Examen Práctico
Convocatoria de Febrero de 2023

Previo. Gestión de fechas.

- Para representar las fechas utilizaremos la clase **`java.time.LocalDateTime`**, que implementa una fecha (día y hora) sin información de la zona horaria de acuerdo al sistema de calendario ISO-8601.
- El formato (por defecto) es `año-mes-día`**`T`**`hora:min:segundos`, `2023-01-16T10:00:00`.
- Las fechas (tanto `LocalDate` como `LocalDateTime`) disponen del método de clase `now()` que devuelve la fecha, y la fecha y hora actual, respectivamente.
- Las fechas (tanto `LocalDate` como `LocalDateTime`) son **comparables**, aun así, también están disponibles los métodos `isAfter`, `isBefore` e `isEqual`.
- Los objetos `LocalDate` y `LocalDateTime` son **inmutables**.

El objetivo de la aplicación es la implementación de un sistema de gestión de tareas de una organización.

1. Funcionalidad básica

Un ***operador*** representa a un trabajador encargado de realizar tareas en un ***área***. Las posibles áreas de la organización son: contabilidad, unidad técnica e informática (enumerado). Así, la clase que implementa el tipo Operador define objetos inmutables caracterizados por:

- Nombre: cadena de texto que lo identifica.
- Área en la que trabaja.

Implementa los métodos `equals/hashCode` en la clase Operador. Se entiende que dos operadores son iguales si tienen el mismo nombre y están asignados a la misma área.

Una ***tarea*** representa una petición de un usuario de la organización en un área de trabajo. Las propiedades que caracterizan a una tarea son: identificador (cadena de texto), peticionario (cadena de texto), descripción (cadena de texto), área, operadores asignados (colección) y fecha-hora de creación (`LocalDateTime`).

La clase ofrece dos constructores. El primer constructor recibe como argumentos el peticionario, descripción y área. El segundo constructor omite el último argumento (área), asignando como valor por defecto *unidad técnica*.

Sobre la inicialización de las propiedades:

- El identificador se inicializa con un valor aleatorio obtenido en el constructor con la llamada `java.util.UUID.randomUUID().toString()`.
- La colección de operadores se inicializa a vacío.
- La fecha-hora de creación se inicializan con el instante actual.

Las propiedades identificador, peticionario, descripción, área y fecha-hora de creación no pueden ser modificadas una vez que han sido inicializadas en la construcción.

La clase ofrece un método que solo debe ser accesible desde el paquete al que pertenece:

- Añadir operador: añade un operador a la colección de operadores si el operador no estaba previamente introducido en la colección. Esta operación retorna un valor booleano que informa si se ha introducido el operador.

2. Clase Gestor

Un ***gestor*** representa la herramienta que utilizan los usuarios de la organización para gestionar las tareas. Las propiedades que caracterizan a esta clase son:

- Nombre. Cadena de texto que identifica al gestor. No puede modificarse una vez inicializada en el constructor.
- Áreas. Colección de áreas que atiende el gestor.
- Tareas que han sido dadas de alta en el gestor. Esta información se almacenará en un **mapa** que utiliza como claves el identificador de las tareas. Por tanto, asocia cadenas de texto con tareas. El método de consulta de esta propiedad debe devolver solo la colección de tareas (no el mapa).
- Colección de operadores que atienden las tareas que lleguen al gestor.
- Tareas huérfanas. Representa la colección de tareas que no tienen asignado todavía ningún operador.

En la construcción se inicializa el nombre y, utilizando un argumento variable, las áreas que atiende el gestor, el resto de las colecciones se inicializan vacías.

La funcionalidad que ofrece la clase es la siguiente:

- Consultar la tarea asociada a un identificador. Retorna un valor nulo si la tarea no está en el mapa.
- Añadir operador. Recibe como parámetro el operador. Se añade a la colección de operadores solo en caso de que el operador sea de una de las áreas del gestor. Retorna un valor booleano informando si se ha realizado la inserción.
- Registrar tarea: este método se encarga de construir y almacenar una tarea en el gestor. Recibe como argumentos el peticionario, la descripción y el área, que son utilizados para construir la tarea. Si el área pasada como parámetro es una de las áreas atendidas por el gestor, el método finaliza retornando el identificador de la tarea almacenada. En caso contrario devolverá el valor nulo.
- Asignar operador a una tarea: el método recibe como argumentos el identificador de una tarea y un operador. Se impone como **precondiciones** que los argumentos no sean nulos, que el identificador corresponda a una tarea del gestor, que el operador esté registrado en el gestor y que su área de trabajo sea la misma que la de la tarea. El método se encarga de añadir el operador a la tarea, retornando un valor booleano que informa si el operador se ha insertado (no estaba previamente). Implementa el *control de precondiciones* en este método.

3. Clase Gestor Automático

Un ***gestor automático*** es un tipo de gestor que se caracteriza por intentar asignar automáticamente el operador más descargado a las nuevas tareas. Para ello necesita almacenar las tareas que han sido asignadas a cada operador. A esta propiedad se

denomina *carga*. La estructura de datos para almacenar esta información será un **mapa** que inicialmente estará vacío. Las tareas las registra en orden secuencial conforme se le van asignando.

Sobre la carga de trabajo, este tipo de gestor permite hacer las siguientes consultas:

- Obtener las tareas asignadas a un operador: recibe como parámetro un operador y devuelve una colección con las tareas que le han sido asignadas. Si el operador no está registrado devuelve el valor nulo.
- Obtener la carga de un operador: recibe como parámetro un operador y retorna el número de tareas que tiene asignadas. Si el operador no está registrado en el gestor devuelve el valor -1.
- Obtener operador candidato*:* dada un área recibida como parámetro, se calcula el operador de esa área que está más descargado (tiene menor número de tareas). Esta operación retorna un valor nulo si no hay ningún operador para esa área.

En relación con la funcionalidad heredada, hay que tener en cuenta que:

- Cuando se añade un nuevo operador, debe quedar registrado en el mapa.
- Cuando se asigna un operador a una tarea, debe añadirse a la colección de tareas del operador.
- Al registrar una tarea hay que intentar asignarle el operador candidato para el área de la tarea.

4. Métodos de `Object`

Implementa el método `toString` en todas las clases.

Implementa el método `clone` en la jerarquía de gestores siguiendo las recomendaciones de la asignatura. La copia de un gestor mantiene las áreas y operadores (en colecciones independientes) y no tiene tareas.

5. Programa

Crea una clase `Programa` que incluya la siguiente funcionalidad:

- Crea un objeto de tipo Gestor Automático de nombre “DUMBO” y que atiende todas las áreas. Asigna la referencia del objeto creado a una variable denominada *gestor*.
- Declara una variable de tipo Operador y asígnale un objeto operador de nombre “Jorge” y área “Informática”.
- Declara una variable de tipo Operador y asígnale un objeto operador de nombre “Rubén” y área “Informática”.
- Declara una variable de tipo Operador y asígnale un objeto operador de nombre “María” y área “Contabilidad”.
- Añade los tres operadores al gestor.
- Registra en el gestor las siguientes tareas:
  - Peticionario “Verónica”, descripción = “Error al acceder al AV”, área = “Informática”.

- Peticionario “Mariano”, descripción = “Instalar certificado”, área = “Informática”.
- Peticionario “Andrés”, descripción = “Error en factura F456”, área = “Contabilidad”.

- Muestra por la consola la información del gestor (`toString`). Nótese que se habrá asignado un operador automáticamente a cada tarea

6. Consultas

Incluye en la clase `Gestor` las siguientes consultas:

- Consultar las tareas ordenadas atendiendo, en primer lugar, al área, y en caso de empate, a la fecha-hora de creación, de más recientes a más antiguas (orden inverso al orden natural de las fechas). Nota: por defecto, los enumerados tienen implementado un orden natural correspondiente con el orden de declaración.
- Consultar las tareas filtradas. Este método de consulta retorna el subconjunto de la colección de tareas que cumpla el criterio (`Predicate`) que se establece como argumento.
- Consultar si hay tareas de un área. Para implementar esta última operación se impone como **requisito** utilizar un *procesamiento basado en stream*.

Añade una prueba de cada método en el programa.

7. Métodos de utilidad

En la clase `Utilidades` incluye los siguientes métodos:

a) Implementa un ***método de clase genérico*** que reciba como parámetro un mapa cuyas claves sean cadenas de texto y que asocie como valores colecciones de objetos del mismo tipo. La tarea del método es retornar una única colección con todos los objetos almacenados en las colecciones del mapa. Añade una prueba del método al final del programa principal del ejercicio 5.

b) Implementa un método de clase `getFecha` que pida por teclado una cadena de texto y devuelva un objeto tipo `java.time.LocalDate`. La cadena de texto tiene que seguir el formato año-mes-día (por ejemplo, “2023-01-17”). Para hacer la conversión se hará uso del método de clase `parse` que ofrece la clase `LocalDate`. El método `parse` recibe como parámetro una cadena de texto y devuelve una instancia de `LocalDate`, si el formato de la cadena es correcto. Si no puede hacer la conversión, el método `parse` lo notifica lanzando la excepción no comprobada o *runtime* `java.time.format.DateTimeParseException`. El usuario tendrá **tres intentos** y si no es capaz de introducir una cadena de texto que represente una fecha correcta se considera un fallo en la postcondición del método `getFecha`. Implementa una prueba en el programa principal de manera que se imprima el objeto `LocalDate` que ha introducido el usuario o recuerde el formato de las fechas si el usuario agota los tres intentos.

Nota: Para leer texto por teclado hay que utilizar la clase `java.util.Scanner` mediante:

```
Scanner sc = new Scanner(System.in);
String texto = sc.nextLine();
```