto));
Objects<Producto> objetos = odb.getObjects(query);
try{
    Producto pro = ([redacted]) objetos.getFirst();
    System.out.println("ID=>" + idproducto + ": " + pro.getDescripcion() + " * STOCK-ACT:" + pro.getStockactual() +

    if(cantidad>0){
        java.sql.Date fechaActual = getCurrentDate();

        System.out.println("CANTIDAD A VENDER: " + cantidad);
        if(actualizarStock(pro,odb,cantidad)){
            int numeroventa = obtenerNumeroVenta(odb);
            Venta ven = new Venta(numeroventa,idproducto,fechaActual,cantidad);
            odb.store(ven); //Almacenar venta
            System.out.println("VENTA: " + numeroventa + " INSERTADA...");
        }else{
            System.out.println("VENTA NO INSERTADA. NO HAY STOCK...");
        }
    }else{
        System.out.println("LA CANTIDAD DEBE SER MAYOR QUE 0");
    }
}catch(IndexOutOfBoundsException e){
    System.out.println("NO EXISTE EL PRODUCTO");
}finally{
    odb.[redacted];
}
```