

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

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

26 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<sup>A</sup>T<sub>E</sub>X.

## 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`. Son:

- `CalculadoraMedia`
- `CalculadoraModa`
- `CalculadoraMax`
- `CalculadoraMin`
- `CalculadoraCuasiVar`
- `CalculadoraMediana`

## 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.

1. El primer tag se ha creado cuando se ha conseguido una versión usable del código por primera vez. Se ha llamado `1.0`.

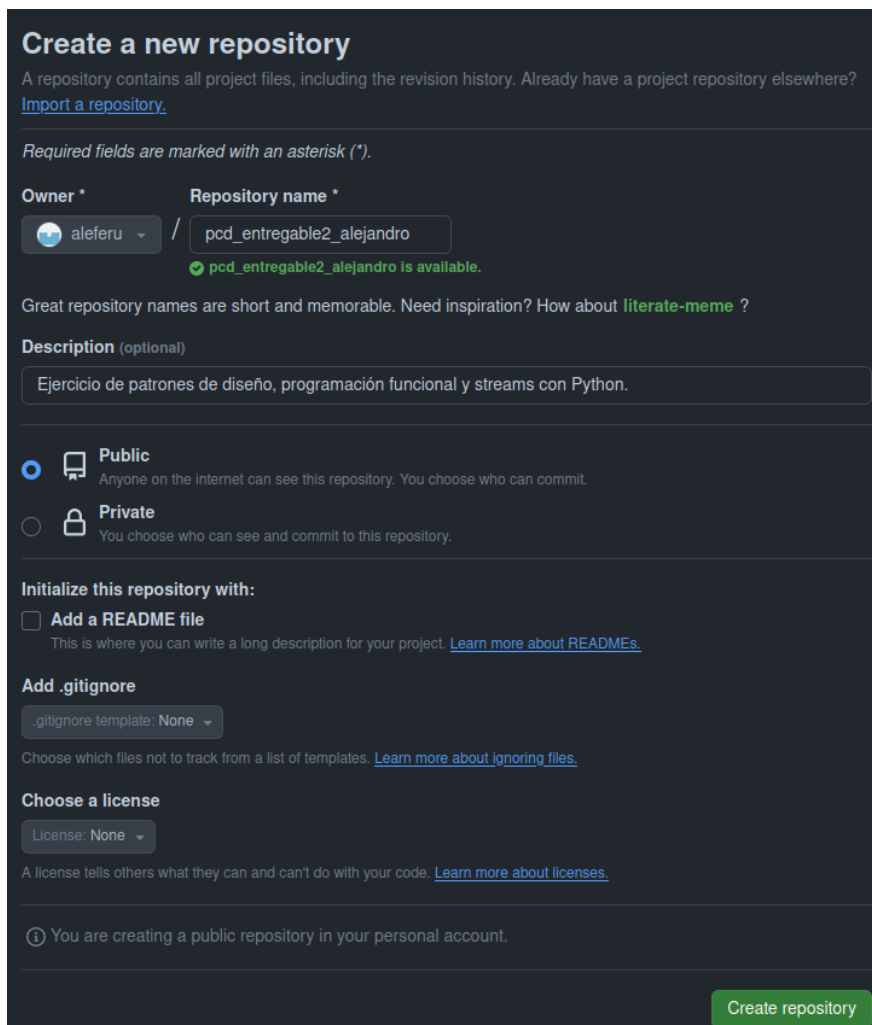
```
(pcd)
~/code/python/practicas-pcd/entrega-2/python
$ git tag 1.0
(pcd)
~/code/python/practicas-pcd/entrega-2/python
$ git tag
1.0
(pcd)
~/code/python/practicas-pcd/entrega-2/python
$ git push origin main
Everything up-to-date
(pcd)
~/code/python/practicas-pcd/entrega-2/python
$ git push origin 1.0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:aleferu/pcd_entregable2_alejandro.git
* [new tag]          1.0 -> 1.0
```

2. El segundo tag se ha creado cuando he terminado todo y estaba a punto de prepararlo todo para la entrega. Lo he llamado entrega.

```
(pcd)
~/code/python/practicas-pcd/entrega-2
$ git tag entrega
(pcd)
~/code/python/practicas-pcd/entrega-2
$ git tag
1.0
entrega
(pcd)
~/code/python/practicas-pcd/entrega-2
$ git push origin main
gitEverything up-to-date
(pcd)
~/code/python/practicas-pcd/entrega-2
$ git push origin entrega
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:aleferu/pcd_entregable2_alejandro.git
* [new tag]          entrega -> entrega
```

## 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).



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two main sections: 'Owner' and 'Repository name'. The 'Owner' is set to 'aleferu' and the 'Repository name' is 'pcd\_entregable2\_alejandro'. A green checkmark indicates that the name is available. Below this, there is a 'Description' field with the text 'Ejercicio de patrones de diseño, programación funcional y streams con Python.' and a 'Public' radio button selected. Under 'Initialize this repository with:', there is an unchecked checkbox for 'Add a README file'. There is also a section for 'Add .gitignore' with a dropdown menu set to 'None'. At the bottom, there is a 'Choose a license' section with a dropdown menu set to 'None'. A green 'Create repository' button is at the bottom right.

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

```
~/code/python/practicass-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/](https://github.com/aleferu/pcd_entregable2_alejandro/)

Su estado inicial era:

