Taller # 5 DPOO

Alejandro Cataño Cardona

202011795

Librería Joda-Time en java

Joda-Time es una biblioteca de Java que proporciona clases y métodos para trabajar con fechas, horas, períodos y duraciones de tiempo. Fue desarrollada como una alternativa más poderosa y fácil de usar a las clases de fecha y hora proporcionadas por el paquete java.util de Java.

Joda-Time se centra en resolver los desafíos comunes asociados con el manejo de fechas y horas, como cálculos de diferencia entre fechas, manipulaciones de fechas y horas, y representación formateada de fechas y horas. Ofrece una amplia gama de funcionalidades, incluyendo:

-Manipulación de fechas y horas: Joda-Time proporciona clases como DateTime, LocalDate, LocalTime y DateTimeZone que permiten crear y manipular fechas y horas de forma precisa. Estas clases ofrecen métodos para obtener y modificar componentes específicos de fechas y horas, como año, mes, día, hora, minuto y segundo.

-Cálculos de diferencia y duraciones: Joda-Time permite calcular la diferencia entre dos fechas o horas, así como realizar operaciones aritméticas con duraciones de tiempo. La clase Duration se utiliza para representar una cantidad específica de tiempo en milisegundos y proporciona métodos para realizar operaciones de suma, resta y comparación.

-Formateo y análisis de fechas y horas: Joda-Time ofrece una clase DateTimeFormatter que permite formatear fechas y horas en una representación legible para los humanos y también analizar cadenas de texto en objetos de fecha y hora. Esto proporciona una flexibilidad significativa para trabajar con diferentes formatos de fecha y hora.

-Zonas horarias: Joda-Time maneja las zonas horarias de manera eficiente y precisa. La clase DateTimeZone permite obtener información sobre las zonas horarias, como el desplazamiento de tiempo y si siguen la hora estándar o tienen un desplazamiento fijo.

Así pues, Joda-Time es una biblioteca de Java que simplifica y mejora el manejo de fechas, horas, períodos y duraciones. Proporciona clases y métodos que facilitan la manipulación, cálculo y formateo de fechas y horas de manera precisa y eficiente. Es una opción popular para trabajar con el tiempo en aplicaciones Java y ha sido ampliamente adoptada en diversos proyectos.

Joda-Time implementa una variedad de diseños y patrones dentro su implementación, aquellos más importantes son

Immutabel Object: Joda-Time sigue el patrón de Immutabel oBject, donde las clases que representan fechas, horas y duraciones son inmutables. Los objetos inmutables garantizan la seguridad de hilos y evitan modificaciones no deseadas.

Builder: Joda-Time utiliza el patrón Builder para crear instancias de objetos complejos con una API fluida. Las clases DateTimeBuilder y DateTimeFormatterBuilder son ejemplos de constructores utilizados en Joda-Time.

Facotry method: Se emplea el Facotry method para crear instancias de sus diversas clases. Las fábricas como DateTimeZone, DateTimeFormatter y PeriodType proporcionan métodos para crear instancias con configuraciones específicas.

Singleton: La clase DateTimeZone en Joda-Time sigue el patrón Singleton, asegurando que solo haya una instancia de una zona horaria disponible en toda la aplicación.

Composite: El patrón Composite se utiliza para representar estructuras complejas de fecha y hora. Por ejemplo, la clase Interval combina dos instancias de DateTime para representar un intervalo de tiempo.

Strategy: Joda-Time utiliza el patrón Strategy con su interfaz Chronology. Diferentes implementaciones de Chronology permiten diferentes sistemas de calendario, como el gregoriano, juliano o budista.

Visitor: el patrón Visitor se utiliza para la funcionalidad de formateo y análisis. La clase DateTimeFormatter acepta un visitor que realiza operaciones en elementos específicos durante el proceso de formateo o análisis.

```
 * @since 1.0
 */
public abstract class DateTimeZone implements Serializable {

    /** Serialization version. */
    private static final long serialVersionUID = 5546345482340108586L;

    /** The time zone for Universal Coordinated Time */
    public static final DateTimeZone UTC = UTCDateTimeZone.INSTANCE;
    /** Maximum offset. */
    private static final int MAX_MILLIS = (86400 * 1000) - 1;

    /**
     * The instance that is providing time zones.
     * This is lazily initialized to reduce risks of race conditions at startup.
     */
```

Implementación Singleton

```
Code  Blame  861 lines (811 loc) · 25.6 KB                                  Raw

 * @author Stephen Colebourne
 * @since 1.0
 */
public class PeriodType implements Serializable {
    /** Serialization version */
    private static final long serialVersionUID = 2274324892792009998L;

    /** Cache of all the known types. */
    private static final Map<PeriodType, Object> cTypes = new HashMap<PeriodType, Object>(32);

    static int YEAR_INDEX = 0;
    static int MONTH_INDEX = 1;
    static int WEEK_INDEX = 2;
    static int DAY_INDEX = 3;
    static int HOUR_INDEX = 4;
    static int MINUTE_INDEX = 5;
    static int SECOND_INDEX = 6;
    static int MILLI_INDEX = 7;

    private static PeriodType cStandard;
    private static PeriodType cYMDTime;
    private static PeriodType cYMD;
    private static PeriodType cYWDTime;
    private static PeriodType cYWD;
```

Implementación Factory method

La implementación de estos patrones de diseños Atraen varias ventajas y desventajas, como las siguientes.

Ventajas

Immutable Object: El uso de objetos inmutables en Joda-Time garantiza la seguridad de hilos y evita problemas de concurrencia al trabajar con fechas y horas en entornos multihebra. Además, la inmutabilidad simplifica el razonamiento y la comprensión del código.

Builder: El patrón Builder utilizado en Joda-Time permite crear objetos complejos de forma clara y legible utilizando una sintaxis fluida. Proporciona un enfoque flexible para configurar objetos paso a paso y facilita la creación de instancias con diferentes configuraciones.

Factory Method: El uso Facotry method en Joda-Time proporciona una forma conveniente de crear instancias de objetos con configuraciones específicas. Esto simplifica la creación de objetos complejos y permite una mayor flexibilidad en la configuración de parámetros.

Singleton: La implementación del patrón Singleton en la clase DateTimeZone garantiza que solo exista una instancia de una zona horaria en todo el sistema. Esto ayuda a conservar memoria y asegura la coherencia al trabajar con zonas horarias.

Composite: El patrón Composite utilizado en Joda-Time para representar estructuras complejas de fecha y hora facilita la manipulación y el cálculo de intervalos de tiempo y componentes relacionados. Permite trabajar con abstracciones de alto nivel y simplifica las operaciones en conjuntos de objetos relacionados.

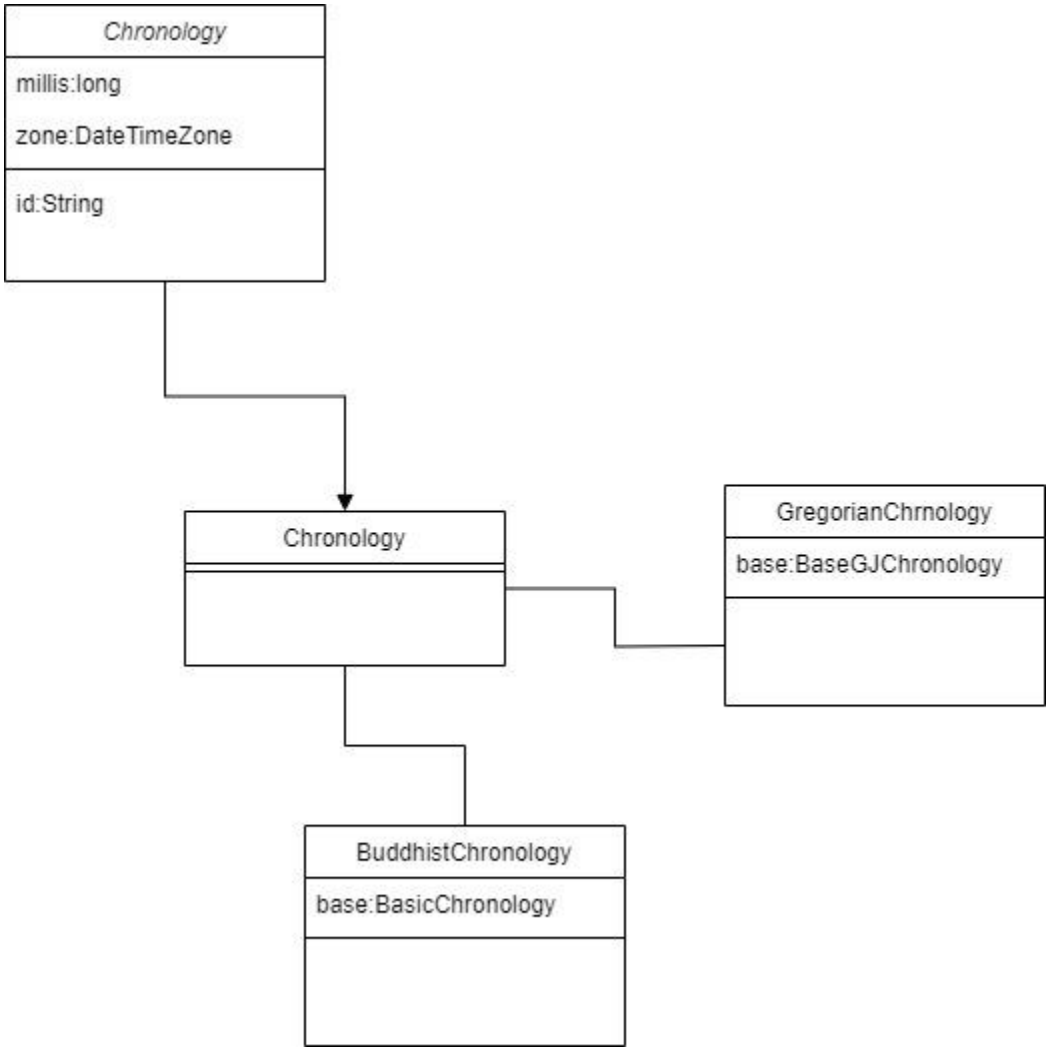
Strategy: El uso del patrón Strategy con la interfaz Chronology en Joda-Time permite adaptar el comportamiento de cálculo de fechas y horas a diferentes sistemas de calendario. Esto proporciona flexibilidad para trabajar con calendarios personalizados y facilita la interoperabilidad entre diferentes sistemas de calendario.

Desventajas

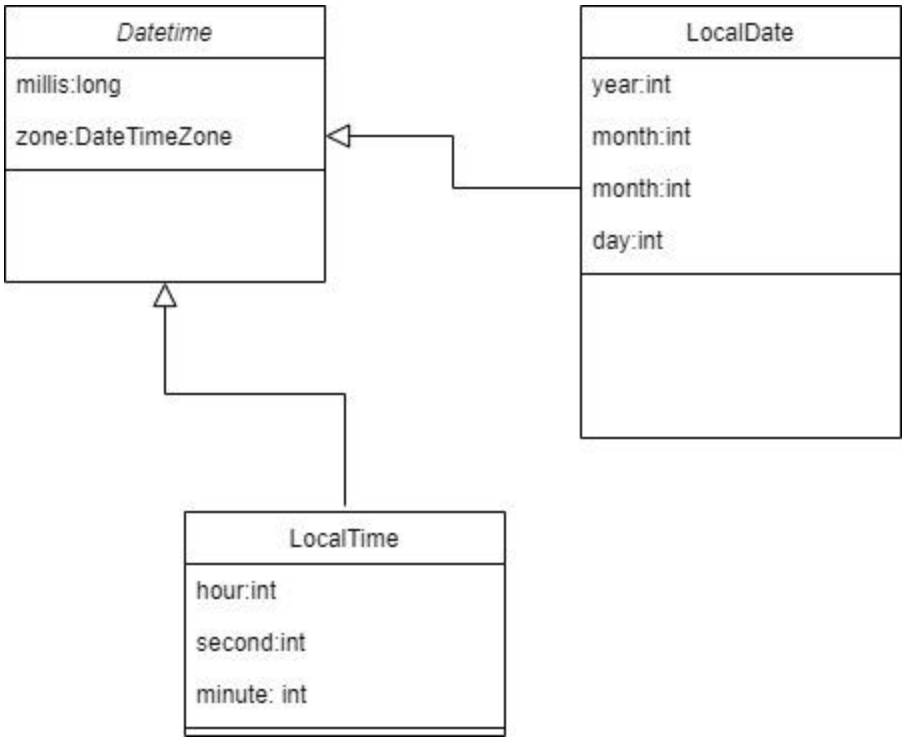
Curva de aprendizaje: Algunos patrones de diseño utilizados por Joda-Time pueden tener una curva de aprendizaje para aquellos que no están familiarizados con ellos. Puede llevar tiempo comprender completamente cómo utilizar y aprovechar al máximo los patrones implementados.

Complejidad adicional: Al implementar varios patrones de diseño, el código puede volverse más complejo y difícil de mantener en comparación con una solución más simple. El exceso de patrones puede complicar innecesariamente el diseño y la comprensión del código.

UML



Este diagrama representa las clases relacionadas con zonas horarias y cronologías en Joda-Time. DateTimeZone representa una zona horaria con un identificador único. Chronology define la interfaz para manipular y calcular fechas y horas según un calendario específico. En este ejemplo, se muestran dos implementaciones de Chronology: GregorianChronology, que utiliza el calendario gregoriano, y BuddhistChronology, que utiliza el calendario budista. Ambas implementaciones tienen una relación de composición con clases específicas de calendario (BaseGJChronology y BasicChronology, respectivamente).



Este diagrama representa la jerarquía de clases principales utilizadas en Joda-Time para el manejo de fechas y horas. La clase base es DateTime, que contiene una representación interna del tiempo en forma de milisegundos y una referencia a la zona horaria. DateTime se compone de una instancia de LocalDate y LocalTime, que representan la fecha y la hora respectivamente.