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 asunto), 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 en su diseño sobre la forma de 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 `12345`. Este paciente posee atributos que cambian cada vez que se realiza una prueba médica.

Por ejemplo, cuando al paciente se le realiza un hemograma, se tienen que registrar nuevos datos sobre diferentes parámetros sanguíneos (glóbulos rojos, glóbulos blancos, hemoglobina, plaquetas, etc.). El número 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.

2. Prerrequisitos

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

- **Java 19 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>systems.intino.datamarts</groupId>
    <artifactId>subject-store</artifactId>
    <version>1.0.0-SNAPSHOT</version>
</dependency>
```

4. Carga de datos

Para iniciar la carga de datos en un `SubjectStore`, es necesario instanciar la clase proporcionando un nombre en el formato `id:tipo`, que identifica de forma única al sujeto y su tipo. Adicionalmente, se puede indicar un archivo o una conexión a base de datos donde almacenar la información. Si no se especifica ninguno, los datos se almacenan en memoria. El nombre del sujeto debe seguir el formato `id:tipo`, por ejemplo: `"12345:paciente"`.

- `new SubjectStore(name)` crea un almacén en memoria.
- `new SubjectStore(name, file)` almacena los datos en el archivo indicado.
- `new SubjectStore(name, connection)` utiliza una conexión SQL existente.

Los datos se introducen mediante el método `feed`, al que se le indica un instante temporal y una cadena que describe la fuente o el contexto del dato. Luego, se añaden atributos con el método `add`, especificando un par `tag-value`. Finalmente, se cierra la entrada con `terminate`. Los valores numéricos se introducen como `double` (coma flotante), y los atributos categóricos como cadenas de texto.

En el siguiente ejemplo se crea un `SubjectStore` asociado a un archivo. Si el archivo no existe, será creado automáticamente.

```
File file = new File("...");
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    Instant instant = Instant.parse("2025-03-23T09:30:00Z");
    store.feed(instant, "Laboratorio.Hemograma.0512301")
         .add("Hemoglobina", 14.2)
         .add("Plaquetas", 250_000)
         .add("GlobulosBlancos", 5_600)
         .add("GrupoSanguineo", "A+")
         .terminate();
}
```

Si una segunda fuente quiere aportar atributos adicionales en otro instante, simplemente se repite el procedimiento:

```
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    Instant instant = Instant.parse("2025-03-25T11:00:00Z");
    store.feed(instant, "Cardiologia.DrCabrera")
         .add("PresionSistolica", 125)
         .add("PresionDiastolica", 80)
         .add("RiesgoCardiovascular", "Moderado")
         .terminate();
}
```

Este el siguiente ejemplo se muestra cómo crear un `SubjectStore` en memoria, sin necesidad de un archivo ni una conexión a base de datos. Es especialmente útil en contextos donde no se requiere persistencia, como pruebas unitarias, simulaciones, o procesos de análisis temporal intermedios. Al no depender del sistema de archivos, permite crear, modificar y descartar datos de forma rápida y eficiente.

```
try (SubjectStore store = new SubjectStore("12345:paciente")) {
    store.feed(Instant.now(), "Consulta.Urgencias")
         .add("Temperatura", 37.8)
         .add("FrecuenciaCardiaca", 88)
         .terminate();
}
```

Cuando se desea registrar múltiples aportaciones en una única operación, se puede utilizar el modo `batch`. Este permite agrupar varias llamadas a `feed`, `add` y `terminate` bajo una única sesión, optimizando la inserción de datos históricos o masivos.

```
try (SubjectStore store = new SubjectStore("12345:paciente", file)) {
    SubjectStore.Batch batch = store.batch();

    batch.feed(day1, "HMG-2").add("Hemoglobin", 145).terminate();
    batch.feed(day2, "HMG-1").add("Hemoglobin", 130).terminate();
    batch.feed(day3, "HMG-B").add("Hemoglobin", 115).terminate();
    batch.feed(day4, "HMG-L").add("Hemoglobin", 110).terminate();

    batch.terminate();
}
```

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 `TemporalReferences` 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).

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.

```
try (SubjectStore store = new SubjectStore("12345:paciente", 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`.
- `id()`: devuelve el id del `Subject`.

- `type()`: devuelve el tipo del `Subject`.
- `size()`: retorna el número total de (`feeds`) registrados.
- `ss(feed)`: devuelve la etiqueta asociada a un `feed` concreto, útil para identificar su origen o propósito.
- `from()`: devuelve el instante de tiempo correspondiente al primer `feed` cronológico real.
- `to()`: devuelve el instante de tiempo correspondiente al último `feed` cronológico real.
- `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:

- `currentNumber(tag)`: devuelve directamente el valor numérico más reciente del atributo dado (o `null` si no existe).
- `currentText(tag)`: devuelve directamente el valor categórico más reciente del atributo dado (o `null` si no existe).
- `numericalQuery(tag)`: devuelve un objeto `NumericalQuery` si el atributo es numérico. Este objeto permite acceder a todos los valores históricos o aplicar operaciones estadísticas sobre los datos.
- `categoricalQuery(tag)`: devuelve un objeto `CategoricalQuery` si el atributo es de tipo texto. Permite acceder al conjunto de valores distintos observados.

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 `get()`, que devuelve un objeto `Point<Double>` 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. El valor devuelto por `get()` corresponde siempre al dato más reciente dentro del historial cronológico real.

```
Point<Double> hemoglobina = store
    .numericalQuery("Hemoglobina")
    .get();

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

Para consultar todos los valores registrados se usa el método `getAll()`. Para consultar los valores en un período determinado, se usa el método `get(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")
    .getAll();
```

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 `get()`, que devuelve un objeto `Point<String>` con el valor, el instante y el identificador del `feed` correspondiente. El resultado de `get()` refleja el valor más reciente dentro del historial cronológico.

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

Para consultar todos los valores registrados se utiliza el método `getAll()`. Para consultar los valores registrados en un rango de fechas, se utiliza el método `get(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")
    .get(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 métodos de `get()`, que devuelven una instancia de `Signal`. Esta serie contiene todos los valores registrados en un intervalo de tiempo y permite operar directamente sobre ellos. Cada objeto `Signal` representa una serie de puntos de tipo `Point<Double>`, 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 `double` 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, a través de la cual 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")
                  .get(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.

En Java, la duración se representa mediante objetos que implementan la interfaz `TemporalAmount`. Si la duración es inferior a un día (por ejemplo, horas, minutos o segundos), se debe utilizar la clase `Duration`, con métodos como `Duration.ofHours(1)` o `Duration.ofMinutes(30)`. En cambio, para unidades de tiempo mayores o basadas en el calendario, como meses o años, se utiliza la clase `Period`, con métodos como `Period.ofMonths(1)` o `Period.ofYears(1)`.

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. Exportación de datos con `dump`

Una vez registrados los datos en un `SubjectStore`, es posible exportarlos en formato de texto plano utilizando el método `dump`. Este método permite volcar el contenido completo del `SubjectStore` a un `OutputStream`, generando una representación legible de los eventos registrados, que puede ser utilizado para inspección manual, depuración, trazabilidad o integración con otros sistemas.

```
File file = new File("...");
try (SubjectStore store = new SubjectStore("12345:paciente", file),
    OutputStream os = new FileOutputStream(os)) {
    store.dump(os);
}
```

La salida generada por `dump` será similar a la siguiente:

```
[paciente]
ts=2025-03-25T00:00:00Z
ss=HMG-2
Hemoglobin=145.0
[paciente]
ts=2025-03-28T00:00:00Z
ss=HMG-2
Hemoglobin=130.0
...
```

Cada bloque de datos exportado mediante el método `dump` comienza con una línea de encabezado que indica el tipo del sujeto entre corchetes.

10. Generación de vistas

La generación de vistas permite transformar los datos registrados en tablas agregadas organizadas por intervalos temporales. Cada fila de la tabla re-

presenta un segmento de tiempo (por ejemplo, una semana), y cada columna muestra el resultado de una operación evaluada sobre los datos pertenecientes a ese intervalo. Las columnas se crean mediante objetos `Column`, los cuales requieren una especificación. Existen tres tipos principales de definiciones de columnas: agregaciones, marcas temporales y fórmulas.

10.1. Agregaciones

Las agregaciones son definiciones con la forma `nombre = operador:tag`. Se usan para extraer datos registrados asociados a un determinado tag. Requieren especificar un operador que resuma los valores dentro de cada intervalo temporal.

Operador	**Tipo**	**Descripción**
`count`		Número de observaciones
`sum`	Numérica	Suma total
`average`	Numérica	Media aritmética
`sd`, `standard-deviation`	Numérica	Desviación estándar
`first`	Numérica	Primer valor observado
`last`	Numérica	Último valor observado
`min`	Numérica	Valor mínimo
`max`	Numérica	Valor máximo
`mode`	Categórica	Valor más frecuente
`entropy`	Categórica	Dispersión o diversidad de valores

10.2. Marcas temporales

Las marcas temporales son definiciones con la forma `nombre = marca`. Se derivan directamente del instante temporal de cada registro.

Marca Temporal	Descripción
`day-of-week`	Día de la semana (1–7)
`day-of-month`	Día del mes (1–31)
`month-of-year`	Mes del año (1–12)
`year`	Año (YYYY)
`quarter-of-year`	Trimestre (1–4)
`year-quarter`	Formato `YYYYQX`
`year-month`	Formato `YYYYMM`
`year-month-day`	Formato `YYYYMMDD`
`year-month-day-hour`	Formato `YYYYMMDDHH`
`year-month-day-hour-minute`	Formato `YYYYMMDDHHmm`
`year-month-day-hour-minute-second`	Formato `YYYYMMDDHHmmss`

10.3. Fórmulas

Las columnas también pueden definirse a partir de otras columnas ya existentes, utilizando expresiones matemáticas, operadores y funciones. Estas definiciones tienen la forma `nombre = expresión`, donde la expresión puede combinar operaciones aritméticas con funciones. Algunos ejemplos:

- `IMC = Peso / (Altura * Altura)`
- `PAM = (PresionSistolica + 2 * PresionDiastolica) / 3`
- `PresionLog = log(PresionSistolica + 1)`
- `RiesgoCombinado = ColesterolTotal / HDL`
- `HemoglobinaNormalizada = Hemoglobina / Peso`
- `CreatininaNorm = sqrt(Creatinina)`

Función	Descripción	Función	Descripción
`abs`	Valor absoluto	`sin`	Seno
`negate`	Opuesto del valor (`-x`)	`cos`	Coseno
`round`	Redondeo	`tan`	Tangente
`floor`	Redondeo hacia abajo	`asin`	Arcoseno
`ceil`	Redondeo hacia arriba	`acos`	Arcocoseno
`signum`	Signo del número	`atan`	Arcotangente
`exp`	Exponencial e^x	`sinh`	Seno hiperbólico
`log`	Logaritmo natural ($\ln x$)	`cosh`	Coseno hiperbólico
`log10`	Logaritmo en base 10	`tanh`	Tangente hiperbólica
`sqrt`	Raíz cuadrada	`rad`	Grados a radianes
`cbrt`	Raíz cúbica	`deg`	Radianes a grados

10.4. Filtros

Las columnas pueden incorporar **filtros**, que son transformaciones aplicadas sobre el resultado de una columna numérica. Los filtros permiten modificar, suavizar o normalizar los valores calculados antes de ser representados en la vista. Su uso es especialmente útil para destacar tendencias, eliminar ruido o preparar datos. Los filtros se aplican de forma encadenada mediante el método `.add(Filter)` sobre una columna previamente definida.

Filtro	Descripción
`BinaryThresholdFilter`	Binariza dado un umbral
`SinFilter`	Aplica la función seno
`CosFilter`	Aplica la función coseno
`CumulativeSumFilter`	Calcula la suma acumulativa
`DifferencingFilter`	Diferencia entre elementos consecutivos
`LagFilter`	Desplaza una cantidad fija de pasos
`MinMaxNormalizationFilter`	Normaliza entre 0 y 1
`ZScoreNormalizationFilter`	Normaliza por Z-score
`RollingAverageFilter`	Media móvil
`RollingSumFilter`	Suma móvil
`RollingStandardDeviationFilter`	Desviación estándar móvil
`RollingMaxFilter`	Máximo móvil
`RollingMinFilter`	Mínimo móvil

10.5. Ejemplos

A continuación, se muestra un ejemplo de generación de una vista semanal, combinando atributos clínicos numéricos y categóricos, junto con fórmulas derivadas y filtros.

```
Format format = new Format(from, to, Duration.ofDays(7));
format.add(new Column("Year=year"));
format.add(new Column("Month=month-of-year"));
format.add(new Column("Day=day-of-month"));

format.add(new Column("SysSum=sum:PresionSistolica"));
format.add(new Column("SysAvg=average:PresionSistolica"));
format.add(new Column("SysTrend=SysAvg")
    .add(new RollingAverageFilter(3)));

format.add(new Column("TempAvg=average:Temperatura"));
format.add(new Column("TempNorm=TempAvg")
    .add(new MinMaxNormalizationFilter()));
format.add(new Column("TempScore=TempNorm * 100"));

format.add(new Column("LastHR=last:FrecuenciaCardiaca"));
format.add(new Column("Riesgo=mode:RiesgoCardiovascular"));
format.add(new Column("Medidas=count:Temperatura"));
```

El resultado es una tabla semanal sin encabezados, donde cada fila representa una semana y cada columna corresponde a un valor clínico agregado, normalizado o derivado. A continuación, se muestra un ejemplo de salida:

2025	1	1	0.0	0.0		0.0	0.0			0.0
2025	1	8	240.0	120.0	120.0	36.8	1.0	88	Moderado	3.0
2025	1	15	0.0	0.0	60.0	0.0	0.0			0.0
2025	1	22	0.0	0.0	90.0	0.0	0.0			0.0
2025	1	29	120.0	120.0	110.0	36.5	0.75	70	Bajo	2.0