

Proyecto Titanic — Elementos Destacables y Problemas Encontrados.

Integrantes del proyecto

- **Ángel Del Río García y Alejandro Rodríguez Rodríguez**
 - Módulo: PSP (Programación de Servicios y Procesos)
 - Curso: 2º DAM
-

Índice

1. [Elementos destacables del desarrollo](#)
 - [Arquitectura modular y extensible](#)
 - [Uso de patrones de diseño](#)
 - [Pruebas automatizadas con JUnit](#)
 - [Uso de Maven y Lombok](#)
 - [Generación de informes reales](#)
 - [Portabilidad y robustez](#)
 2. [Problemas encontrados y soluciones aplicadas](#)
 - [Dificultad para entender el enunciado y los objetivos del proyecto](#)
 - [Dificultad para pasar objetos entre procesos](#)
 - [Falta de experiencia con JUnit y pruebas automatizadas](#)
-

Elementos destacables del desarrollo

1. Arquitectura modular y extensible

El proyecto presenta una **estructura modular clara**, donde cada clase tiene una responsabilidad bien definida:

- **CreadorBote** → genera datos simulados.
- **BoteData** → encapsula la información de los botes.
- **GeneradorInforme** → organiza los resultados y produce el informe.
- **FormatoMarkdown** / **FormatoHTML** → definen la salida final.
- **ServicioEmergencia** → coordina el flujo completo del programa.

Esto hace que el código sea fácil de mantener, probar y ampliar (por ejemplo, sería muy sencillo agregar un nuevo formato de salida como PDF o CSV).

2. Uso de patrones de diseño

El proyecto aplica el patrón **Strategy** en la generación de informes. Las diferentes implementaciones de **FormatoInforme** (Markdown y HTML en este caso) permiten cambiar el formato de salida **sin modificar el resto del código**.

Se ha llegado a este uso de patrón de diseño mediante búsquedas en internet y aplicando el conocimiento aportado en los apuntes del **aula virtual de PSP**.

3. Pruebas automatizadas con JUnit

Se incluyen **tests unitarios y de integración** que garantizan el correcto funcionamiento del programa. Estas pruebas comprueban:

- La coherencia de los datos generados por **CreadorBote**.
- La correcta estructura y formato de los informes generados.
- Que el método **main()** de **ServicioEmergencia** se ejecuta sin errores.

Gracias a estos tests, el proyecto puede ejecutarse sin fallos en cualquier equipo, lo que es especialmente importante para la evaluación del profesor.

4. Uso de Maven y Lombok

El uso de **Maven** permite una **gestión profesional de dependencias**, facilitando la compilación y ejecución en cualquier entorno mediante comandos como **mvn clean test**.

Además, **Lombok** elimina la necesidad de escribir código repetitivo (como getters y setters), haciendo el código más legible y limpio.

5. Generación de informes reales

El sistema no solo simula datos, sino que también **genera informes finales reales** en disco, en formato **Markdown** en esta ocasión, ya que como bien se indica en el proyecto, la implementación del uso de **HTML** se hará en futuras actualizaciones.

Esto añade un resultado tangible al proyecto y demuestra la capacidad de combinar lógica de negocio con salida a fichero.

6. Portabilidad y robustez

El proyecto está diseñado para **funcionar correctamente en cualquier ordenador**, sin depender de configuraciones externas:

- No utiliza rutas absolutas.
- No requiere instalación manual de librerías.
- Todas las dependencias están declaradas en el **pom.xml**.

Esto garantiza que el programa sea totalmente portable y reproducible independientemente también del **sistema operativo**.

Problemas encontrados y soluciones aplicadas

1. Dificultad para entender el enunciado y los objetivos del proyecto

Problema: Al comienzo, tuvimos problemas para **comprender lo que se pedía exactamente en el ejercicio**. El planteamiento inicial del proyecto no era del todo claro, y no entendíamos cómo debíamos coordinar los distintos módulos a pesar de comprender el resultado final.

Solución: Mediante una revisión detallada del enunciado y consultas con el profesor conseguimos **entender la lógica general del proyecto**. Esto nos permitió establecer un plan de trabajo ordenado y repartir tareas de manera efectiva.

Aprendizaje obtenido: Aprendimos la importancia de **analizar bien el problema antes de empezar a programar**, así como de la utilidad de realizar un **diseño adecuado**.

2. Dificultad para pasar objetos entre procesos

Problema: Durante el desarrollo nos encontramos con la dificultad de **cómo pasar objetos entre procesos** sin depender de la consola (`System.out` y `System.in`). Al principio, intentamos hacerlo con simples flujos de texto, pero resultaba complicado mantener la estructura de los datos y era complicado de entender desde un punto externo de vista.

Solución: Tras investigar sobre la comunicación entre procesos en Java, descubrimos que podíamos utilizar `ObjectOutputStream` y `ObjectInputStream` para enviar y recibir objetos completos. Esto nos permitió **serializar los datos** y compartirlos directamente entre procesos de forma mucho más eficiente y limpia.

Aprendizaje obtenido: Comprendimos la importancia de la **serialización de objetos** en Java y cómo puede usarse para transmitir información compleja entre distintos componentes sin necesidad de depender de la salida estándar.

3. Falta de experiencia con JUnit y pruebas automatizadas

Problema: No teníamos experiencia previa escribiendo pruebas con **JUnit**, y al principio nos costó comprender cómo estructurarlas y ejecutarlas correctamente con Maven.

Solución: Estudiamos la documentación de JUnit 5 y ejemplos prácticos, lo que nos permitió crear **tests unitarios y de integración** funcionales. Además, verificamos que los tests se ejecutaban automáticamente con `mvn test`.

Aprendizaje obtenido: Descubrimos el valor de las **pruebas automatizadas** para garantizar la calidad y estabilidad del software, especialmente cuando otras personas lo ejecutan en entornos distintos.
