CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MODELO EXAMEN 3

M06. ACCESO A DATOS

UF1: Persistencia en ficheros

1) Explica todo lo que sepas sobre los ficheros XML y su tratamiento en JAVA, operaciones que podemos hacer sobre ellos, tratamiento de excepciones, etc. (3 Pts).

Los archivos XML son archivos de texto en lenguaje XML donde se organiza la información de forma secuencial y jerarquizada.

Los archivos XML se pueden utilizar para distintas funciones:

- Proporcionar datos en una base de datos
- Almacenar copias de esas bases de datos
- Escribir archivos de configuración de programas
- Efectuar comandos en servidores remotos en el protocolo SOAP

Para la lectura de estos tipos de documentos XML se utilizan procesadores de XML o ***parser***, los cuales permiten acceder a su contenido y estructura. Hay versiones de los parser DOM y SAX para ser utilizados desde una aplicación Java.

Cuando existe un error dentro de un método Java, este crea un objeto **Exception** y lo maneja fuera. Está diseñado para cuando no es capaz de manejar información y lanza un error.

Para capturar excepciones se utiliza el bloque llamado **try-catch**. En el primer bloque try se incluye el código que podrá generar una excepción y seguidamente se crean tantos bloques catch como se deseen, indicando el tipo de excepción que podrá generar en cada uno de ellos.

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)

```
/*
 * Función de creación/apertura del archivo
 *
 */
public static RandomAccessFile createFile(String file) throws [redacted]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 [redacted](fichero, "rw");
}
```

```java
/*
 * 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 todo lo que sepas sobre el acceso a base de datos a través de JDBC (3 Pts)

JDBC no es solo una librería con la que poder acceder a los datos, sino que también **define una arquitectura estándar en la que los fabricantes pueden crear sus drivers para que las aplicaciones Java puedan acceder a los datos**. JDBC tiene una intefaz que es distinta para cada BD. Es lo que se denomina como *driver* (controlador).

La función de este driver es que los métodos de las clases JDBC se puedan corresponder con el API de la BD.

A través de JDBC se pueden realizar las siguientes **tareas**:

- Conectar con una base de datos.
- Realizar consultas e instrucciones para actualizar la BD.
- Recuperar y procesar los resultados de la BD.

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)

```
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)

UF3: Persistencia en Bases de datos XML

1) Describe todo lo que sepas sobre el acceso a eXist desde Java. (3 Pts)

Hay varias APIs que sirven para acceder a la base datos eXists de un programa Java:

- La API XML:DB: los componentes básicos empleados son los drivers (encapsulan la lógica de acceso a la base de datos XML), una colección (contenedor de recursos y otras sub-colecciones) y los servicios (solicitados para tareas como consultar una colección con XPath o la gestión de una colección).

- La API XQJ que es una propuesta de estandarización de interfaz Java para el acceso a base de datos XML nativas basado en el modelo de datos XQuery. Con XQJ no necesitamos seleccionar la colección de los documentos XML, la búsqueda la realiza en todas las colecciones.

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 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).

   Ver solución en modelo 2

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

   Ver solución en modelo 1.

3) Describe brevemente todo lo que sepas sobre el empaquetado de componentes.

   Una vez creado el *JavaBean* es necesario empaquetarlo para proceder a su distribución y uso por las aplicaciones. Para distribuirlo se crea un fichero *JAR* que contiene un fichero de manifiesto (MANIFEST.MF) que describe el contenido. Se deberán incluir las clases y los recursos que lo forman.

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

   Ver solución en 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 en 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];
}
```