UNIVERSIDADE DA CORUÑA

Diseño Software

Práctica de Diseño (2020-2021)

INSTRUCCIONES: Fecha límite de entrega: 22/12/2020 (hasta las 23:59).

- **Los problemas se han de resolver aplicando principios y patrones de diseño. No será válida una solución que no los use.**
- **Informe**: Para cada problema hay que hacer un informe en el que se incluya:
  - Explicación de los **principios de diseño** usados (en particular los **SOLID**) y dónde en concreto se han usado (nombrar clases específicas de vuestro código).
  - Explicación del **patrón o patrones de diseño** usados. Para cada patrón se incluirá lo siguiente:
    - **Breve explicación del patrón elegido** y justificación de su utilización.
    - **Diagrama de clases** en el que se muestren las clases involucradas en el patrón. Es importante señalar en el propio diagrama el rol que juega cada clase propia en el patrón con anotaciones UML.
    - **Diagramas dinámicos** (secuencia, comunicación o estados) que muestren el funcionamiento dinámico de aspectos fundamentales del código. Deberéis decidir qué tipo de diagrama es el más adecuado para cada problema.
- **Código y forma de entrega**:
  - El código desarrollado debe reflejar fielmente el diseño del informe.
  - Deberá incluir pruebas y se debe comprobar la cobertura de las mismas.
  - Se deberá subir al repositorio un único proyecto de IntelliJ IDEA cuyo nombre sea vuestro grupo con el sufijo `-PD`. Por ejemplo: `DS-11-01-PD`.
  - La documentación explicando el diseño y los principios y patrones utilizados en ambos problemas se entregará como un fichero `PDF` dentro de un directorio `doc` del proyecto IntelliJ IDEA.
- **Evaluación**:
  - Esta práctica corresponde a 1/3 de la nota final de prácticas que consistirá en una evaluación de la memoria y el código según los siguientes criterios.
  - **Calidad de la documentación**: selección del patrón y principios adecuados, explicaciones claras de su uso, calidad y claridad de los diagramas entregados, correspondencia con el código, etc.
  - **Calidad del código**: Aplicación correcta de los patrones y principios, seguimiento correcto de la filosofía orientada a objetos, correspondencia con el diseño, pruebas adecuadas, etc.

1. **Control de la temperatura**

   Para controlar la calefacción de una vivienda diseñamos un termostato, que contará con los siguientes cuatro modos de funcionamiento:

   - *Off*: La calefacción está apagada.
   - *Manual*: Se enciende la calefacción.
   - *Timer*: Se enciende la calefacción con un temporizador. La calefacción estará encendida durante el tiempo (en minutos) indicado, pasando al modo *Off* al acabar dicho tiempo.
   - *Program*: Se establece una temperatura de consigna. La calefacción se enciende si la temperatura actual es menor que la temperatura de consigna, en caso contrario se apaga.

   El termostato puede transitar entre los distintos modos de funcionamiento pero ha de tener en cuenta la siguiente restricción: no se puede cambiar del modo *Timer* al modo *Program* directamente (o viceversa), antes hay que pasar por el modo *Off* o por el modo *Manual*.

   La clase `Termostato` debe incluir un atributo que permita conocer si la calefacción está encendida o apagada. Además, contará con los siguientes métodos:

   - `newTemperature(float currentTemperature)` permite registrar una nueva temperatura. Este método se llamará periódicamente para actualizar la temperatura en base a la información almacenada en los sensores. Con el objeto de simplificar el problema, simularemos el paso del tiempo considerando que cada vez que se llama al método han pasado 5 minutos.
   - `screenInfo()` proporciona la información requerida para la pantalla del termostato. A continuación, se indica los datos a mostrar para cada modo de funcionamiento y el formato solicitado. Además, se facilitan algunos ejemplos concretos.

```
* Modos de funcionamiento Off y Manual:
  Formato: Temperatura  Calefacción (ON/OFF)  Modo funcionamiento (O/M)
  Ejemplo 1: 20.1 OFF O
  Ejemplo 2: 20.1 ON  M
* Modo de funcionamiento Timer:
  Formato: Temperatura  Calefacción  Modo funcionamiento (T) (tiempo restante)
  Ejemplo: 21.0 ON  T 14
* Modo de funcionamiento Program:
  Formato: Temperatura  Calefacción  Modo funcionamiento (P) (temp. consigna)
  Ejemplo: 20.1 OFF P 20.0
```

   A mayores, es preciso registrar cada evento (un cambio de temperatura, una modificación en el modo de funcionamiento, un cambio en el estado de la calefacción, etc.). A continuación se indica el formato para registrar la información en cada caso y algunos ejemplos concretos.

```
* Modos de funcionamiento Off y Manual:
  Formato: Temperatura Modo funcionamiento -  Estado calefacción
  Ejemplo 1: 20.1 Modo Off - Calefacción apagada.
  Ejemplo 2: 20.1 Modo Manual - Calefacción encendida.
* Modo de funcionamiento Timer:
  Formato: Temperatura Modo funcionamiento (tiempo restante) -  Estado calefacción
  Ejemplo: 21.0 Modo Timer (faltan 14 minutos) - Calefacción encendida.
* Modo de funcionamiento Program:
  Formato: Temperatura Modo funcionamiento (temperatura) -  Estado calefacción
  Ejemplo: 19.2 Modo Program (a 20.0 grados) - Calefacción encendida.
```

A mayores, en el momento en el que se registra un cambio en el modo de funcionamiento del termostato es necesario registrar el evento. El formato, al igual que antes, depende del modo de funcionamiento como se puede ver en los ejemplos que se muestran a continuación,

```
    Se activa el modo Manual.
    Se activa el modo Timer 19 minutos.
    Se activa el modo Program a 20.0 grados.
```

Únicamente se registrará el evento de desactivar el modo *Timer*.

```
     Se desactiva el modo Timer.
```

De esta forma se facilita la obtención de un *log* con la información sobre el funcionamiento del termostato. Teniendo en cuenta el formato indicado para el registro de eventos, un ejemplo de dicho log de salida podría ser el siguiente:

```
    20.1 Modo Off - Calefacción apagada.
    Se activa el modo manual.
    20.1 Modo Manual - Calefacción encendida.
    21.5 Modo Manual - Calefacción encendida.
    21.1 Modo Manual - Calefacción encendida.
    Se activa el modo Timer 19 minutos.
    21.0 Modo Timer (faltan 14 minutos) - Calefacción encendida.
    21.9 Modo Timer (faltan 9 minutos) - Calefacción encendida.
    22.8 Modo Timer (faltan 4 minutos) - Calefacción encendida.
    Se desactiva el modo Timer.
    22.5 Modo Off - Calefacción apagada.
    21.4 Modo Off - Calefacción apagada.
    Se activa el modo Program a 20.0 grados.
    21.2 Modo Program (a 20.0 grados) - Calefacción apagada.
    20.8 Modo Program (a 20.0 grados) - Calefacción apagada.
    20.1 Modo Program (a 20.0 grados) - Calefacción apagada.
    19.2 Modo Program (a 20.0 grados) - Calefacción encendida.
    19.9 Modo Program (a 20.0 grados) - Calefacción encendida.
    20.7 Modo Program (a 20.0 grados) - Calefacción apagada.
    22.8 Modo Program (a 20.0 grados) - Calefacción apagada.
    Se activa el modo Off.
    20.2 Modo Off - Calefacción apagada.
```

Desarrolla una solución, basada en principios y patrones de diseño, que permita representar adecuadamente el termostato y sus modos de funcionamiento, representando adecuadamente los cambios entre ellos, y que permita añadir fácilmente nuevos modos de funcionamiento en el futuro.

2. **Gestión de equipos de trabajo**

   Un proyecto se organiza en una jerarquía de equipos de trabajo, de forma que en un equipo puede haber varios trabajadores. A su vez, un equipo puede estar formado por varios subequipos. Cada **proyecto** tiene un nombre que lo identifica de manera única. Para este caso, asumiremos que cada trabajador o equipo puede trabajar en varios proyectos a la vez. Además, si un equipo se añade a un proyecto, todos sus miembros también se añaden.

   Los **trabajadores** tienen un nombre que será un identificador único y un coste por hora que no cambian. Al finalizar la jornada laboral, se contabilizan las horas que ha invertido cada trabajador en cada proyecto. Así, en cualquier momento podemos obtener las horas invertidas o el coste total que ha supuesto un trabajador para el proyecto. Además, se podrá obtener en un `String` la información del trabajador que vendrá en el siguiente formato: nombre, horas invertidas y coste total del trabajador para un proyecto dado. Por ejemplo, para el trabajador *Juan* del proyecto *Software Design* se mostrará:

```
Worker Juan: 60.0 hours, 900.0 €
```

   Para simplificar, supondremos que un trabajador no puede pertenecer a dos equipos diferentes del mismo proyecto.

   Los **equipos de trabajo** se identificarán a través de un nombre que será único. De cada equipo nos interesan las horas invertidas por todos los trabajadores (incluidos los de sus subequipos) y el coste total de todos ellos para un proyecto dado. Se podrá obtener la información de un equipo en un `String` con el siguiente formato: para cada equipo, las horas invertidas por sus trabajadores así como su coste, y además, la información de cada uno de ellos de manera individual así como de sus subequipos, todo ello para un proyecto dado, aplicando indentación para su correcta lectura. De esta forma, la información para un proyecto dado *Software Design* será la que se muestra a continuación:

```
Team Teaching: 180.0 hours, 3060.0 €
    Worker Edu: 20.0 hours, 500.0 €
    Team Theory: 10.0 hours, 180.0 €
        Worker David: 10.0 hours, 180.0 €
    Team Practicum: 150.0 hours, 2380.0 €
        Worker Juan: 60.0 hours, 900.0 €
        Worker Bea: 50.0 hours, 1000.0 €
        Worker Laura: 40.0 hours, 480.0 €
```

   A mayores, se podrán obtener en una lista todos los cotrabajadores de un equipo o de un trabajador. Los cotrabajadores de un trabajador o equipo son todos los trabajadores que participan en el mismo proyecto.

   Desarrolla una solución, basada en principios y patrones de diseño, que nos permita gestionar adecuadamente los distintos equipos de trabajo de un proyecto.