Subject Store

Guía de usuario

Marzo 2025

1. Introducción

`SubjectStore` es una biblioteca para la gestión de datos en torno a un *Subject* (sujeto o tema), permitiendo asociar atributos a marcas temporales. Su diseño se centra en *cómo* se alimentan los datos, y no en *cómo* se presentan, lo que posibilita fusionar distintas visiones parciales de un mismo *subject* sin imponer restricciones sobre la forma de consulta o visualización final.

Para entender mejor el funcionamiento y utilidad de `SubjectStore`, consideremos un caso de uso típico: la gestión de información médica de un paciente, identificado como `Paciente:12345`. Este paciente posee atributos que cambian con poca frecuencia, como sus datos personales (dirección, teléfono, correo electrónico), así como otros que varían constantemente, como los resultados de pruebas clínicas.

Por ejemplo, cada vez que el paciente realiza un hemograma, se tienen que registrar nuevos datos sobre diferentes parámetros sanguíneos (glóbulos rojos, glóbulos blancos, hemoglobina, plaquetas, etc.). El valor de estas mediciones es variable y no siempre se registran todos los parámetros. Por otra parte, se podrían registrar diagnósticos o tratamientos específicos desde diversas especialidades médicas (como cardiología, neurología, etc.), cada una aportando información parcial sobre el estado del paciente.

Con `SubjectStore`, cada fuente aporta datos de forma independiente, registrando únicamente aquellos atributos de los que dispone información en cada instante. Esta flexibilidad permite almacenar conjuntamente atributos altamente dinámicos (como las mediciones del hemograma) y atributos menos variables (como diagnósticos) sin necesidad de definir previamente una estructura rígida de datos. La librería se encarga posteriormente de conciliar

todas estas fuentes, ofreciendo una visión integral y fácilmente consultable del historial médico completo del paciente.

En resumen, las funcionalidades clave de `SubjectStore` incluyen:

- Almacenar información en un archivo vinculado a un `Subject`.
- Registrar sucesivas entradas de datos con fechas o marcadores de tiempo.
- Consultar tanto los valores más recientes como los historiales completos de cada atributo, abarcando distintos intervalos de tiempo.
- Generar informes tabulados con diferentes granularidades temporales.

El objetivo de esta guía es describir, paso a paso, cómo incorporar esta herramienta a tu proyecto y cómo utilizar sus métodos para almacenar, recuperar y presentar datos de manera flexible y eficiente.

2. Prerrequisitos

Antes de comenzar a utilizar `SubjectStore` en tu proyecto, es necesario cumplir con los siguientes requisitos básicos:

- **Java 11 o superior**: Asegúrate de contar con una versión compatible del JDK instalada en tu entorno de desarrollo. Puedes verificar tu versión ejecutando:

  ```
  java -version
  ```

- **Apache Maven**: Es necesario disponer de Maven como gestor de dependencias y herramienta de compilación para integrar fácilmente la biblioteca en tu proyecto. Puedes comprobar si está instalado con el siguiente comando:

  ```
  mvn -version
  ```

- **Conocimientos básicos sobre series temporales y atributos categóricos**: Dado que `SubjectStore` maneja datos temporales, es recomendable entender conceptos elementales sobre series de tiempo y datos categorizados.

3. Instalación

Puedes instalar `SubjectStore` en tu proyecto Java utilizando un gestor de dependencias como Maven o Gradle.

Para incorporar `SubjectStore` mediante Maven, añade la siguiente dependencia al archivo `pom.xml` de tu proyecto:

```
<dependency>
    <groupId>io.intino.alexandria</groupId>
    <artifactId>subject-store</artifactId>
    <version>1.0</version>
</dependency>
```

Si estás utilizando Gradle, añade esta línea en tu archivo `build.gradle` dentro del bloque de dependencias:

```
implementation 'io.intino.alexandria:subject-store:1.0'
```

4. Carga de datos

Para iniciar la carga de datos en un `SubjectStore`, es necesario instanciar la clase indicando el archivo donde se almacenará la información. Si el archivo no existe, será creado automáticamente. El nombre del `Subject` se deriva del nombre del archivo, sin la extensión. Por ejemplo, un archivo llamado `Paciente-12345.oss` definirá un `Subject` con nombre `Paciente-12345`. La carga de datos se realiza mediante el método `feed`, al que se le indica un instante temporal y una etiqueta que describe la fuente o contexto del dato. Luego, se añaden los atributos con `add` y se finaliza la operación con `execute`.

```
File file = new File("Paciente-12345.oss");
try (SubjectStore store = new SubjectStore(file)) {
    Instant instant = Instant.parse("2025-03-23T09:30:00Z");
    store.feed(instant, "Laboratorio.Hemograma.0512301")
         .add("Hemoglobina", 142)
         .add("Plaquetas", 250_000)
         .add("GlobulosBlancos", 5_600)
         .add("GrupoSanguineo", "A+")
         .execute();
}
```

Los valores numéricos se almacenan como enteros. Si se requiere precisión decimal (por ejemplo, `14.2` para hemoglobina), debe aplicarse una escala fija previamente (en este caso, multiplicando por 10). Los atributos categóricos, como el grupo sanguíneo, se almacenan directamente como cadenas de texto. Si una segunda fuente quiere aportar atributos adicionales en otro instante, simplemente se repite el procedimiento:

```
try (SubjectStore store = new SubjectStore(file)) {
    Instant instant = Instant.parse("2025-03-25T11:00:00Z");
    store.feed(instant, "Cardiologia.DrCabrera")
         .add("PresionSistolica", 125)
         .add("PresionDiastolica", 80)
         .add("RiesgoCardiovascular", "Moderado")
         .execute();
}
```

5. Referencias temporales

`SubjectStore` requiere que se indiquen instantes temporales para ubicar cronológicamente cada aportación de datos (`feed`). Para facilitar el manejo de estas referencias temporales, la librería proporciona en la clase `Instants` algunas referencias especiales predefinidas:

- `thisYear()`: Inicio del año actual (1 de enero).
- `thisMonth()`: Inicio del mes actual (día 1).
- `thisWeek()`: Inicio de la semana actual (lunes).
- `today()`: Día actual, truncado a medianoche UTC.
- `thisHour()`: Hora actual, truncada a la hora exacta.
- `thisMinute()`: Minuto actual, truncado al minuto exacto.
- `thisSecond()`: Segundo actual, truncado al minuto (nota: parece un error, devuelve el minuto actual).

Además, la clase `Instants` define dos marcadores temporales especiales:

- `LEGACY`: Representa datos históricos o heredados cuya fecha exacta no se conoce o no es relevante. Estos datos quedan registrados, pero permanecen pendientes de conciliación.
- `BIG_BANG`: Es el instante en el que se produce la conciliación inicial o primera sincronización de los datos históricos (`LEGACY`), permitiendo integrarlos de forma definitiva al historial del sujeto.

6. Consultas

Una vez que un `SubjectStore` ha sido alimentado con información, es posible consultar su contenido. Para abrir un archivo basta con instanciar la clase `SubjectStore` indicando el archivo (`.oss`) donde está almacenada la información.

```
File file = new File("Paciente:12345.oss");
try (SubjectStore store = new SubjectStore(file)) {
  ...
}
```

La instancia de `SubjectStore` proporciona una serie de métodos que permiten acceder a información estructural y de estado sobre el sujeto y los datos almacenados. A continuación se detallan los principales métodos disponibles:

- `name()`: devuelve el nombre del `Subject`, extraído del nombre del archivo.
- `feeds()`: retorna el número total de (`feeds`) registrados.
- `ss(feed)`: devuelve la etiqueta asociada a un `feed` concreto, útil para identificar su origen o propósito.
- `hasLegacy()`: verifica si existen datos legacy.
- `hasBigBang()`: verifica si se ha realizado la conciliacion.
- `from()`: devuelve el instante de tiempo correspondiente al primer `feed` cronológico real.
- `to()`: devuelve el instante de tiempo correspondiente al último `feed` cronológico real.

- `legacyExists()`: verifica si existen datos `LEGACY` almacenados.
- `bigBangExists()`: verifica si se ha registrado una conciliación.
- `legacyPending()`: verifica si hay datos históricos pendientes de ser conciliados.
- `tags()`: devuelve todos los atributos registrados.
- `exists(tag)`: verifica si un atributo específico ha sido registrado.

Además, se pueden realizar consultas para un atributo dado. Hay dos tipos de consultas:

- `numericalQuery(attribute)`: devuelve una consulta de tipo `NumericalQuery` si el atributo existe y es numérico.
- `categoricalQuery(attribute)`: devuelve una consulta de tipo `CategoricalQuery` si el atributo existe y es categórico.

6.1. Consulta de datos numéricos

Los atributos numéricos se consultan utilizando el método `numericalQuery`, que recibe por parámetros el nombre del atributo que se desea consultar. Para obtener el valor más reciente de un atributo, se utiliza el método `current()`, que devuelve un objeto `Point<Long>` con la información del valor, la fecha y el identificador del `feed` donde fue registrado. Si los datos fueron almacenados con escala (por ejemplo, multiplicando por 10 para representar decimales), será necesario dividir el valor al momento de interpretarlo. Cabe destacar que los datos registrados como `LEGACY` no se consideran en esta operación; el valor devuelto por `current()` corresponde siempre al dato más reciente dentro del historial cronológico real, excluyendo cualquier dato pendiente de conciliación.

```
Point<Long> hemoglobina = store
    .numericalQuery("Hemoglobina")
    .current();

double value = hemoglobina.value() / 10.0;
Instant instant = hemoglobina.instant();
int feed = hemoglobina.feed();
```

Para consultar todos los valores registrados en un período determinado, se usa el método `signal(from, to)`. Esto devuelve una serie temporal correspondiente al atributo dentro del intervalo indicado. La forma de operar con estas series se explica en la siguiente sección.

```
Signal signal = store
    .numericalQuery("Hemoglobina")
    .signal(today(-30), today());
```

6.2. Consulta de datos categóricos

Los atributos categóricos se consultan utilizando el método `categoricalQuery`, que recibe por parámetro el nombre del atributo que se desea consultar. A diferencia de los datos numéricos, los valores categóricos son cadenas de texto y no requieren normalización ni escalado.

Para obtener el valor más reciente de un atributo categórico, se utiliza el método `current()`, que devuelve un objeto `Point<String>` con el valor, el instante y el identificador del `feed` correspondiente. Al igual que en el caso de los atributos numéricos, los datos registrados como `LEGACY` no se tienen en cuenta en esta operación; el resultado de `current()` refleja exclusivamente el valor más reciente dentro del historial cronológico consolidado, una vez excluidos los datos históricos pendientes de conciliación.

```
Point<String> riesgo = store
    .categoricalQuery("RiesgoCardiovascular")
    .current();
String value = riesgo.value();
Instant instant = riesgo.instant();
int feed = riesgo.feed();
```

Para consultar todos los valores registrados en un rango de fechas, se utiliza el método `sequence(from, to)`. Este método devuelve una serie temporal que contiene los valores del atributo dentro del intervalo especificado. La forma de operar con estas series se explica en la siguiente sección.

```
Sequence sequence = store
    .categoricalQuery("RiesgoCardiovascular")
    .sequence(thisMonth(-12), today());
```

7. Series temporales

7.1. Numéricas: `Signal`

Una serie temporal numérica se obtiene a partir de una consulta a un atributo mediante el método `signal(from, to)`, que devuelve una instancia de `Signal`. Esta serie contiene todos los valores registrados en ese intervalo de tiempo y permite operar directamente sobre ellos. Cada objeto `Signal` representa una serie de puntos de tipo `Point<Long>`, ordenados cronológicamente, e incluye los siguientes atributos y operaciones principales:

- `from()`: instante de tiempo del primer valor de la serie.
- `to()`: instante de tiempo del último valor de la serie.
- `duration()`: duración total de la serie.
- `points()`: lista completa de puntos de la serie, cada uno con su valor, instante y origen (`feed`).
- `values()`: array de `long` con los valores numéricos de todos los puntos, sin metadatos.

Además de acceder a los valores individuales de una serie temporal, también es posible realizar análisis estadístico mediante los métodos `summary()` y `distribution()`.

El método `summary()` calcula estadísticas básicas sobre la serie y devuelve un objeto `Summary` con los siguientes datos:

- `count`: número total de observaciones.
- `sum`: suma total de los valores.
- `mean`: media aritmética.
- `sd`: desviación estándar.
- `min`: punto con el valor mínimo registrado, incluyendo el instante y el feed.
- `max`: punto con el valor máximo registrado, también con metadatos temporales.

Por otro lado, el método `distribution()` construye una distribución probabilística basada en `TDigest`, una estructura eficiente para el cálculo de estadísticas de orden, especialmente útil en grandes volúmenes de datos. A través del objeto `Distribution` es posible obtener:

- `quantile(p)`: valor correspondiente al percentil `p`, con `p` entre 0 y 1.
- `q1()`, `q2()` (o `median()`), `q3()`: primer, segundo y tercer cuartil.
- `probabilityLeftTail(value)`: probabilidad acumulada de que un valor observado sea menor o igual al indicado.
- `probabilityRightTail(value)`: probabilidad de observar un valor superior al indicado.

7.2. Categóricas: `Sequence`

También es posible obtener un resumen estadístico sobre los valores categóricos registrados en un intervalo. Este resumen incluye métricas como:

- `count()`: número total de elementos registrados.
- `mode()`: valor más frecuente.
- `categories()`: conjunto de valores distintos observados.
- `frecuency(category)`: frecuencia absoluta de cada valor.

```
var summary = store.categoricalQuery("RiesgoCardiovascular")
                  .sequence(today(-90), today())
                  .summary();

int count = summary.count();
String mode = summary.mode();
Set<String> categories = summary.categories();
int frecuency = summary.frecuency("Moderado");
```

8. Segmentación de series temporales

Las series temporales, tanto numéricas (`Signal`) como categóricas (`Sequence`), pueden dividirse en segmentos más pequeños para facilitar análisis periódicos o localizados. Existen dos métodos principales para segmentar:

- `segments(Duration duration)`: divide la serie en tramos consecutivos de duración fija. Por ejemplo, segmentos diarios, semanales o por hora.
- `segments(int number)`: divide la serie en un número fijo de segmentos de igual longitud temporal.

Cada segmento resultante es una serie temporal del mismo tipo que la original, lo que permite aplicar directamente sobre él los mismos métodos de consulta y análisis, así como volver a segmentarlo si es necesario. Por ejemplo, en el caso de una serie temporal numérica (`Signal`), cada segmento también es un `Signal`, por lo que admite operaciones como `summary()`, `distribution()`. Del mismo modo, una secuencia categórica (`Sequence`) segmentada produce otras instancias de `Sequence`, que pueden consultarse con `mode()`, `count()`, `categories()`.

```
Signal[] segments = hemoglobina.segments(Duration.ofDays(1));
for (Signal segment : segments) {
    var median = segment.distribution().median();
}
```

9. Generación de tablas

La generación de tablas tiene como objetivo producir una vista agregada de los datos en función del tiempo, donde cada fila representa un segmento temporal y cada columna contiene un valor agregado correspondiente a ese intervalo. El primer paso para construir una tabla es definir su formato mediante la clase `Format`, en el que se indica:

- El intervalo y la resolución temporal (por ejemplo, diaria, semanal o mensual), que determinan las filas de la tabla.
- Las columnas que se desean incluir, junto con el operador de agregación o transformación asociado a cada una.

Las columnas pueden ser de tres tipos principales: temporales, numéricas o categóricas.

- Las columnas temporales se derivan directamente de la marca temporal y permiten extraer propiedades como el día del mes, el mes del año, el día de la semana o combinaciones como año-mes o año-trimestre. Estas columnas no dependen de ningún atributo registrado, sino únicamente del instante asociado a cada registro.
- Las columnas numéricas realizan operaciones estadísticas sobre atributos de tipo numérico, como el promedio, la suma o el valor mínimo o máximo.
- Las columnas categóricas operan sobre atributos de tipo texto y permiten incluir la moda (valor más frecuente) o la entropía, que mide la variabilidad de los valores categóricos.

El siguiente ejemplo genera un informe semanal entre dos fechas, incluyendo columnas temporales, numéricas y categóricas:

```
try (SubjectStore store = new SubjectStore(File.createTempFile("
    patient", ".gss"))) {
    Instant from = Instant.parse("2025-01-01T00:00:00Z");
    Instant to = Instant.parse("2025-03-01T00:00:00Z");

    Format format = new Format(from, to, Duration.ofDays(7))
        .add(new Column.Temporal(DayOfMonth))
        .add(new Column.Temporal(MonthOfYear))
        .add(new Column.Temporal(DayOfWeek))
        .add(new Column.Numerical("temperature", Average))
        .add(new Column.Categorical("sky", Mode));

    SubjectReport report = new SubjectReport(store, format);
    report.export(new File("view.tsv"));
}
```

El resultado del ejemplo se muestra en la siguiente tabla. Cada fila representa un segmento semanal y las columnas corresponden a operadores aplicados sobre los atributos definidos en el `Format`. El archivo generado (`view.tsv`) no contiene encabezados: únicamente los datos en formato tabulado por filas y columnas.

11	1	4	19.0	clear
13	1	6	25.0	clear
17	1	3	22.5	clear
23	1	2	24.0	cloudy
29	1	1	18.5	cloudy
2	2	5	21.0	overcast
8	2	6	20.0	rain
14	2	4	23.3	rain
20	2	2	19.5	cloudy
26	2	3	24.2	clear