



Proyecto de laboratorio - Java

Estructura general del proyecto

- a) El proyecto debe contener tres carpetas: src, data y doc.
- b) La carpeta src debe incluir el código fuente, organizado en paquetes, la carpeta data el fichero de trabajo que más adelante se le proporcionará y, por último, la carpeta doc donde se guardarán la documentación del proyecto.
- c) El proyecto debe contener un fichero README.md donde se describirán los datos y los tipos implementados.

Contenido

Entrega 1.....	2
Entrega 2.....	6
Entrega 3.....	9



Entrega 1

1) Diseño moderno¹

Cree un paquete de nombre `fp.clinico` y programe los siguientes tipos como records.

Persona

- Propiedades:
 - nombre, de tipo `String`.
 - apellidos, de tipo `String`.
 - dni, de tipo `String`.
 - fecha de nacimiento, de tipo `LocalDate`.
 - edad, de tipo `Integer`. (Derivada a partir de la fecha de nacimiento).
- Restricciones:
 - La fecha de nacimiento debe ser anterior a la fecha actual.
 - El dni debe ser una cadena con ocho dígitos y seguidos de una letra.
- Representación como cadena: por defecto asociado al record.
- Criterio de igualdad: por defecto asociado al record.
- Orden natural: por dni.
- Comentarios: añada los siguientes métodos de factoría:
 - Método static of:
 - recibe nombre, apellidos, dni y fecha de nacimiento y devuelve una persona.
 - Método static parse:
 - Recibe una cadena con un formato específico y devuelve una persona.
Ejemplo de cadena: "Juan, García Rodríguez, 12755078Z, 20/03/1965".
 - Incluya un método main para comprobar el correcto funcionamiento del método parse. `public static void main(String[] args){ ... }`

Paciente

- Propiedades:
 - persona, de tipo `Persona`.
 - código de ingreso, de tipo `String`.
 - fecha y hora de ingreso, de tipo `LocalDateTime`.
 - fecha de ingreso, de tipo `LocalDate`. (Derivada a partir de la fecha y hora de ingreso)
 - hora de ingreso, de tipo `String`. (Derivada a partir de la fecha y hora de ingreso).
Ejemplo de cadena: "15:03". Es decir, la hora y los minutos tienen que tener dos dígitos. No valdría escribir "15:3".
- Restricciones:
 - La fecha y hora de ingreso debe ser anterior o igual a la fecha actual.
- Representación como cadena: por defecto asociado al record.
- Criterio de igualdad: por defecto asociado al record.
- Comentarios: añada los siguientes métodos de factoría:
 - Método static of:

¹ Siguiendo las pautas de diseño vistas en clase de teoría con Miguel Toro.



- recibe nombre, apellidos, dni, fecha de nacimiento, código y fecha y hora de ingreso y devuelve un paciente.
- Método static of:
 - recibe un objeto persona, un código y una fecha y hora de ingreso y devuelve un paciente.

PacienteEstudio

- Propiedades:
 - id, de tipo String.
 - genero, de tipo String.
 - edad, de tipo Double.
 - hipertensión, de tipo Boolean.
 - enfermedad del corazón, de tipo Boolean.
 - tipo de residencia, enumerado TipoResidencia, cuyos valores son rural o urbana.
 - nivel medio de glucosa, de tipo Double.
 - factor de riesgo, de tipo Boolean. (Derivada, si tiene hipertensión y más de 40 años se considerará que tiene factor de riesgo).
- Restricciones:
 - La edad tiene que ser mayor o igual que cero y menor o igual que 130.
 - El nivel medio de glucosa tiene que ser mayor o igual que cero.
- Representación como cadena: informa del id y la edad del paciente.
- Criterio de igualdad: por defecto asociado al record.
- Criterio de orden: según la edad y el id.
- Comentarios: añada los siguientes métodos de factoría:
 - Método static of:
 - recibe valores para cada propiedad básica y devuelve un objeto del tipo.
 - Método static parse:
 - recibe una cadena con un formato especificado y devuelve un objeto del tipo. Ejemplo de cadena: "6306;Male;80;false;false;URBANA;83.84"
 - Incluya un método main para comprobar el correcto funcionamiento del método parse. *public static void main(String[] args){ ... }*

Cree un paquete de nombre **fp.vacunas** y programe el siguiente tipo como un record.

Vacunacion

- Propiedades:
 - fecha, de tipo LocalDate.
 - comunidad, de tipo String.
 - pfizer, de tipo Integer.
 - moderna, de tipo Integer.
 - astrazeneca, de tipo Integer.
 - janssen de tipo Integer.
 - número de personas, de tipo Integer.



- número total, de tipo Integer. (Derivada, siendo la suma de dosis de Pfizer, moderna, astrazeneca y janssen).
- Restricciones:
 - La fecha de debe ser posterior al 01/02/2021.
- Representación como cadena: por defecto asociado al record.
- Criterio de igualdad: por defecto asociado al record.
- Orden natural: por comunidad y en caso de igualdad por fecha.
- Comentarios: añade los siguientes métodos de factoría:
 - Método static of:
 - recibe valores para cada propiedad básica y devuelve un objeto del tipo.
 - Método static parse:
 - recibe una cadena con un formato específico y devuelve un objeto del tipo.
Ejemplo de cadena: “04/01/2021;Andalucía;140295;0;0;0;0”.
 - Incluya un método main para comprobar el correcto funcionamiento del método parse. *public static void main(String[] args){ ... }*

2) Diseño clásico²

Cree un paquete de nombre `fp.farmaceutico` y programe el siguiente tipo como una clase.

Medicamento

- Propiedades:
 - nombre del medicamento, de tipo String, observable.
 - tipo de medicamento, enumerado de tipo TipoMedicamento, observable. Los valores del enumerado son anatómico, químico y terapéutico.
 - código de la enfermedad, de tipo String, observable.
 - farmacéutica, de tipo String, observable.
 - puntuación, de tipo Double, observable.
 - índice somático, de tipo Integer, observable.
 - fecha de catálogo, de tipo LocalDate, **observable y modificable**.
 - tratar enfermedad, de tipo Boolean. (Derivada, siendo cierta si el código de la enfermedad coincide con un parámetro de tipo cadena que reciben como argumento la propiedad).
- Restricciones:
 - La puntuación tiene que ser mayor estricta que cero.
 - El índice somático tiene que ser mayor o igual que 1000.
 - La fecha de catálogo tiene que ser posterior al 01/01/2015.
- Representación como cadena: según el nombre del medicamento y de la farmacéutica.
- Criterio de igualdad: por nombre del medicamento y farmacéutica.
- Orden natural: por nombre del medicamento y en caso de igualdad por la farmacéutica.
- Comentarios:

² Los records fueron introducidos en la versión 14 de Java – el JDK14 – que salió a la luz el 17 de marzo de 2020. Llamaremos *diseño clásico* al diseño que se solía hacer hasta ese momento. En general, no se insistía en el uso de tipos inmutables para los tipos simples – que teniendo records se programan con mucha comodidad – ni, por otro lado, se utilizaban una serie de pautas o patrones software a la hora de programar los tipos.



- Programe una **clase de nombre FactoriaMedicamentos** que incluya, de momento, un método static de nombre **parseaMedicamento**, que recibe una cadena con un formato específico y devuelve un objeto de tipo Medicamento.

Ejemplo:

“efavirenz,Anatomico,Y212XXA,Actavis Mid Atlantic LLC,90.0,1848,04/12/2019”.

(Siendo la información del nombre del medicamento, el tipo de medicamento, el código de la enfermedad, la farmaceutica, la puntuacion, el índice somatico, y la fecha de catalogo)

- Implemente una clase de nombre TestFactoriaMedicamentos en un paquete de nombre fp.farmaceutico.test y compruebe el correcto funcionamiento del método anterior.



Entrega 2

Se le suministran los siguientes ficheros que deberá guardar en la carpeta data:

- *estudio_clinico.csv*: contiene la información sobre objetos del tipo PacienteEstudio.
- *medicamentos.csv*: contiene la información sobre objetos del tipo Medicamento.
- *ccaa_vacunas_3.csv*: contiene la información sobre objetos del tipo Vacunacion.

Así mismo, **se le proporciona la siguiente interfaz** que debe incorporar en el paquete `fp.clinico`. Para ello, cree una nueva interfaz con el nombre indicado – **EstudioClinico** – y copie los métodos que aparecen en el enunciado. Esta interfaz va a tener dos implementaciones: una de ellas de tipo imperativa, que llamaremos **EstudioClinicoBucles**, y otra de tipo funcional, que llamaremos **EstudioClinicoStream**.

```
public interface EstudioClinico {
    // Propiedades de lista
    Integer numeroPacientes();
    void incluyePaciente(PacienteEstudio paciente);
    void incluyePacientes(Collection<PacienteEstudio> pacientes);
    void eliminaPaciente(PacienteEstudio paciente);
    Boolean estaPaciente(PacienteEstudio paciente);
    void borraEstudio();
    //
    // Método de factoría
    EstudioClinico of(String nombreFichero);
    List<PacienteEstudio> leeFichero(String nombreFichero);
    // Tratamientos secuenciales: implementaciónn funcional vs. imperativa
    //existe, paraTodo
    Boolean todosPacienteSonDelTipo(TipoResidencia tipo);
    Boolean existeAlgunPacienteDelTipo(TipoResidencia tipo);
    //contador, suma, media
    Integer numeroPacientesFactorRiesgo();
    Double edadMediaPacientesConFactorRiesgo();
    //filtrado
    List<PacienteEstudio> filtraPacientesPorEdad(Double edad);
    //devuelve Map que agrupa
    Map<String,List<PacienteEstudio>> agruparPacientesEdadMayorQuePorGenero(Double edad);
    //devuelve Map que realiza un cálculo
    Map<String,Long> numeroPacientesPorGenero();
    Map<String,Double> edadMediaPacientesPorPorGenero();
}
```

Detalles: una vez incorporada la interfaz en el paquete `fp.clinico`, construya las `EstudioClinicoBucles` y `EstudioClinicoStream`.

Al hacerlo, `new/class/` se pone el nombre de la clase y, en el recuadro de “Interfaces” se añade el nombre de la interfaz mediante el botón Add. De esta manera se crea la estructura de la clase con todos los métodos de la interfaz con TODOS. Al ir programando se van borrando poco a poco las marcas azules de los TODOS.

- Al realizar la clase `TestEstudioClinicoBucles`, o la clase `TestEstudioClinicoStream` – que creará en la Entrega 3 –, cuando necesite crear un objeto. Bastará utilizar el constructor de la clase de la siguiente manera:
`EstudioClinico est = new EstudioClinicoBucles(---);`
- Observe que el tipo debe tener un atributo que sea un objeto del tipo `List<PacienteEstudio>`
- Los métodos son todos propiedades derivadas u operaciones.



- El método `leeFichero` es un método auxiliar que carga el fichero correspondiente en una lista de objetos del tipo. El método `of` realiza una operación parecida (de hecho, al programar hará una llamada al método `leeFichero`), pero construye un objeto del tipo `EstudioClinico`.
- Los nombres de los métodos son lo suficientemente explicativos para poder comprender qué debe hacer cada uno.
- Observe que los métodos `of` y `leeFichero` son métodos normales, no son `static`. Por lo tanto, para poder llamarlos tiene que construir antes un objeto `EstudioClinico`. Por este motivo, se debe hacer un constructor sin parámetros que construya un objeto sin contenido y cuya misión es poder utilizar estos métodos.

Se debe programar:

1) Factorías

Hay dos maneras de hacer las factorías. En ambos casos la programación es similar, aunque el diseño – es decir, la manera de estructurar el código – son diferentes.

a) Como un método más dentro de un tipo que contiene otros tratamientos secuenciales.

Este diseño es más adecuado para proyectos grandes que serán mantenidos a lo largo del tiempo. Por ejemplo, es la manera de trabajar en la API de Java.

Debe programar:

- Método `of` y método `leeFichero` de la clase `EstudioClinicoBucles` que implementa a la interfaz `EstudioClinico`.
Para ello tiene que haber creado previamente dicha clase y haberle incorporado una lista de objetos del tipo `PacienteEstudio`, así como los constructores:
 - Constructor vacío, que construya la lista sin ningún elemento.
 - Constructor que reciba una lista de objetos del tipo `PacienteEstudio` como parámetro.
 - Nota: deberá utilizar el método `parse` que, dada una línea del fichero, construya un objeto del tipo `PacienteEstudio` y que programó en la entrega 1.
- Realice una comprobación de que la programación funciona bien en una clase de nombre `TestEstudioClinicoBucles` que testee la lectura del fichero.

b) Como una clase independiente cuya una misión es construir listas de objetos.

Este diseño es más adecuado para proyectos pequeños. Por ejemplo, para un examen que solo dura unas horas.

Debe programar:

- Una clase de nombre `FactoriaVacunaciones` que contenga un método de nombre `leeFichero` que, dada una cadena con el nombre del fichero, devuelva una lista de objetos `Vacunacion`. Deberá programar un método auxiliar que parsee cada línea del fichero.
 - Realice una comprobación en una clase de testeo que lo programado funciona correctamente.
- Una clase de nombre `FactoriaMedicamentos` que contenga un método de nombre `leeFichero` que, dada una cadena con el nombre del fichero, devuelva una



lista de objetos Medicamento. Deberá programar un método auxiliar que parsee cada línea del fichero.

- Realice una comprobación en una clase de testeo que lo programado funciona correctamente.

2) Métodos de la clase EstudioClinicoBucles

Programa **todos los métodos de la clase EstudioClinicoBucles**, teniendo en cuenta que son métodos:

- Métodos con propiedades de listas.
- Métodos de tratamientos secuenciales:
 - Operaciones de existencia, si todos los objetos verifican una propiedad, contadores, cálculos de una media.
 - Operación de filtrado en una lista de Pacientes
- Métodos para la construcción de diccionarios o mapas:
 - Construcción de diccionarios que agrupan.
 - Construcción de diccionarios que sirven para hacer un cálculo múltiple. (Por ejemplo. contador múltiple).

Programa tan solo todos los métodos de las propiedades de lista de la clase **EstudioClinicoStream** y deje el resto con los TODOs, que haremos en la siguiente entrega.



Entrega 3

En esta entrega se programará todo lo relativo a la **programación en streaming**.

1) Implementación funcional del tipo EstudioClinico: métodos de la clase **EstudioClinicoStream**

Programa **todos los métodos de la clase EstudioClinicoStream**, teniendo en cuenta que los métodos iniciales son exactamente iguales que en la clase EstudioClinicoBucles – métodos de propiedades de lista y métodos de factoría - y que los métodos de tratamientos secuenciales deben programarse utilizando operaciones del tipo Stream<T>

- Nota: se recomienda utilizar la opción `Collectors.groupingBy(- , -);` para los métodos que construyan diccionarios.

Recordatorio de la interfaz

```
public interface EstudioClinico {  
    // Propiedades de lista  
    Integer numeroPacientes();  
    void incluyePaciente(PacienteEstudio paciente);  
    void incluyePacientes(Collection<PacienteEstudio> pacientes);  
    void eliminaPaciente(PacienteEstudio paciente);  
    Boolean estaPaciente(PacienteEstudio paciente);  
    void borraEstudio();  
    //  
    // Método de factoría  
    EstudioClinico of(String nombreFichero);  
    List<PacienteEstudio> leeFichero(String nombreFichero);  
    // Tratamientos secuenciales: implementación funcional vs. imperativa  
    // existe, paraTodo  
    Boolean todosPacienteSonDelTipo(TipoResidencia tipo);  
    Boolean existeAlgunPacienteDelTipo(TipoResidencia tipo);  
    // contador, suma, media  
    Integer numeroPacientesFactorRiesgo();  
    Double edadMediaPacientesConFactorRiesgo();  
    // filtrado  
    List<PacienteEstudio> filtraPacientesPorEdad(Double edad);  
    // devuelve Map que agrupa  
    Map<String, List<PacienteEstudio>> agruparPacientesEdadMayorQuePorGenero(Double edad);  
    // devuelve Map que realiza un cálculo  
    Map<String, Long> numeroPacientesPorGenero();  
    Map<String, Double> edadMediaPacientesPorPorGenero();  
}
```

2) Diseño clásico o para proyectos pequeños

a) Implemente una clase que se llame **Vacunaciones** en el paquete `fp.vacunas`.

Incluya un único constructor que reciba un Stream<Vacunacion> e inicialice el atributo (que es un List<Vacunacion>) con los objetos de dicho Stream. No es necesario implementar representación como cadena ni criterio de igualdad.

Implemente los siguientes métodos:

- **anyadeVacunacion**: dado un objeto del tipo Vacunacion lo añade al atributo de List<Vacunacion>.



- ***vacunacionesEntreFechas***: dadas dos fechas como parámetros de entrada, devuelve una lista con aquellas vacunaciones entre dichas fechas.
- ***existeNumPersonasPautaCompletaPorEncimaDe***: dada una comunidad y un valor entero, indica si existen o no vacunaciones con un número de personas con la pauta completa de vacunación por encima del valor entero dado.
- ***diaMasVacunacionesEn***: dada una comunidad, devuelve la fecha en la que hubo más personas vacunadas.
- ***vacunacionesPorFecha***: devuelve un mapa, o diccionario, en el que las claves son las fechas y los valores son listas de vacunaciones asociadas a dichas fechas.
- ***maximoNumTotalVacunasporComunidad***: devuelve un mapa, o diccionario, en el que las claves son las comunidades y los valores son el máximo para el número total de vacunas puestas para cada comunidad.

b) Implemente una clase que se llame **ListadoMedicamentos** en el paquete **fp.farmaceutico**.

Implemente la clase ListadoMedicamentos. Incluya un único constructor que reciba un `Stream<Medicamento>` e inicialice el atributo con los objetos de dicho Stream. No es necesario implementar representación como cadena ni criterio de igualdad. Implemente en la clase los siguientes métodos:

- ***existeMedicamentoSegunTipoAnteriorA***: dado un tipo de medicamento y una fecha, indica si existe un medicamento de dicho tipo posterior a la fecha dada.
- ***nombreMedicamentosPuntuacionMayorA***: dada una puntuación, devuelve un conjunto con los nombres de los medicamentos con una puntuación mayor a la dada.
- ***nombreMedicamentoMayorIndiceSomaticoSegunTipoMedicamento***: dado un tipo de medicamento, devuelve el nombre del medicamento con mayor índice somático. En caso de no haber ninguno, se eleva una excepción.
- ***agrupaTipoMedicamentoSegunPuntuacionMedia***: devuelve un diccionario que asocia a cada tipo de medicamento su puntuación media.
- ***fechaCatalogoMasFrecuente***: devuelve la fecha del catálogo más frecuente, es decir, la que aparece más veces.