[Resumen Examen 2 Programación 2](#_Toc202128152)

[JDBC 2](#_Toc202128153)

[4 tipos de divers 2](#_Toc202128154)

[Conexión a la base de datos 2](#_Toc202128155)

[Interface Statement (las herramientas/metodos que te da JDBC) 2](#_Toc202128156)

[Connection 3](#_Toc202128157)

[Statement 3](#_Toc202128158)

[ResultSet 3](#_Toc202128159)

[DatabaseMetaData 3](#_Toc202128160)

[ResultSetMetaData 4](#_Toc202128161)

[Interfaz Gráfica de Usuario (GUI) 4](#_Toc202128162)

[Maneras de visualizar en JAVA: 4](#_Toc202128163)

[AWT (Abstract Windows Toolkit) 4](#_Toc202128164)

[Swing 4](#_Toc202128165)

[Applets 4](#_Toc202128166)

[JavaFX 4](#_Toc202128167)

[AWT vs Swing 5](#_Toc202128168)

[NetBeans y creación de GIU 5](#_Toc202128169)

[Crear interfaz 5](#_Toc202128170)

[Mas información sobre esto acá: 6](#_Toc202128171)

[Patrones de diseño 6](#_Toc202128172)

[¿Qué NO es un patrón? 6](#_Toc202128173)

[Características de patrones 7](#_Toc202128174)

[Tipos de patrones 7](#_Toc202128175)

[Estructura de un patrón 7](#_Toc202128176)

[Elementos de un patrón 8](#_Toc202128177)

[Formato para describir patrones (según GoF) 8](#_Toc202128178)

[Tipos de Patrones según GoF 8](#_Toc202128179)

[Creacionales 8](#_Toc202128180)

[Estructurales 8](#_Toc202128181)

[Comportamiento 9](#_Toc202128182)

[Ejercicios posibles de examen 9](#_Toc202128183)

[1-Responder Verdadero o Falso según corresponda, Justificar en caso de Falso 9](#_Toc202128184)

[2.1- Completa dando los valores a los “?#” según correspondan: 10](#_Toc202128185)

[2.2 -Completar dando los valores a los ‘?#’ según corresponda: 10](#_Toc202128186)

[3- Completar el siguiente código para que compile sin errores, y para que en tiempo de ejecución no arroje ninguna excepción (No es necesario definir la variable readP5) 11](#_Toc202128187)

[4.1 - Dado el siguiente código, indicar cuales de las afirmaciones son correctas 12](#_Toc202128188)

[4.2 - Dado el sig. código, indique cual/es de las afirmaciones son correctas: 13](#_Toc202128189)

[5-Desarrollar una clase que cumpla con el patrón de diseño Singleton: 13](#_Toc202128190)

[6 - Tomando como base las clases DAO y AlumnoDAOTxt vistas en clase: 14](#_Toc202128191)

# Resumen Examen 2 Programación

[Recomendado más a la práctica]

## JDBC

**Java Database Connectity** **es una herramienta que permite a programas Java conectarse a bases de datos y hacer consultas**. Es como un puente que une Java con distintos sistemas de bases de datos

### 4 tipos de divers

* **JDBC-ODBC Bridge Driver**: Usa tecnología **antigua** y **ya no se recomienda (Desde JAVA 8)**. Usa drivers ODBC para conectarse a la BD
* **Native Driver**: Usa **librerías** **propias** del sistema de base de datos, pero **requiere instalar software adicional**
* **Network Protocol Driver**: Necesita un **intermediario** (middleware) **entre Java y la base de datos**
* **Thin Driver**: Es el **más moderno y eficiente**. **No necesita instalar** nada **extra** y se **comunica directamente** con la base de datos. Mejor performance

### Conexión a la base de datos

El proceso tiene **5 pasos clave**:

1. **Registrar el driver**: con **Class.forName()** cargás el driver para que Java lo entienda
2. **Crear la conexión**: con **getConnection(url, user, pass)** te conectás a la base

Diagrama

El contenido generado por IA puede ser incorrecto.

1. **Hacés la consulta**: usás **createStatement()** para prepararla
2. **La ejecutás**: con **executeQuery("SQL"),** por ejemplo, para un SELECT
3. **Y cerrás todo**: con **close()** para que no quede nada colgado

Si usás **MySQL**, hay que tener el driver cargado en el proyecto (como una librería)

### Interface Statement (las herramientas/metodos que te da JDBC)

#### Connection

Es la que representa la conexión con la base. Con esta podés:

* **Hacer consultas (public Statement createStatement())**
* **Auto commit de transacciones (public void SetAutoCommit(boolean status))**
* Manejar si querés **confirmar o cancelar cambios (commit(), rollback())**
* **Cerrar la conexión (close())**

#### Statement

Sirve para ejecutar **comandos SQL:**

* **executeQuery()** es para obtener datos (como un SELECT)
* **executeUpdate() para modificar** cosas (INSERT, UPDATE, DELETE)
* **execute()** para cualquier otra consulta**.**
* También podés ejecutar varias juntas con **addBatch() y executeBatch()**

#### ResultSet

Es lo que te **devuelve** **la** **base** cuando hacés una consulta. Con esto podés leer los datos uno por uno, como recorrer una tabla

Tabla

El contenido generado por IA puede ser incorrecto.

#### DatabaseMetaData

Te da **info general de la base de datos**: qué versión es, qué funciones soporta, etc. Se obtiene desde la conexión

Imagen que contiene Aplicación

El contenido generado por IA puede ser incorrecto.

#### ResultSetMetaData

Te da **info de lo que devuelve una consulta**: cuántas columnas hay, qué tipo de datos tiene cada una, etc

Tabla

El contenido generado por IA puede ser incorrecto.

## Interfaz Gráfica de Usuario (GUI)

Una **interfaz gráfica** es lo que le permite a una persona interactuar con una app de **forma visual**, con ventanas, botones, cuadros de texto, etc., en lugar de usar solo texto o comandos

### Maneras de visualizar en JAVA:

#### AWT (Abstract Windows Toolkit)

Es una **librería vieja** de Java para hacer ventanas. Usa los elementos del sistema operativo, o sea que cada vez que haces una ventana, se ve como en Windows, Linux o Mac, según dónde corra  
*Es* ***dependiente del sistema*** *y* ***pesado*** *porque usa recursos de la máquina*

#### Swing

Mejora de AWT, viene con Java y no depende del S.O.  
Es mas liviano y flexible que AWT

#### Applets

Programas chiquitos que antes se usaban para correr Java en pagnias web. Hoy está en **desuso**

#### JavaFX

Moderno, sirve para interfaces lindas y completas, incluso apps webs que se vean como si fueran escritorio

### AWT vs Swing

**Tabla

El contenido generado por IA puede ser incorrecto.**

Swing mejora lo que hacía AWT. Con Swing podés:

* **Agregar componentes** con add(Component c)
* **Cambiar el tamaño** de los elementos con setSize()
* **Elegir cómo se acomodan los componentes** con setLayout()
* **Hacer que se vea todo** con setVisible(true)

Diagrama

El contenido generado por IA puede ser incorrecto.

### NetBeans y creación de GIU

### Crear interfaz

Con NetBeans podés armar ventanas arrastrando botones y campos, sin escribir todo el código a mano

*Ejemplo: hacés un proyecto llamado NumberAddition para sumar dos números.  
Agregás un formulario (JFrame), lo armás con paneles, etiquetas, botones, etc*

Por ejemplo:

* Botón **"Salir"**: hace que se cierre la app con System.exit(0);
* Botón **"Limpiar"**: borra el texto de los campos con .setText("")
* Botón **"Sumar"**: toma los números, los suma, y muestra el resultado

Para ejecutar:

* Dentro del IDE: Run > Run Project o apretás **F6**.
* Fuera del IDE: generás el .jar y lo corrés por consola con java -jar NombreDelArchivo.jar

#### Mas información sobre esto acá:

[*https://www.youtube.com/watch?v=18UA7X2ss8g&ab\_channel=ProgramadorNovato*](https://www.youtube.com/watch?v=18UA7X2ss8g&ab_channel=ProgramadorNovato)

*Si no se encuentra el enlace del video es: YouTube>Programador Novato> Curso de Java Netbeans Completo - 01*

## Patrones de diseño

Los patrones de diseño son **soluciones ya probadas para problemas comunes** que aparecen cuando desarrollas software

No son formulas mágicas, dan una mano

Un patrón es:

* Una guía para resolver problemas
* Algo que ya se uso varias veces y funcionó
* Se logra un nivel de abstracción mayor
* Un lenguaje común entre programadores para decir “Mira, acá usamos un Singleton” y se entienda a que se refiere
* Evita reiteraciones de basqueadas de soluciones
* Lo solución tiene un “template genérico”

No es algo que se “usa” tal cual, como si fuera librería, es mas **una idea o estrategia**, que **adaptas** a tu código según lo necesites

### ¿Qué NO es un patrón?

* No es garantía de que tu programa esté bien hecho
* No es obligatorio ni mágico
* No es algo que puedas copiar y pegar
* No se usa si no es necesario. Si no aporta nada, mejor no complicarse

### Características de patrones

* **Resuelven problemas reales**
* Son **probados**, no teoría
* No siempre son obvios
* Te muestran **cómo se relacionan los objetos y qué hace cada uno**
* Consideran también **la experiencia del usuario**

### Tipos de patrones

* **De Arquitectura:** Organizan sistemas grandes: cómo se conectan los módulos y subsistemas
* **De Diseño:** Ayudan a mejorar la estructura de componentes dentro del sistema
* **De Programación:** Tienen que ver con cómo se implementa el código, más al detalle
* **De Análisis:** Sirven para entender y modelar bien un problema antes de codificar
* **Organizacionales:** Aplican al funcionamiento de grupos humanos o empresas (no tan técnicos, pero igual importantes)

### Estructura de un patrón

1. **Nombre del patrón:** cómo se lo conoce (ej: Singleton, Observer)
2. **Clasificación:** si es creacional, estructural o de comportamiento
3. **Intención:** qué problema busca resolver
4. **También conocido como:** otros nombres o sinónimos
5. **Motivación:** ejemplo práctico de cuándo usarlo
6. **Aplicabilidad:** en qué situaciones es útil
7. **Estructura:** diagrama o explicación de cómo se organizan las clases
8. **Participantes:** quiénes intervienen en el patrón (clases, objetos) y qué hace cada uno
9. **Colaboraciones:** cómo se relacionan entre sí los participantes
10. **Consecuencias:** qué ventajas y desventajas tiene aplicar el patrón
11. **Implementación:** consejos o detalles a tener en cuenta para programarlo
12. **Código de ejemplo:** fragmento que muestra cómo se aplicaría en la práctica
13. **Usos conocidos:** ejemplos reales donde se usó este patrón
14. **Patrones relacionados:** otros patrones que se usan junto con este o se parecen

### Elementos de un patrón

1. **Nombre del patrón:** algo corto y claro que describa el problema
2. **Problema:** ¿cuándo conviene usarlo?
3. **Solución:** ¿cómo lo resuelve? (la idea general, no el código)
4. **Consecuencias:** ¿qué ventajas y desventajas tiene?

### Formato para describir patrones (según GoF)

Incluye:

* **Nombre** y **clasificación** (creacional, estructural o de comportamiento),
* Qué problema resuelve,
* Escenario de ejemplo (motivación),
* Cuándo conviene usarlo (aplicabilidad),
* Estructura (diagrama de clases),
* Participantes y cómo se relacionan,
* Código de ejemplo y sistemas reales que lo usan.

### Tipos de Patrones según GoF

Creacionales**:**

Abstraen el proceso, son dinamicas

Cómo se crean objetos y quién se encarga:

* Singleton
* Factory Method
* Abstract Factory
* Builder
* Prototype

Estructurales**:**

Estructuras entre clases y objetos, estudian con diagramas de clases/objetos

Cómo se organizan las clases y objetos

* Adapter
* Bridge
* Composite
* Decorator
* Facade
* Flyweight
* Proxy

#### Comportamiento

Asignan responsabilidades entre objetos y su comunicación, suelen ser secuenciales. Combinan comportamientos

Cómo interactúan los objetos entre sí:

* Observer
* Strategy
* Command
* Iterator
* Mediator
* State
* Chain of Responsibility
* Template Method
* Visitor
* (entre otros)

# Ejercicios posibles de examen

#### 1-Responder Verdadero o Falso según corresponda, Justificar en caso de Falso

1. Al definir un constructor de clase sin modificador de acceso, es decir package-private, significa que solamente se lo puede invocar desde la misma clase
2. El método split() de la clase String elimina los espacios en blanco en ambos lados de un string
3. Las siglas del patrón DAO significan Data Abstract Object
4. Singleton es un patrón de diseño que garantiza que tan solo exista un objeto de su tipo y proporciona un único punto de acceso a él para cualquier otro código.

#### 2.1- Completa dando los valores a los “?#” según correspondan:

public class DAOAlumnoFactory {

public static final String TIPO\_DAO = "TIPO\_DAO";

public static final String DAO\_TXT = "DAO\_TXT";

public static final String DAO\_SQL = "DAO\_SQL";

public static final String FULL\_PATH = "FULL\_PATH";

public static final String SQL\_CONNECTION = "SQL\_CONNECTION";

public static final String URL\_DB = "URL\_DB";

public static final String USER\_DB = "USER\_DB";

public static final String PWD\_DB = "PWD\_DB";

public static DAO<Alumno> crearDAO(Map<String, String> **?1**) throws **?2** {

try {

String tipoDAO = configMap.get(TIPO\_DAO);

switch (tipoDAO) {

case DAO\_TXT:

String fullpath = configMap.get(FULL\_PATH);

return new DAOAlumnoTxt(fullpath);

case **?3**:

String url = configMap.get(URL\_DB);

String user = configMap.get(USER\_DB);

String password = configMap.get(PWD\_DB);

return new DAOAlumnoSQL(url, user, password);

default:

throw new **?4**("No implementado");

}

} catch (DAOException **?5**) {

Logger.getLogger(DAOAlumnoFactory.class.getName()).log(Level.SEVERE, null, ex);

throw new DaoFactoryException("Error al crear el DAO ("

+ ex.getLocalizedMessage() + ")");

}

}

}

#### 2.2 -Completar dando los valores a los ‘?#’ según corresponda:

public class DAOAlumnoFactory {

public static final String TIPO\_DAO = "TIPO\_DAO";

public static final String DAO\_TXT = "DAO\_TXT";

public static final String DAO\_SQL = "DAO\_SQL";

public static final String FILE\_NAME = "FILE\_NAME";

public static final String SQL\_CONNECTION = "SQL\_CONNECTION";

public DAO crearDAO**(?1**<String, String> config) **?2** DAOException {

String tipo = config.get(TIPO\_DAO);

switch (tipo){

case **?3**:

String filename = config.get(FILE\_NAME);

return new AlumnoDAOTXT**(?4);**

case DAO\_SQL:

return new AlumnoDAOSQL(config.get(**?5**), "root", "root");

default:

throw new **?6**("Tipo de DAO no implementado");

}

}

}

#### 3- Completar el siguiente código para que compile sin errores, y para que en tiempo de ejecución no arroje ninguna excepción (No es necesario definir la variable readP5)

@Override

public Alumno read(Integer dni) throws DaoException {

try {

readPS.setInt(0, dni);

ResultSet rs = readPS.executeQuery();

if (rs.next()) {

Alumno alumno = new Alumno();

alumno.setDni(rs.getString ("DNI"));

alumno.setNombre(rs.getString("NOMBRE"));

alumno.setFechaNac(AlumnoUtils.sqlDate2LocalDate(rs.getDate("FEC\_NAC")));

return alumno;

}

} catch (SQLException ex) {

Logger.getLogger(DaoSQL.class.getName()).log(Level.SEVERE, null, ex);

throw new DaoException("Error SQL => No se pudo leer el alumno ("

+ ex.getLocalizedMessage() + ")", ex);

} catch (PersonaException ex) {

Logger.getLogger(DaoSQL.class.getName()).log(Level.SEVERE, null, ex);

throw new DaoException("Error al crear el alumno ("

+ ex.getLocalizedMessage() + ")", ex);

}

return;

}

#### 4.1 - Dado el siguiente código, indicar cuales de las afirmaciones son correctas

@Override

public void update(Alumno alu) throws DaoException {

try {

long filePointer = 0;

raf.seek(0);

String lineaAlu;

Integer dniAlu;

while ((lineaAlu = raf.readLine()) != null) {

dniAlu = Integer.valueOf(lineaAlu.substring(0, 8));

if (dniAlu.equals(alu.getDni())) {

raf.seek(filePointer);

raf.writeBytes(alu.toString());

return;

}

filePointer = raf.getFilePointer();

}

} catch (IOException ex) {

Logger.getLogger(DaoTXT.class.getName()).log(Level.SEVERE, null, ex);

throw new DaoException("Error E/S ==> No se pudo leer el archivo"+

“(“+ ex.getLocalizedMessage() + ")");

}

}

1. La variable filePointer guarda la posición de fin de la linea con el objetivo que raf pueda continuar leyendo.
2. La línea de código raf.seek(0); se utiliza para abrir el archivo de texto para comenzar a utilizarlo.
3. El método getFilePointer() de RandomAccessFile me permite posicionar el puntero del archivo en la posición deseada.
4. Todas son correctas.
5. Ninguna es correcta

#### 4.2 - Dado el sig. código, indique cual/es de las afirmaciones son correctas:

@Override

public void update(Alumno alu) throws DAOException

{

archivoRandomAccessFile.seek(0);

String linea;

String dniString;

long posLinea = 0;

while((linea = archivoRandomAccessFile.readLine()) != null)

{

dniString = linea.substring(0, 8);

if(Integer.valueOf(dniString).equals(alu.getDni()))

{

archivoRandomAccessFile.seek(posLinea);

archivoRandomAccessFile.writeBytes(alu.toString());

return;

}

posLinea = archivoRandomAccessFile.getFilePointer();

}

}

1. El objetivo de la variable posLinea es la de guardar la posición de fin de la línea de texto a actualizar para que archivoRandomAccessFile pueda continuar con la lectura en la siguiente línea.
2. La línea de código archivoRandomAccessFile.seek(0); se utiliza para abrir el archivo de texto para comenzar a utilizarlo.
3. El método getFilePointer() de RandomAccessFile me permite posicionar el puntero del archivo en la posición deseada.
4. Todas son correctas.
5. Ninguna es correcta.

#### 5-Desarrollar una clase que cumpla con el patrón de diseño Singleton:

public class MiSingleton {

private static MiSingleton instancia;

private MiSingleton() { //Inicializaciones internas que hagan falta }

public static MiSingleton getInstancia() {

if (instancia == null) {

instancia = new MiSingleton();

}

return instancia;

}

}

#### 6 - Tomando como base las clases DAO y AlumnoDAOTxt vistas en clase:

desarrollar el método insertar() y los atributos que el método utiliza. Suponer que se dispone de un método existe(Integer dni), no debe desarrollarlo, que le informa de la existencia o no de un alumno en el archivo. (Utilizar tipos de datos genéricos)

public void insertar(Alumno alu) throws DaoException {

if (exists(alu.getDni())) {

throw new DaoException("Alumno ya existe");

}

try {

raf.seek(raf.length());

raf.writeBytes(alu.toString() + "\n");

} catch (IOException ex) {

throw new DaoException("Error al insertar el alumno", ex);

}

}