

Ejercicio de patrones de diseño, programación funcional y *streams* con Python

Alejandro Fernández Sánchez
alejandro.fernandezs@edu.upct.es
49274537G

9 de mayo de 2024

En este documento se cumple con el apartado **c** de la documentación a entregar y con el apartado **1.a** de las tareas a realizar. Se pide hacerlo en word pero se me ha dado permiso para hacerlo en L^AT_EX.

1. Patrones de diseño implementados

En esta sección se pide una justificación de cada patrón de diseño implementado.

1.1. Patrón R1 - Singleton

Se pide: “Debe de existir una única instancia del sistema que gestione todos los componentes y recursos del entorno IoT”. El comienzo de la oración (“debe de existir una única instancia”) ya me indica que el patrón de diseño *singleton* tiene sentido en este contexto. La clase a la que he asociado este patrón de diseño se llama *Sistema*.

Lo que busco es tener un objeto, y solo uno, en mi programa que se encargue de recibir los datos del sensor de temperaturas y hacer lo que tenga que hacer con ella. Este objeto será el responsable de enviar la temperatura a donde haga falta para que los pasos definidos en *R3* tengan lugar.

1.2. Patrón R2 - Observer

Este patrón de diseño está implementado en dos sitios.

1. Primero trataré a una clase llamada *SensorTemperatura* como la clase publicadora. Esta clase utilizará el software *Apache Kafka* para publicar las temperaturas y su *timestamp* asociado. La clase que recibirá esta información será la ya mencionada *Sistema*, que se encargará de mandar hacer los cálculos necesarios para que todo funcione perfectamente.
2. Después, la propia clase *Sistema* volverá a actuar de clase publicadora, mandando la temperatura a los objetos que necesite para que lo que se pide en el enunciado se realice. La clase que actúa de observador en este caso es la clase *CalculaEstadísticos*, la cual actúa de primer eslabón en la cadena de responsabilidad que se explica en la siguiente subsección.

1.3. Patrón R3 - Chain of Responsibility

La idea detrás de este patrón de diseño en esta implementación es que, tal y como se pide en el enunciado, suceda lo siguiente, y en ese orden:

1. Se calculan unos estadísticos. Más sobre esto en la subsección siguiente. Clase *CalculaEstadísticos*.
2. Se comprueba que la temperatura no alcance cierto valor. Clase *ComprobadorUmbral*.
3. Se comprueba si, durante los últimos 30 segundos, la temperatura ha aumentado más de 10 grados centígrados. Clase *ComprobadorDelta*.

El patrón de *Chain of Responsibility* tiene sentido porque se pide que se ejecuten estas acciones una detrás de la otra. Así, será el objeto de la clase `CalculaEstadisticos` el que se encargará de que el objeto de la clase `ComprobadorUmbral` haga su papel una vez se hayan calculado todos los estadísticos. De igual manera, será el objeto de la clase `ComprobadorUmbral` el que se encargará de que el objeto de la clase `ComprobadorDelta` haga su papel una vez haya terminado su trabajo.

Todas estas clases heredarán de una clase `ManejaTemperaturas`, clase que contiene la estructura que estas tres clases deben tener en común para cumplir con el esquema del patrón de diseño *Chain of Responsibility*.

1.4. Patrón R4 - Strategy

Se pide que se deben tener diferentes estrategias para computar los estadísticos. Para la implementación, será el objeto de la clase `CalculaEstadisticos` el que se encargará de contener objetos especializados en las distintas estrategias de cálculo de estadísticos. Estos objetos especializados en las distintas estrategias serán todas instancias de clases que heredan de `CalculadoraEstadistico`.

2. Git - Sistema de control de versiones

En esta sección se pide que se añadan capturas de pantalla mostrando los comandos git ejecutados para realizar las siguientes acciones:

2.1. Creación de dos tags

Se pide definir dos tags y se deja a criterio del estudiante en qué momento del proyecto realizarlos.

- Primer tag. **TODO**.
- El segundo tag se ha creado cuando he terminado todo y estaba a punto de prepararlo todo para la entrega. Lo he llamado `entrega`.

TODO

2.2. Creación de un repositorio público en GitHub

Se pide crear un repositorio público en GitHub que se llame `pcd_entregable2_alejandro` (en mi caso).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * **Repository name ***

aleferu / pcd_entregable2_alejandro

pcd_entregable2_alejandro is available.

Great repository names are short and memorable. Need inspiration? How about [literate-meme](#) ?

Description (optional)

Ejercicio de patrones de diseño, programación funcional y streams con Python.

☒ **Public**
Anyone on the Internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

You are creating a public repository in your personal account.

Create repository

Para enlazar el repositorio local y el online se ha utilizado el siguiente comando:

```
~/code/python/practicas-pcd/entrega-2  
$ git remote add origin git@github.com:aleferu/pcd_entregable2_alejandro.git
```

2.3. Repositorio público - GitHub

En esta sección se pide indicar la url del repositorio en github generado. El repositorio se encuentra en el siguiente enlace:

https://github.com/aleferu/pcd_entregable2_alejandro/

Su estado inicial era:

