TAREA 1 - ELO329

Paulo Vargas, Henriette Stromsvaag, Alonso Toro

La tarea se desarrolló por GitHub. El desarrollo se dividió por stages:

STAGE 1: https://github.com/alonsoo7/tarea1_elo329/tree/stage1

STAGE 2: https://github.com/alonsoo7/tarea1_elo329/tree/stage2

STAGE 3: https://github.com/alonsoo7/tarea1_elo329/tree/stage3

STAGE 4: https://github.com/alonsoo7/tarea1_elo329/tree/main

STAGE EXTRA: https://github.com/alonsoo7/tarea1_elo329/tree/StageExtra

Documentación del desarrollo

Stage 1

Se comenzó con el stage 1. Para esto se utilizó el código base del profesor y se implementaron los siguientes métodos y constructores:

- subscribe, notify y constructor en Topic.
- Constructor en Subscriber y en Component.
- update y constructor en Follower.
- Constructor y publishNewEvent en Publisher.

Además, se creó el primer Makefile. Con esto, se logró el caso básico donde un Streamer envía notificaciones que son almacenadas por solo un Suscriptor de tipo Seguidor. Las clases fueron inicializadas desde T1Stage, y la entrada de mensajes se realizó por teclado.

Stage 2

Se continuó con el stage 2, donde se implementó la clase Recorder. Para esto, se basó en la estructura de la clase Follower, pero con la modificación de que la salida ahora se graba en un formato CSV. Se implementaron los siguientes métodos y constructores:

- Constructor en Recorder.
- Método para grabar las posiciones en formato CSV.
- Método subscribe para que el Recorder pueda recibir notificaciones.

Además, se adaptaron las clases **Publisher** y **Subscriber** para trabajar con la nueva funcionalidad de almacenamiento en CSV. Se creó el nuevo Makefile para este stage, lo que permitió que el Recorder recibiera las posiciones publicadas por el Streamer y las

almacenara correctamente en un archivo CSV. Esto facilitó el caso básico donde un GPS publica posiciones y el Recorder las almacena de manera persistente.

El sistema fue probado y validado mediante la simulación de entradas, y la grabación en el archivo CSV fue verificada.

Stage 3

Siguiendo, en el stage3 se generalizó el simulador para configurar un Publicador y dos seguidores usando un archivo de configuración.

Se realizan dos lecturas del archivo de configuración: la primera identifica al publicador y crea su instancia junto con el tópico en el Broker. Esto asegura que el tópico exista antes de que los suscriptores intenten suscribirse. La segunda lectura configura a los dos suscriptores, validando que el tópico coincida y que haya exactamente dos. Así, se evita problemas de ejecución por un orden incorrecto en la configuración.

La entrada por teclado debe seguir el formato (streamer) (mensaje) y solo se publican mensajes si el streamer es el nombre del publicador registrado en el archivo de configuración; de lo contrario, se arroja "Unknown Publisher".

Stage 4

En el Stage 4, presentamos nuestro simulador. Para ello, reutilizamos el trabajo de las etapas anteriores, tanto en la definición de clases como en los métodos implementados. Lo primero fue crear la nueva clase solicitada: el suscriptor "Monitor". Su funcionamiento se basó en el del "Registrador", con la diferencia de que el Monitor solo registra posiciones cuya distancia al origen sea mayor o igual a 500, considerando que las coordenadas X e Y varían entre 0 y 500. Luego, el siguiente trabajo fue implementar la generalización de los archivos de configuración. Para esto se realizó:

- *Implementación de revisión para abrir el archivo de configuración*.

Ya que requeríamos leer múltiples posibles archivos de configuración (pero siempre respetando que sean del tipo por cada línea:

publicador <nombre> <tópico>

suscriptor <tipo> <nombre> <tópico> <archivo>

El tipo de suscriptor puede ser Monitor, Seguidor o Registrador. El problema detectado en la lógica inicial fue que se trabajaba con configuraciones simples y restringidas. Para esta entrega, se implementó una solución que consiste en realizar dos lecturas al archivo de configuración: primero se cargan los publicadores y luego los suscriptores, a estos últimos diferenciándolos por tipo. Esta decisión se tomó porque, en el patrón existe un broker, con el cual los tópicos se crean al inicializar un publicador; así, si se leía antes un tipo de suscriptor cuyo tópico que suscribía aún no existía, se generaban errores. Con este enfoque, se evita la dependencia del orden en el archivo de configuración para el funcionamiento del programa.

- Implementación de la lógica mencionada en el enunciado: *Si el nombre de un suscriptor está en más de una línea, significa que desea suscribirse a más de un tópico. Un publicador sólo publica en un tópico.*

Para implementar la funcionalidad, se diseñó la siguiente solución: en la clase Component (base de Subscriber y Publisher) se añadió un ArrayList para manejar múltiples tópicos y un atributo String para un único tópico, permitiendo compatibilidad con la inicialización básica y extensión a múltiples tópicos. Durante la lectura del archivo de configuración, si un suscriptor ya existe, se agrega el nuevo tópico a su lista mediante un método de Component. Para los publicadores, se sobrescribió este método en Publisher dejándolo vacío, de modo que solo puedan publicar en un único tópico correspondiente al cual se inicializan; si se intenta agregar otro, se muestra un error en pantalla.
Además, como se discute en "Dificultades", se decidió permitir que distintos publicadores compartan el mismo tópico, por lo que al inicializar un publicador se verifica si el tópico ya existe para asignarlo mediante el broker en vez de crear uno nuevo.

También fue necesario modificar la clase Broker, implementando un método específico llamado subscribeToTopic(sub, topic), que permite suscribir un suscriptor a cualquier tópico, independientemente del tópico con el que fue instanciado. En el código base original, un suscriptor solo podía suscribirse al tópico inicial. Con esta modificación, el Broker ahora puede gestionar la suscripción de un mismo suscriptor a múltiples tópicos de manera individual.

De esta manera, se logró implementar el diseño del patrón Seguidor simplificado solicitado en la tarea. El simulador permite cargar un archivo de configuración con el formato por línea:

publicador <nombre> <tópico>

suscriptor <tipo> <nombre> <tópico> <archivo>

Luego, mediante la consola, se hace que la entrada sea del tipo:

<nombre_publicador> <mensaje o coordenadas>

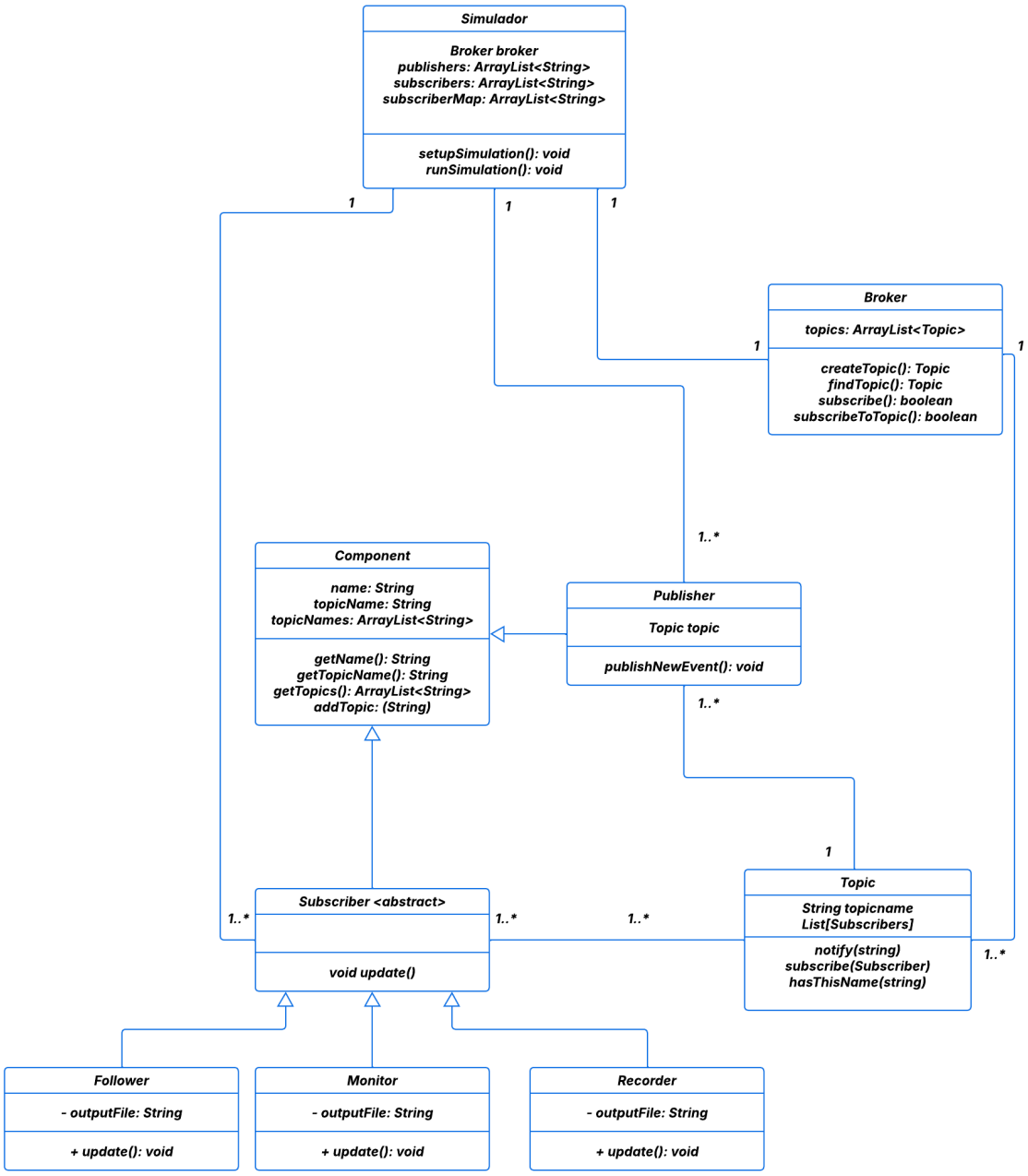
para enviar mensajes a los suscriptores. Los suscriptores de tipo Registrador o Monitor reciben coordenadas en formato CSV, mientras que los de tipo Seguidor reciben texto, almacenándose en cada caso en los archivos de salida indicados.
Se permite que un suscriptor esté suscrito a uno o más tópicos (repitiendo su nombre en el archivo de configuración) y que varios publicadores compartan un mismo tópico, respetando siempre que cada publicador publique en un único tópico

Estructura final y diagrama UML:

- Se creó una clase base Component, que proporciona la funcionalidad común para publicadores y suscriptores, incluyendo el manejo de múltiples tópicos mediante un ArrayList.
- La clase Publisher permite publicar mensajes en un único tópico. Para garantizar esto, se sobrescribió el método de añadir tópicos dejándolo vacío.
- La clase Subscriber sirve como base para los tres tipos de suscriptores implementados:
    - Follower: recibe notificaciones de texto (por ejemplo, de streamers).
    - Recorder: registra posiciones GPS en formato CSV.
    - Monitor: registra posiciones solo cuando el móvil supera un umbral de distancia al origen, utilizando el mismo formato CSV que Recorder.
- El Broker actúa como intermediario, gestionando las suscripciones a tópicos y la distribución de mensajes. Se modificó para permitir que un mismo suscriptor pueda estar suscrito a múltiples tópicos y recibir mensajes de cada uno.
- La clase tópico implementa métodos para acceder a los atributos de este y realizar la notificación a los suscriptores.

Además, se implementó manejo de excepciones en las clases Monitor y Recorder para validar que los mensajes de posición contengan únicamente números, y se incorporó control de errores para casos de configuración incorrecta en los archivos de entrada.

En la siguiente figura, se presenta un diagrama tipo UML en el que se observan las relaciones entre las clases especificados en la tarea, junto con los atributos y métodos que viven en cada una, donde se aprecia lo explicado anteriormente.



Stage Extra

Se desarrollo un nuevo tipo de suscriptor llamado “Contador”, cuya función es registrar cuántos mensajes recibe a lo largo del tiempo y poner esa información a un archivo CSV y se ve en consola.

Se creó una nueva clase llamada Contador, que hereda de Subscriber y agrega los siguientes componentes adicionales:

- Un contador interno (count), que se incrementa cada vez que se invoca el método update.
- Una marca de tiempo de inicio (startTime), capturada al momento de arrancar el simulador.
- Un flujo de salida (PrintStream out), utilizado para escribir registros en un archivo CSV en el formato <segundos>, <número de mensajes>.

El método update(String topic, String message) de la clase Contador realiza las siguientes acciones:

1. Incrementa el contador interno count.
2. Calcula el tiempo transcurrido en segundos desde que arrancó el simulador.
3. Escribe en el archivo CSV una línea que contiene el par (segundos, cantidad de mensajes).
4. Muestra en consola un mensaje informativo con el siguiente formato:

“Contador_1 ha recibido 3 mensajes en sus tópicos suscritos hasta el segundo 5.”

Para integrar el Contador, se realizaron algunas modificaciones en la clase Simulador.

- Se agregó un nuevo campo startTime, el cual se inicializa al partir el programa.
- Durante la segunda leída del archivo de configuración cuando se leen los suscriptores, el simulador identifica aquellos de tipo "Contador" y los instancia, pasándoles como argumento adicional el startTime.

En el archivo de configuración se debe explicitar igual que los otros tipos de suscriptor, por ejemplo, como:

suscriptor Contador Contador_1 Notificaciones_1 contador1.csv

Esto permitió que Contador_1 quedara suscrito al mismo tópico que otros suscriptores, pero registrando la cantidad de mensajes respecto al tiempo en un archivo CSV llamado contador1.csv.

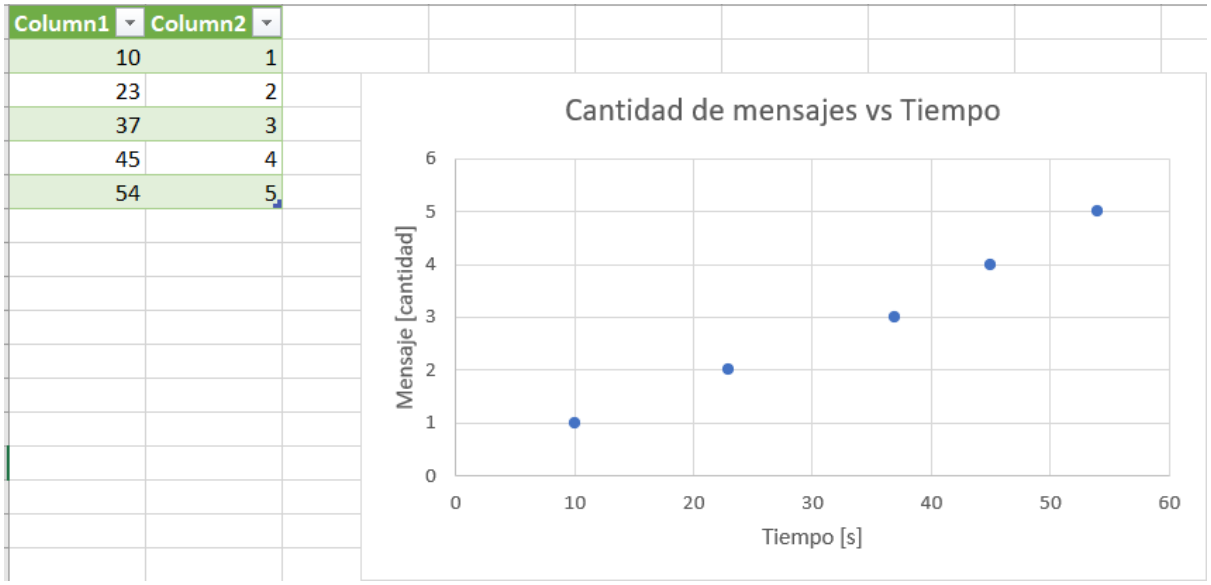
Finalmente importamos el CSV a Excel, pusimos el tiempo en el eje X y la cantidad de mensajes en el eje Y. Con eso armamos un gráfico puntual que muestra cómo van llegando los mensajes con el tiempo. Sirve para ver que el contador efectivamente está haciendo su trabajo.

Ejemplo

Salida por consola:

```
PS C:\Users\Hp\Desktop\USM 2025-1\POO\ESTA-SI\tarea1_elo329> java -cp . Simulador config4.txt
STREAMERS DISPONIBLES:
Streamer1
Streamer2
GPS_1
Simulador listo. Formato de evento:
<PublisherName> <mensaje o coordenadas>
Streamer1 HOLA MI GENTE, COMENZAMOS EL DIRECTITO
Contador_1 ha recibido 1 mensajes en los topicos que se suscribio hasta el segundo 10.
Streamer2 MI GENTE PRENDIMOS FARMEANDO EN DBD
Contador_1 ha recibido 2 mensajes en los topicos que se suscribio hasta el segundo 23.
Streamer1 MI GENTE VENGAN RAPIDO QUE SE VA ACABAR
Contador_1 ha recibido 3 mensajes en los topicos que se suscribio hasta el segundo 37.
Streamer2 NO LLEGO NADIE, SE CIERRA DIRECTO
Contador_1 ha recibido 4 mensajes en los topicos que se suscribio hasta el segundo 45.
Streamer1 CHAO MI GENTE, NOS FUIMOS
Contador_1 ha recibido 5 mensajes en los topicos que se suscribio hasta el segundo 54.
PS C:\Users\Hp\Desktop\USM 2025-1\POO\ESTA-SI\tarea1_elo329>
```

Gráfico con el archivo csv:



Dificultades

Las principales dificultades se dieron al momento de generalizar el archivo de configuración, en el último stage al unificar el trabajo anterior.

1. **Publicadores y suscriptores fuera de orden**

Nos dimos cuenta de que, si en el archivo de configuración un suscriptor se suscribía a un tópico antes que un publicador se configurara para dicho tópico, no lograba suscribirse al tópico, ya que este aún no existía. Aunque no se generaban errores explícitos en consola, los archivos de salida quedaban vacíos, ya que los métodos update() nunca se ejecutaban.

¿Cuál fue nuestra solución?

En el método SetupSimulator() del simulador, modificamos la carga de configuración: primero se lee todo el archivo Config4.txt, luego se recorre la lista agregando solo los publicadores (creando así los tópicos mediante el Broker), y finalmente se recorre nuevamente para procesar los suscriptores. Además, se explicitó un error que termina la ejecución si se intenta colocar un archivo de configuración con solo suscriptores sin publicador.

De esta forma, eliminamos la necesidad de un orden específico en el archivo de configuración y aseguramos que todos los suscriptores puedan vincularse correctamente a sus respectivos tópicos.

2. **Soporte de múltiples tópicos por suscriptor vs. único tópico por publicador**

La pauta especifica que los suscriptores pueden suscribirse a múltiples tópicos, pero un publicador a solo uno; Por ello surgía el problema con la lógica del código base donde siempre instanciaba un nuevo componente al leer la línea del archivo de configuración, lo que impedía manejar la multiplicidad de tópicos para un suscriptor. Entonces sucedía que, si se tenía un suscriptor en distintos tópicos, este no hacia efecto ya que se instanciaba cada vez que se leía por la línea del archivo de configuración, al igual que para el caso de publicador, sucedía que no teníamos una limitación para arrojar un error si aparecía en distintas líneas.

¿Cuál fue nuestra solución?

Comenzamos incorporando una lista de tópicos por componente y listando los componentes registrados en listas en la clase Simulador. En lugar de usar un solo atributo String topicName, que mantuvimos para la inicialización, definimos List<String> topicNames para permitir que los suscriptores puedan suscribirse a múltiples tópicos.

Al leer cada línea del archivo de configuración, para los distintos tipos de suscriptores verificamos:

- Si no existe, se crea la instancia del componente y se añade el tópico.
- Si ya existe, no se crea ni se reinicia el objeto, sino que simplemente se agrega el nuevo tópico a su lista topicNames.

Para los publicadores, se mantiene la regla de "un solo tópico". Por lo tanto, si encontramos una línea para un publicador ya existente, mostramos un mensaje indicando que "este ya existe" y cerramos el proceso, ya que se considera que el archivo de configuración no es válido.

Con estas modificaciones, conseguimos que los suscriptores puedan suscribirse a múltiples tópicos, mientras que cada publicador solo tenga asignado un único tópico.

3. Relación Muchos a 1 entre publicador y tópico

Al revisar el enunciado, notamos que el formato de salida solo menciona el tópico del mensaje, sin identificar al publicador. Esto nos planteaba un problema: si varios publicadores compartieran un tópico, no sería posible saber quién envió cada mensaje, pero se indica que cada publicador publica en un solo tópico. En general el problema es que no sabíamos si un tópico puede ser compartido entre publicadores.

¿Cuál fue nuestra solución?

En el UML definimos una relación muchos a uno (1...*:1) entre publicador y tópico: al instanciar un publicador, se verifica si el tópico ya existe; si no, se crea. Esta decisión se basó en que: (1) cada publicador tiene un único tópico, (2) los mensajes solo indican el tópico y (3) se busca mantener la coherencia del patrón de diseño mostrado:

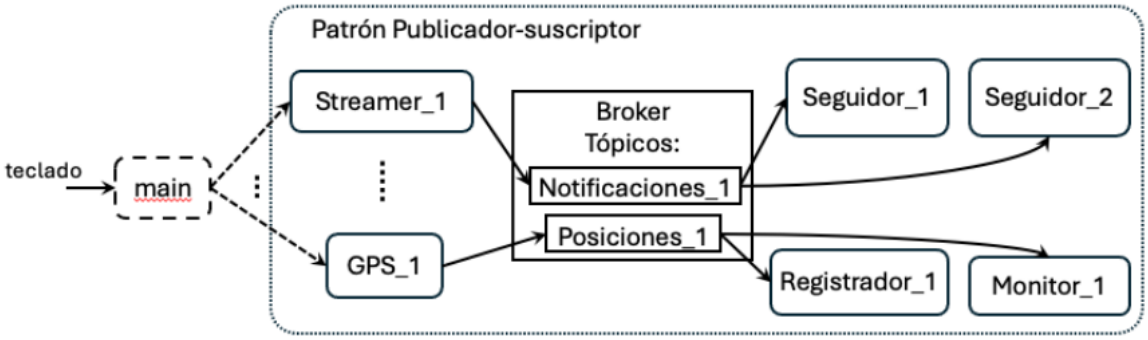


Figura 2. Simulacion del patron en enunciado de la tarea.

Así, asumimos que un tópico puede ser compartido entre varios publicadores. Es decir, distintos publicadores pueden publicar en el mismo tópico.