Titanic

Miguel Ángel Prieto, Manuel Pavon

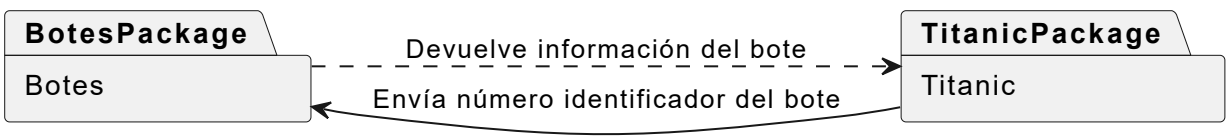
Índice

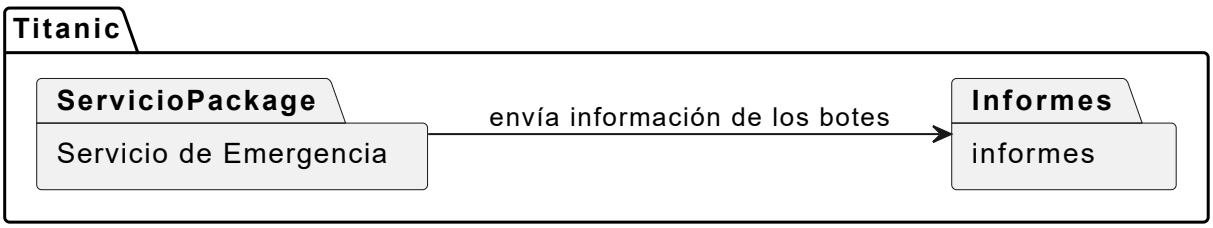
Analisis del Problema

- Hay que implementar una aplicacion que gestione los botes de emergencia teniendo en cuenta que los botes son 20, el tiempo de despliege de botes es finito y el recuento se realiza una vez el bote ya ha sido soltado, los botes tienes un minimo de 1 pasajero y un maximo de 100 divididos en hombres, mujeres y niños.
- Para calcular el número de pasajeros de cada bote se genera un número aleatorio de 1 a 100 y se reparte entre mujeres, hombre y niños y esto se manda al informe.
- el servício de emergencia genera un informe apartir de la información recibida de cada bote y generando un informe individual de cada bote con el recuento total y el desglose de hombres, mujeres y niños y al final un total general y un desglose del total de hombres, mujeres y niños.

Diseño de la solucion

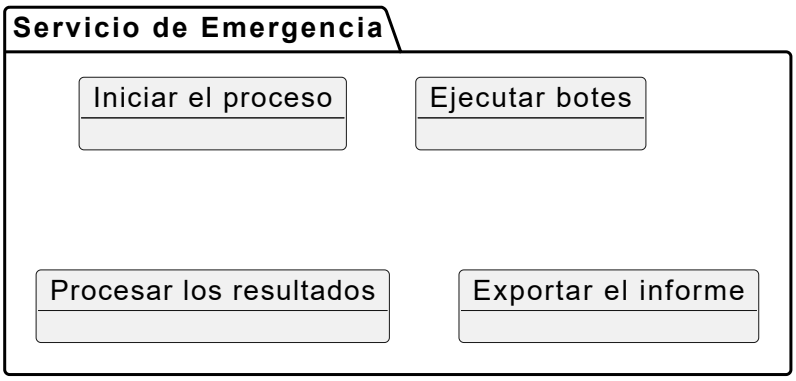
Arquitectura

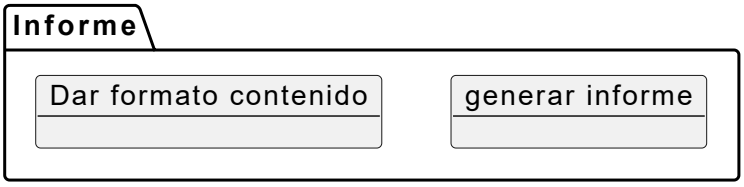




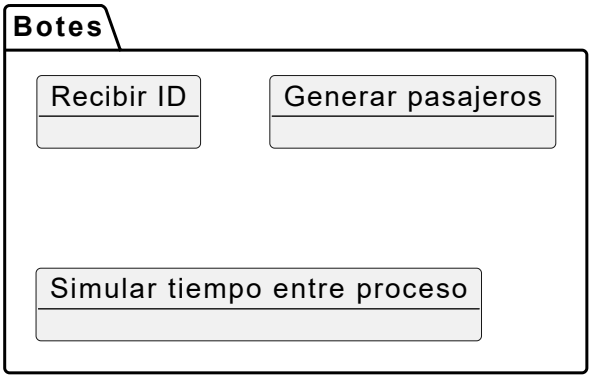
Componentes

Titanic





Botes



Protocolo de comunicacion

ServicioEmergencia pide el número de ID del bote mediante la entrada estándar (System.in). Luego lanza el proceso Botes usando Runtime.getRuntime().exec(), y le envía el ID del bote junto con el contenido del bote a través del OutputStream del proceso, que se conecta con la entrada estándar (System.in) de Botes. Botes recibe esa información por su InputStream, le da formato, y lo imprime por su salida estándar (System.out). ServicioEmergencia recoge esa salida a través del InputStream del proceso Botes, y la redirige al proceso Informes, enviándola por el OutputStream que conecta con la entrada estándar de Informes. Informes genera

el informe a partir de esa entrada, lo imprime por su salida estándar (System.out), pero no devuelve nada directamente a ServicioEmergencia, ya que el flujo termina con la salida del informe.

Plan de pruebas

El plan de pruebas se encuentra disponible como documento independiente. Puede consultarse en el siguiente enlace: Plan de pruebas

Manual de usuario

El Manual de Usuario se encuentra disponible como documento independiente. Puede consultarse en el siguiente enlace: Enlace al manual

Elemantos destacables del desarrollo

Botes

El paquete `Botes` está estructurado con una interfaz IBote y dos clases principales Bote y EjecutarBote.

- La interfaz **IBote** define el contrato que deben seguir todas las clases que implementen un bote, asegurando consistencia y modularidad. Tiene los siguientes métodos:

```
    String getId();
    Map<String, Integer> generarPasajeros() throws InterruptedException;
    void simularTiempoConteo() throws InterruptedException;
```

Esto permite que se puedan crear diferentes tipos de botes sin modificar el resto del sistema.

- **Bote** es la clase que representa un bote individual y se encarga de generar los pasajeros y simular el tiempo de conteo.
    - **Atributos principales**:
        - **id**: identificador del bote.
        - **MAX_PASAJEROS**: número máximo de pasajeros posibles.
        - **TIEMPO_MINIMO y TIEMPO_MAXIMO_AMPLIABLE**: para simular retrasos en el conteo.
    - **Método principal**:

      ```
          generarPasajeros(): Map<String, Integer>
      ```

      Genera un número aleatorio de pasajeros, distribuidos en mujeres, varones y niños, asegurando consistencia.
    - **Método auxiliar**:

```
simularTiempoConteo(): void
```

Introduce un retraso aleatorio para simular el tiempo real de conteo de pasajeros.

- **EjecutarBote** es la clase que permite ejecutar un bote de manera individual desde línea de comandos.
  - Recibe como argumento el ID del bote.
  - Genera la salida de pasajeros en formato clave=valor, lo que facilita su posterior procesamiento.
  - Salida típica:

```
codigo=1,mujeres=20,varones=30,ninos=10
```

Esta estructura permite un código modular y extensible, donde se pueden añadir nuevos tipos de botes o modificar la generación de pasajeros sin afectar el resto del sistema.

Servicio de emergencia

El paquete `ServicioEmergencia` está compuesto por una interfaz IServicioEmergencia y dos clases principales: **ServicioEmergencia** y **BoteProcess**. Su objetivo es coordinar la ejecución de los botes, procesar los datos obtenidos y generar un informe final con los resultados de la simulación.

IServicioEmergencia

```
void iniciarProceso() throws Exception
Map<String, Map<String, Integer>> ejecutarBotes() throws Exception
Map<String, Map<String, Integer>> procesarResultados(Map<String, Map<String,
Integer>> resultados)
void exportarInforme(List<Map<String, Integer>> datos) throws Exception
```

ServicioEmergencia

Clase principal que implementa la interfaz. Controla toda la lógica de la aplicación.

- **Constantes destacables**:
  - `NUM_BOTES`: número total de botes a procesar.
  - `FORMATO_ID`: estandariza nombres como B01, B02...
  - `TITULO_INFORME / EXTENSION_INFORME`: ruta y extensión final del archivo exportado.
- **Metodos principales**:

```
iniciarProceso()
```

Coordina todo el proceso: ejecutar botes, ordenar resultados y exportar el informe.

```
ejecutarBotes()
```

Lanza los botes uno por uno invocando la clase `BoteProcess`, almacenando el resultado de cada bote en un `HashMap`.

```
procesarResultados()
```

Ordena los resultados mediante un `TreeMap`, facilitando la lectura final del informe.

```
exportarInforme()
```

Transforma los datos en Markdown utilizando las clases del paquete Informes.

BoteProcess

Clase auxiliar encargada de ejecutar externamente cada bote como **proceso independiente**, simulando concurrencia real.

- Construye un comando java para ejecutar la clase EjecutarBote.
- Captura su salida estándar y de error.
- Valida resultados mediante:

  ```
  exitCode
  ```

  para identificar fallos durante la ejecución.
- `convertirLineaAHashMap()` Convierte la cadena clave=valor recibida en un mapa estructurado, verificando:
  - Formato correcto
  - Valores numéricos válidos
  - Línea no vacía Ejemplo de entrada válida:

```
codigo=5,mujeres=20,varones=30,ninos=10
```

Informes

El paquete informes esta escructurado con una interfaz **Exportador**, dos objetos **Informe y Seccion** y dos metodos **Generador informeRescate y ExportadorMarkown**.

Exportador

define el contrato que deben seguir todas la clases que quiran exportar un informe especifico a un formato especifico, esto permite implementa multiples formatos de exportacion sin modificar las clases principales del sistema. Tiene un metodo:

```
exportar(informe informe):String
```

Seccion

es una clase objeto que representa el contenido del informe y sus diferentes seciones. - Atributos: - titulo: String - contenido: String (puede estar en Markdown, HTML, etc.)

```
Devuelve el contenido del informe en el formato deseado.
```

Informe

es una clases objeto representa un informe completo sin darle formatos ni fuentes de datos externos. Tienen sus atributos y 2 metodos:

```
    agregarSeccion(Seccion seccion)
    getSecciones():List<Seccion>
```

Añaden una sección al informe y devuelve todas las seciones agregada.

- **GeneradorInformeRescate** se encarga de crear un objeto **Infrome** a partir de datos externos tiene un metodo principal que Recibe datos y devuelve un informe estructurado.

```
- generar(List datos): Informe
```

- **ExportadorMarkdown** una vez que se ha generado el informe, esta clase la trasforma en markdown e implemente la interfaz exportador y posee un metodo:

```
exportar(Informe informe):String
```

Esta estructura permite un facil mantenimiento y la posiblidad de añadir nuevos metodos que generer nuevos tipos de infromes como pfd word etc.

Problemas encontrados

Durante la creación de la aplicación de simulación de rescate del Titanic, se identificaron varios problemas y desafíos técnicos. A continuación se describen con sus causas, soluciones adoptadas y lecciones aprendidas.

1. Generación aleatoria de pasajeros

Problema:

- Se necesitaba generar un número aleatorio de pasajeros para cada bote, distribuidos en mujeres, varones y niños, garantizando que la suma coincidiera con el total.

Solución adoptada:

- Se generó primero el número total de pasajeros, luego el número de mujeres, varones y finalmente los niños se calcularon como total - mujeres - varones.
- Esto asegura coherencia en todos los registros.

2. Comunicación con procesos externos

Problema:

Cada bote se ejecuta como un proceso independiente de Java (Process) desde BoteProcess.

Desafío:

- Capturar correctamente la salida del proceso y manejar errores de ejecución.
- Evitar que un error en un solo bote detuviera la simulación completa.

Solución adoptada:

- Se implementó lectura de InputStream y ErrorStream.
- Se generaron mensajes de error claros y excepciones manejadas para cada fallo en la ejecución de un bote.

3. Generación de informes

Problema:

- Los datos generados por los botes debían consolidarse en un informe legible, en Markdown, con secciones por bote y resumen total.

Desafío:

- Asegurar que la información fuera clara, coherente y correctamente formateada.
- Permitir ampliaciones futuras (otros formatos de informe).

Solución adoptada:

- Se creó la clase Informes que contiene secciones (Seccion).
- Se implementó GeneradorInformeRescate para consolidar los datos y GenerarInformeMd para exportar en Markdown.
- Se definió un formato consistente y profesional, con título, fecha de ejecución y detalle por bote.

Conclusiones

De manera conjunta, este proyecto nos ha permitido profundizar en la ejecución de procesos externos en Java, así como en la lectura y gestión de flujos de entrada y salida. También hemos comprendido la importancia de diseñar el sistema de forma modular mediante interfaces, lo que facilita la escalabilidad, mantenibilidad y futura ampliación del código.

Uno de los mayores retos a nivel grupal fue evitar bloqueos al capturar la salida de los procesos Bote. Tras investigar diferentes enfoques, conseguimos establecer una comunicación robusta y fiable entre procesos. Además, desarrollamos la estructura para generar informes en Markdown y elaboramos la documentación necesaria para comprender el funcionamiento de la aplicación.

La coordinación entre los miembros del equipo fue positiva, realizando una buena distribución de tareas y revisiones conjuntas de código. Como posibles mejoras futuras, valoramos incorporar exportación a otros formatos, como PDF, y paralelizar el conteo mediante hilos para simular de forma más realista el despliegue de los botes.