1) 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)

```
//IMPORTANTE para el EXISTS, INSERT, DELETE y UPDATE
//--> Agregar al nombre del método el arrastre del THROWS
//--> Menos el EXISTS el resto devuelve VOID
//--> Agregar @Override porque se implementa de la clase abstracta de base DAO.java que tiene las firmas
@Override
public void Insertar(Alumno alu) throws DaoException
{
//Declara variable raf de RandomAccessFile. IMPORTANTE parametros: "Nombre del file", "rws" que es escritura/lectura
        RandomAccessFile raf = new RandomAccessFile("C:\\alumno.txt", "rws");
        try {
                //Valida si EXISTE ya otro DNI, si existe -> DaoException.
                if (existe(alu.getDni()))
                {
                  //Si el Alumno EXISTE se tira el error para "arriba" con el mensaje correspondiente y termina el método.
                    throw new DaoException("Alumno duplicado ==> "+ alu.getDni());
                }
                // .seek([posicion]]) : Se posiciona al final del archivo usando el .length()
                raf.seek(raf.length());
                // .writeBytes :ESCRIBE en el archivo, el dato que es la sobrecarga de alu.toString() + el separador de línea}}
                raf.writeBytes(alu.toString()+System.lineSeparator());
    }
    //El catch es para cualquier error ABRIENDO/LEYENDO el archivo
    catch (IOException ex) {
    //Logger puede no estar ya que es para escribir la consola.
        Logger.getLogger(AlumnoDaoTXT.class.getName()).log(Level.SEVERE, null, ex);
    //Acá se captura y se tira el error para "arriba"
      throw new DaoException("Error E/S ==> No se pudo crear el alumno"+
"("+ex.getLocalizedMessage()+")");
    }
  }
```

---

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();
	}
}
```

a) El objetivo de la variable ***posLinea*** es la de guardar la posición de fin de la línea para que ***archivoRandomAccessFile*** pueda continuar leyendo.
b) La línea de código ***archivoRandomAccessFile.seek(0)***; se utiliza para abrir el archivo de texto para comenzar a utilizarlo.
c) El método ***`getFilePointer()`*** de ***`RandomAccessFile`*** me permite posicionar el puntero del archivo en la posición deseada.
d) Todas son correctas.
e) Ninguna es correcta.

---

3) Completar el siguiente código para que compile sin errores, y para que en tiempo de ejecución no arroje ninguna excepción. No es necesario declarar la variable *"alu"*.

```
try {
	String sqlInsert = "INSERT INTO alumnos\n" +
		"(dni, apellido, nombre, fecha_nac, promedio, cant_mat_aprob)\n" +
		"VALUES (?,?,?,?);";
	prepareStatementInsert = conn.prepareStatement(sqlInsert);
	prepareStatementInsert.setInt(index++, alu.getDni());
	prepareStatementInsert.setString(index++, alu.getNombre());
	prepareStatementInsert.setDate(index++,
				MiCalendario.convert2SqlDate(alu.getFechaNac()));
	prepareStatementInsert.setInt(index++, alu.getCantMatAprob());

	prepareStatementInsert.executeUpdate();

} catch (SQLException ex) {
	Logger.getLogger(AlumnoDAOSQL.class.getName()).log(Level.SEVERE, null, ex);
}
```

```
try {

//LINEA FALTANTE: Declarar "conn" y Realizar la conexión a la BD
Connection conn = DriverManager.getConnection(url, user, password);
//LINEA FALTANTE: Declarar variable "prepareStatementInsert"
PreparedStatement prepareStatementInsert;
//LINEA FALTANTE: Declarar variable "index" que es la que cuenta el índice de parámetro
int index = 1;

//La Cantidad de PARAMETROS que recibe está mal, tiene 4 y son 6.
//Tienen que coincidir cant columnas con la cant de parametros.
//OTRO faltante podría ser el caracter "`" depende de cuan mañoso sea el profe.
String sqlInsert = "INSERT INTO `alumnos`\n" +
 "(`DNI`, `APELLIDO`, `NOMBRE`, `FECHA_NAC`, `PROMEDIO`, `CANT_MAT_APROB`)\n" +
                 "VALUES(?, ?, ?, ?, ?, ?);";

//LINEA OK
prepareStatementInsert = conn.prepareStatement(sqlInsert);

//Están mal como arma el statement, faltan columnas y no se corresponden en orden:
prepareStatementInsert.setInt(index++, alu.getDni());
prepareStatementInsert.setString(index++, alu.getNombre());
prepareStatementInsert.setDate(index++, MiCalendario.Convert2SqlDate(alu.getFechaNac()));
prepareStatementInsert.setInt(index++, alu.getCantMatAprob());

//CORRECTO Orden, tipo (setInt, setString, setDate y setDouble) y cantidad:
prepareStatementInsert.setInt(index++, alu.getDni());
prepareStatementInsert.setString(index++, alu.getApellido());
prepareStatementInsert.setString(index++, alu.getNombre());
prepareStatementInsert.setDate(index++, MiCalendario.Convert2SqlDate(alu.getFechaNac()));
prepareStatementInsert.setDouble(index++, alu.getPromedio());
prepareStatementInsert.setInt(index++, alu.getCantMatAprob());

//Está mal el tipo de Execute()
prepareStatementInsert.executeUpdate();

//CORRECTO (IDEM para INSERT/UPDATE/DELETE; los SELECT usan executeQuery() y
devuelve un ResultSet):
prepareStatementInsert.execute();

    } catch (SQLException ex) {
        Logger.getLogger(AlumnoDaoSQL.class.getName()).log(Level.SEVERE, null, ex);
    }
```

---

4) 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");
       }
}
}
```

?1 → Map
?2 → throws
?3 → DAO_TXT
?4 → filename
?5 → SQL_CONNECTION
?6 → DAOException

5) Desarrollar una clase que cumpla con el patrón de diseño **Singleton**

```
// SINGLETON: patrón que resuelve tener una sola/única instancia de una clase
// IMPORTANTE: propiedad "instance" y todos los métodos que necesite acceder son PUBLIC STATIC
// FUNCIONAMIENTO: si ya se creo la instancia (!= null), la devuelve.
// sino (== null) crea la nueva instancia.
// USO desde otra clase:
// -> EjemploClass claseEjemplo = EjemploClass.getInstance();
// -> claseEjemplo.Hacer1();
public class EjemploClass
{
        //Estrucutura básica de Singleton
        //Propiedad PRIVADA STATICA
        private static EjemploClass instance;
        //Método PUBLICO STATICO que resuelve la instancia de la clase
        public static EjemploClass getInstance() {
        if (instance == null) {
            instance = new EjemploClass();
        }
        return instance;
        }

        //ESTO ES OPCIONAL, solo para ejemplo.
        public static Hacer1() {
             System.out.println("Hacer algo.");
        }
}
```