Titanic

Creado por:

- Alberto Ubeda-Portugues
- Carlos Hermoso Delgado

Indice

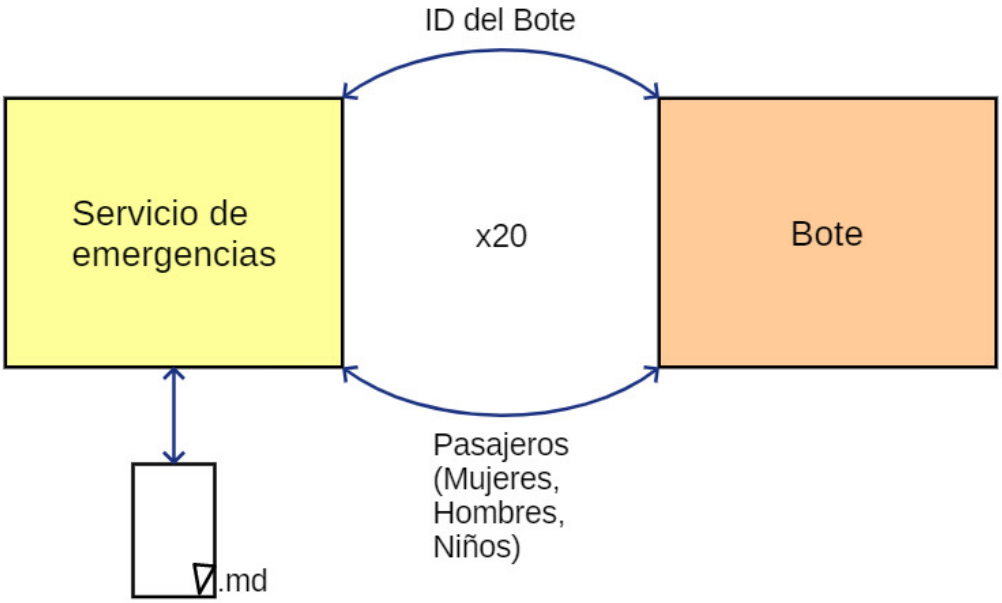
1. Análisis del problema

Se nos plantea simular los botes salvavidas del Titanic. Por cada bote se selecciona un número aleatorio de pasajeros del 0 al 100 y se les clasifica en mujeres, hombres o niños. El servicio de emergencia es el encargado de embarcar los botes y asignarles un número que va del B00 al B19. Los botes recuentan sus pasajeros y pasan el recuento al servicio de emergencia. Este lo recoge y, tras embarcar todos los botes, escribe un documento con todo el recuento completo.

2. Diseño de la solución

Arquitectura

El proyecto contará con dos partes diferenciadas: el servicio de emergencia y el bote. El primero se encargará de ejecutar el bote, dándole un ID, y este generará y enviará los datos de los pasajeros. Posteriormente, el servicio de emergencia generará un documento en Markdown.

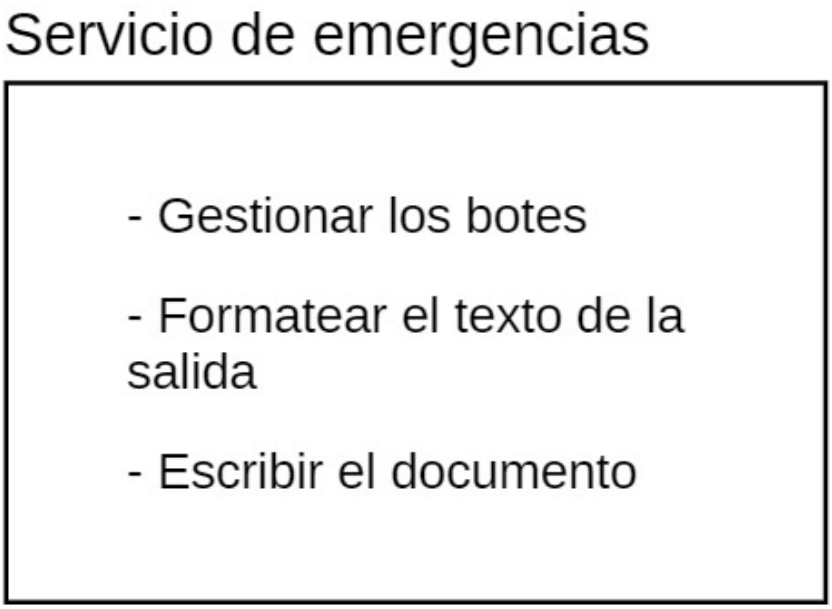


Componentes

- Servicio de emergencias

El servicio de emergencias tendrá un programa principal (*Main*) llamado `ServicioEmergencia` que será el que ejecutaremos.

Además, tiene varias funcionalidades: gestionar los botes, darle el formato deseado al texto de salida y escribir ese texto en un documento.



> **Gestión de los botes**(`gestion_botes`):

La gestión de los botes se encargará de ejecutar el comando de los botes y de recibir la salida en forma de *String*.

> **Gestión del formato**(`gestion_formato`):

La gestión del formato contará con una enumeración con los diferentes formatos que soporte el programa, MD y HTML; contará con una factoría de formateadores que, dependiendo del formato, creará un formateador u otro.

Los formatadores serán los encargados de darle formato al texto de salida. Estos contarán con maneras de incluir los elementos necesarios para el documento final: título, fecha, cuerpo del documento (información de cada bote) y la información total recogida.

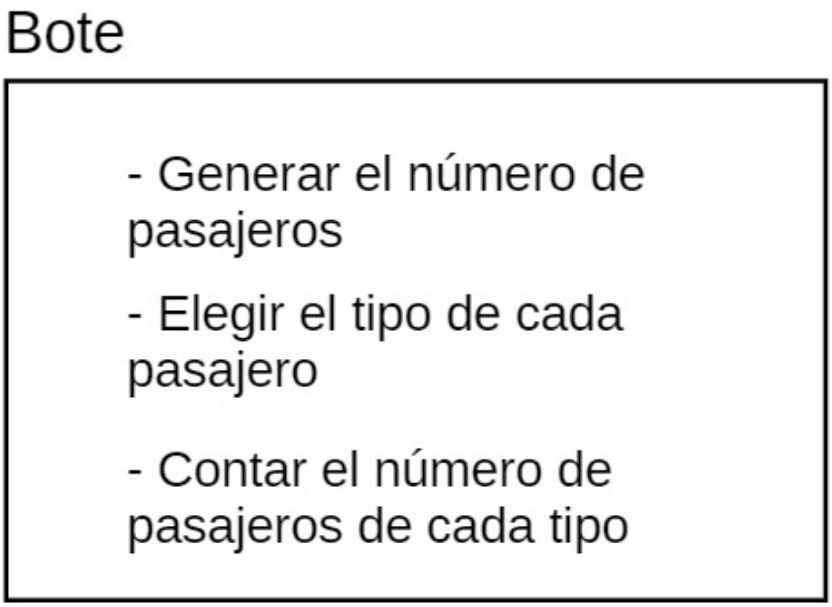
> **Gestión del documento** (`gestion_documento`):

Finalmente, la gestión del documento será la encargada de, mediante la ruta y el texto final con su formato correspondiente, crear el archivo con la información.

- Bote

El bote también contará con su propio Main llamado `Bote` que será el que ejecutará el servicio de emergencias, lanzándolo como un proceso.

Este debe generar un número de pasajeros de forma aleatoria, asignarles un tipo entre mujeres, hombres y niños y contar cuántos pasajeros de cada tipo hay.



> **Generar pasajeros**:

Para generar el número aleatorio de pasajeros, el bote utiliza la clase `Math` de Java y el método `random` de la misma, parseado a int y multiplicado por 100, el número máximo de tripulantes que puede tener un bote.

> **Dar tipo a los pasajeros**(*obtenerTipo()*):

A cada pasajero se le asignará un tipo, como ya se mencionó antes. Para hacerlo, se vuelve a utilizar el `random` como en el punto anterior, multiplicado esta vez por 5, por el número máximo de opciones. Se usa el 5 para darle una probabilidad mayor a que salga hombre o mujer y una menor a los niños; si salen 0 o 1, será mujer; 3 y 4, hombre; y 2, niño.

> **Contar**(*contar()*):

Ahora, con los tipos asignados a cada pasajero, podemos contar qué cantidad de cada uno hay por bote.

Protocolo de comunicación

El servicio de emergencias será el que nosotros ejecutemos. Este utilizará `Runtime` para ejecutar el bote. Este le pasará el id del bote, del B00 al B19, y de vuelta enviará un String con el id de nuevo, el número de pasajeros totales y el número de cada tipo de pasajero, todo junto y separado únicamente por `:`.

Plan de pruebas

Las pruebas que aplicaremos serán divididas por clases, generando una por cada uno de los métodos principales que utilizará el *Main* para ejecutar normalmente el problema, y comprobaremos si todo devuelve los valores esperados. Para el bote, al no tener más que una clase, aplicaremos los test a las dos funciones que esta tiene.

3. Manual de usuario

Este programa no cuenta con una interfaz gráfica. Para ejecutarlo tenemos dos formas. Podemos situarnos en nuestro IDE en el archivo `ServicioEmergencia.java` y ejecutar el código dándole al play directamente, o podemos dirigirnos a nuestra terminal, situarnos en la carpeta principal del proyecto y ejecutar el comando `java src/main/java/es/etg/dam/psp/servicio_emergencia/ServicioEmergencia.java`. Si es la primera vez que ejecutamos el programa y lo hacemos de esta manera, es posible que se necesite realizar el proceso anterior, pero cambiando el comando `java` por `javac` para crear el archivo `class`. Posteriormente, volvemos a ejecutarlo usando `java` y se arrancará el programa con normalidad.

El programa muestra por la terminal de cualquiera de las dos maneras un pequeño seguimiento de lo que está ocurriendo en todo momento, indicando cuándo se han empezado a embarcar los botes, cuándo se está ejecutando cada uno de ellos y cuándo se recibe la información de los mismos. Al finalizar, mostrará un mensaje indicando que el documento fue escrito de manera correcta y finalizará.

Se puede comprobar el resultado final del documento en la carpeta de resources dentro de `src/main/resources`.

4. Elementos destacables del desarrollo

El proyecto sigue la línea de lo visto en clase. Se ha separado y paquetizado todo de manera que quede clara la división de las funcionalidades. Hemos intentado crear un código lo más reutilizable posible y evitando la repetición de código. Se han implementado varias interfaces para apoyar este hecho y se ha reutilizado una de ellas de ejercicios anteriores, la interfaz `Lanzador` para ejecutar el comando de los botes. Además, hemos utilizado las enumeraciones para la utilización de una factoría de formateadores. Por ahora solo soporta MD, aunque se nombre HTML para darle variedad, pero en el futuro siempre sería posible agregar cualquier formato y optimizar la detección del formato a partir de la ruta del archivo.

5. Problemas encontrados

- Al comienzo del desarrollo se realizó un diseño donde el apartado dirigido a los botes contaba con diferentes clases para cada uno de los métodos presentados en el mismo. Al utilizarlos de esta manera y llamar al programa Main como proceso, acababa devolviendo como valor de salida 1, es decir, un error al ejecutarse. Por ello se optó por simplificar este apartado, solucionando el problema.
- Tuvimos algunos problemas al realizar los test, ya que no conseguíamos darles un enfoque que nos ayudase en el desarrollo. Al final optamos por tomar la solución más breve, creándolos para los métodos más importantes que utilizaríamos.

6. Conclusiones individuales

Alberto:

A nivel personal, reconozco que aún tengos aspectos que mejorar en programación, no por falta de conocimiento, sino por el tiempo limitado de práctica del que dispongo sumado a ciertas carencias producidas por la falta de profesor el curso pasado.

En el ámbito grupal, destaco el talento de Carlos y la paciencia en su colaboración. Su capacidad y aportes me han permitido aprender y mejorar por lo que ha sido una buena experiencia compartir tiempo de trabajo para

sacar el proyecto adelante.

Carlos:

Personalmente, ha sido gratificante el desarrollo de un proyecto más exigente de lo que estaba acostumbrado. Tras el primer año sin profesor de programación durante gran parte del curso, muchos de los conocimientos y prácticas habituales no nos fueron transmitidos. Con este proyecto creo que hemos avanzado notablemente y cubierto algunas de las carencias que pudiésemos tener.

Por otro lado, ha sido muy gratificante trabajar con Alberto en este proyecto. Creo que hemos logrado compenetrarnos muy bien y trabajar a la par.

7. Anexos

- Repositorio de Github:

https://github.com/TingusC/Titanic