# ÍNDICE

[ÍNDICE 1](#_Toc152366283)

[Enunciado 2](#_Toc152366284)

# Resumen de desarrollo práctica

Al principio he creado dos clases principales, una era la que tenía el JFrame del programa y la clase con todas las operaciones de cada uno de los ejercicios.

En el ejercicio 3 ya me di cuenta que quizás me sería más fácil realizarlo usando clases **POJO**. Las clases serían: **Entidad**, **Estudiante**, **Proyecto** y **Recursos**. Mientras que el resto de clases serían listas que interrelacionan las clases. No obstante, opté por realizarlo según consta el ejercicio a expensas de cómo se desarrollasen el resto de ejercicios para posteriormente refactorizar el código.

En el ejercicio 4, he podido emplear elementos de tipo estilo para dotar al informe de un aspecto más personalizado. Para hacer las bandas de colores hay que usar una expresión condicional y en el elemento **details** hay que usar un elemento **frame** que estará anidado dentro del elemento **band**.

Ya en el último ejercicio he tenido que crear las clases **POJO** para poder cargar más fácilmente los datos mediante el uso de **DefaultTableModel**. He creado otra clase **Logica**, que agrupa tres listas de cada una de las clases POJO.

La parte del control de errores y muestra de salida en el **JTextArea** ha ido evolucionado desde unos simples **try-catch**, el uso de **throws** para propagar los errores y realizar un tratamiento centralizado, extraer el código de control de errores a una clase especifica hasta crear un campo estático en la clase de control de errores para enlazar con el elemento **JTextArea**. De esta manera, se puede obtener una referencia de dicho objeto en cualquier clase que importe la clase **GestionErrores**.

Por último, he tenido que realizar un control de clases de java.util.Date y java.sql.Date. Esta última es una subclase de java.util.Date, permite asignarla a una clase java.util.Date con el peligro de usar métodos de esta última clase que no están implementados en java.sql.Date. Esto se debe a que ambas comparten el mismo nombre y causa un efecto shadowed de variables. Ejemplo:

Date fechaTemporal = fechaAlta;

// Comprobamos si la fecha es una instancia de java.sql.Date y la convertimos a java.util.Date

if (fechaAlta instanceof java.sql.Date) {

fechaTemporal = new Date(fechaAlta.getTime());

}

// Con el tipo convertido a java.util.Date si podemos usar métodos para poder usar la clase LocalDateTime

LocalDateTime ldt = LocalDateTime.*ofInstant*(fechaTemporal.toInstant(),ZoneId.*systemDefault*());

String fechaFormateada = ldt.format(dtf);

He optado por usar las clases **DateTimeFormatter** y **LocalDateTime** ya que son librerías más modernas.