Examen Práctico
Convocatoria de Julio de 2023

Previo. Gestión de fechas.

- Para representar las fechas y horas utilizaremos las clases: **`java.time.LocalDate`** (fecha) **y `java.time.LocalDateTime`** (fecha y hora) sin información de la zona horaria de acuerdo con el sistema de calendario ISO-8601.
- El formato (por defecto) es `año-mes-día` para **`LocalDate`** y `año-mes-día`**`T`**`hora:min:segundos`, `2023-01-16T10:00:00` para **`LocalDateTime`**.
- Todas las clases citadas disponen del método de clase `now()` que devuelve la fecha, y la fecha y hora actual, respectivamente.
- Para crear objetos de tipo `LocalDateTime` se utiliza el método de clase **`of`** que recibe como parámetro (fecha, mes, día, hora, minutos). Por ejemplo, para la fecha de 26 de junio de 2023 a las 10:00 sería: `LocalDateTime examen = LocalDateTime.of(2023, 6, 26, 10,0);`
- Tanto los objetos que representan las fechas como las horas son **comparables**, aun así, también están disponibles los métodos `isAfter`, `isBefore` e `isEqual`.
- Los objetos `LocalDate` y `LocalDateTime` son **inmutables**.
- La clase `LocalDateTime` ofrece el método `toLocalDate` que devuelve la fecha (`LocalDate`) representada en el objeto fecha-hora.
- Las clases `LocalDate/LocalDateTime` disponen de métodos de consulta de todas sus propiedades: `getYear`, `getMonth`, `getDayOfMonth`, `getDayOfWeek`.
- Las clases `LocalDate/LocalDateTime` disponen de métodos para incrementar y decrementar sus propiedades, por ejemplo: `plusYears, plusDays, plusMinutes.`

Los ejercicios propuestos tienen como objetivo el desarrollo de una herramienta de gestión de calendarios.

1. Funcionalidad básica.

La clase **Evento** representa un suceso programado en un calendario. Una característica de esta clase es que define *objetos inmutables*. Las propiedades de esta clase son:

- Nombre: cadena de texto que describe el nombre del evento.
- Inicio: instante en el que comienza el evento (`LocalDateTime`).
- Fin: instante en el que concluye el evento (`LocalDateTime`).

La funcionalidad de la clase es:

- Comprobar si un instante de tiempo (`LocalDateTime`) está dentro del evento, esto es, es igual o posterior al inicio e igual o anterior al fin. El resultado de la comprobación se retorna en un valor booleano.
- Comprobar si *solapa en el tiempo* con otro evento. Se cumple la condición si el inicio o el fin del otro evento están dentro del evento actual. El resultado de la comprobación se retorna en un valor booleano.

Por último, esta clase redefine los métodos `equals/hashCode` utilizando todos los atributos en el cálculo.

La clase **Recordatorio** representa avisos previos a un evento. Esta clase también define *objetos inmutables*. Las propiedades son:

- Evento: corresponde con el evento al que está asociado.
- Fecha y hora: es el momento en el que debe notificarse el recordatorio.

Esta clase ofrece tres constructores. El primero recibe las dos propiedades como parámetro. El segundo recibe como parámetro el evento y la antelación a la fecha del evento en minutos (por ejemplo, 5 minutos, 10 minutos, etc.). El tercero recibe sólo el evento y se entiende que se va a crear con una antelación de 30 minutos antes del comienzo del evento.

La funcionalidad de la clase es:

- Obtener un recordatorio anterior. Esta operación construye y retorna un objeto recordatorio cuya *fecha y hora* sea unos minutos antes que la fecha y hora del **recordatorio actual** y esté asociado al mismo evento**.** El número de minutos se establece como parámetro de la operación.

Un **Calendario** es un tipo de datos que registra eventos y los recordatorios asociados a estos. Las propiedades son:

- Nombre (cadena de texto). Esta propiedad no cambia una vez establecida en el constructor.
- Colección de eventos registrados. **Nota**: Esta propiedad se organiza en un **mapa** que asocia eventos con una colección de sus recordatorios. Este mapa es un atributo de implementación que no debe consultarse. En cambio, sí se consulta la propiedad que representa los eventos.
- Número de eventos.
- Colección de todos los recordatorios del calendario.

La funcionalidad de la clase es:

- Consultar los recordatorios de un evento. **Nota**: declara esta operación como un método *sobrecargado* del método de consulta de todos los recordatorios del calendario.
- Añadir evento. La tarea de esta operación es añadir un nuevo evento al calendario. Recibe como parámetros el nombre, la fecha-hora de inicio y la duración del evento (en minutos). El método retorna el evento creado. Implementa el ***control de precondiciones*** en el método siguiendo las siguientes indicaciones: 1) El nuevo evento no debe *solapar en el tiempo* con ningún otro evento del calendario; y 2) Los parámetros tienen que ser correctos.

- Versión sobrecargada del método anterior que crea eventos de un día. Recibe como parámetros el nombre del evento y la fecha (`LocalDate`). Crea el evento a las 00:00 de esa fecha y como duración un día.
- Borrar un evento. Esta operación elimina el evento del calendario junto con todos sus recordatorios. Devuelve un valor booleano para indicar si se ha borrado. • Borrar recordatorios de un evento. Con esta operación se borran todos los recordatorios de un evento recibido como parámetro. Si el evento no existe el método devuelve el valor falso para indicar que no ha habido éxito. Devolverá verdadero en caso contrario.
- Crear un recordatorio. La tarea de este método es añadir un recordatorio para un evento recibido como parámetro. También se recibe como parámetro el número de minutos de antelación del recordatorio. El evento debe existir en el calendario y el nuevo recordatorio no debe coincidir en el tiempo con otro recordatorio para ese evento. El método devuelve el objeto creado o el valor nulo si no se ha podido crear. • Versión sobrecargada del método anterior en el que no se recibe como parámetro el número de minutos de antelación y se toma el valor por defecto de la clase Recordatorio.
- Obtener los eventos futuros. Devuelve una ***lista*** con los eventos posteriores a la fecha actual **ordenada** por la propiedad inicio de forma ascendente.

2. Calendario automático

Un **calendario automático** es un tipo de calendario que se caracteriza por crear automáticamente un recordatorio para cada evento creado. Los minutos de antelación para crear los recordatorios es una propiedad de la clase que se establece en la construcción y no varía.

La funcionalidad que añade este calendario es la siguiente:

- Registrar una **serie de eventos anuales**. Recibe como parámetros el nombre, que va a ser el mismo para todos los eventos de la serie, la fecha de inicio del primer evento (`LocalDate`) y el número de repeticiones en la serie. Este método crea eventos con duración de un día y tantas repeticiones como indique el parámetro con un espacio temporal de un año entre los inicios de los eventos de la serie. **NOTA:** Para gestionar las series de eventos se debe utilizar un atributo de implementación, por ejemplo, una lista de colecciones, donde cada colección es una serie de eventos, o un mapa donde la clave es el evento inicial y el valor asociado es la serie de eventos.

Por último, cuando se borra un evento, si este pertenece a una serie (puede ser el evento inicial de la serie o alguno de los eventos anuales), se tienen que borrar todos los eventos de esa serie.

3. Métodos de la clase `Object`.

Implementa el método `toString` en todas las clases implementadas. **NOTA**: Añade `\n` al final de la cadena de texto que muestra la información de los eventos y recordatorios para visualizar mejor el resultado del programa.

Implementa el método `clone` en la jerarquía de calendarios siguiendo las recomendaciones de la asignatura. La copia de un calendario tiene inicialmente los mismos eventos y no tendrá recordatorios asociados a los eventos.

4. Programa

Implementa un programa con la siguiente funcionalidad:

- Crea un calendario con nombre “personal”.
- Añade dos eventos a ese calendario: “dentista” el 14/9/23 a las 17:30 con 60 minutos de duración y “cita médico” el 13/9/23 a las 9:30 con 15 minutos.
- Crea un calendario automático llamado “celebraciones” con antelación de recordatorios de 60 minutos.
- Registra una serie de eventos en ese calendario con nombre “Cumple María” para el 16/7/23, con 10 repeticiones.
- Añade los dos calendarios creados a una lista.
- Recorre la lista y para cada uno:
    - Muestra su nombre.
    - Si es un calendario automático, registra una serie de eventos con nombre “¡Acabo el grado!”, con fecha mañana y 50 repeticiones.
    - Muestra los eventos futuros y todos sus recordatorios.

5. Programación funcional.

Resuelve los siguientes ejercicios haciendo uso de las *expresiones lambda* y, cuando sea necesario, del *procesamiento basado en streams*:

5.a) Añade a la clase que representa los ***calendarios automáticos*** la siguiente funcionalidad

- Consultar los eventos de hoy. Devuelve una colección con los eventos del calendario registrados para el día (`LocalDate`) actual.
- Añadir recordatorios. Crea un nuevo recordatorio para todos los eventos futuros con una antelación que se establece como parámetro.

5.b) Añade al final del bucle del programa la consulta para comprobar si el calendario normal tiene algún evento en el mes de septiembre de 2023.

6. Métodos de utilidad

Implementa una clase Utilidad que incluya los siguientes métodos de clase:

6.a) Implementa un *método genérico* que reciba como parámetros dos mapas y que retorne un nuevo mapa con aquellas entradas que solo están presentes en uno de los dos mapas.

6.b) Implementa un método de clase llamado `touch` que genere ficheros vacíos. El parámetro del método es un *argumento variable* con los nombres de los ficheros que se quiere generar. Para la implementación se hará uso de la clase `java.io.File` que representa un fichero. Ofrece un constructor que recibe como parámetro el nombre del fichero. Una vez construido, la aplicación del método `createNewFile` crea el fichero en disco si no existe. Este método notifica la

excepción `java.io.IOException` si se produce un error. Al finalizar la ejecución del método `touch` todos los ficheros tienen que existir. Si no cumple esta tarea, debe notificarlo lanzando una excepción cuyo mensaje de error sea la descripción del error que ha ocurrido y el nombre de todos los ficheros que no se han podido crear. Por último, realiza una prueba en el programa principal.