FUNDAMENTOS DE PROGRAMACIÓN – PROGRAMACIÓN ORIENTADA A OBJETOS

Proyecto: Comics de Marvel y DC

Autor: José A. Troyano
Revisor: Daniel Mateos
Última modificación: 08/04/2019

1 Material

Para la realización de esta práctica se dispone de los siguientes elementos contenidos en el fichero zip:

- **/doc/Enunciado.pdf**: fichero PDF con este enunciado
- **/data/**: carpeta de datos
  - **/data/characters.csv**: fichero CSV con datos de personajes de comic de Marvel y DC, el dataset original se puede encontrar en kaggle
- **/src/fp.comics.test**: paquete Java con las clases de test para las distintas clases que habrá que desarrollar en el proyecto
- **/src/fp.utiles**: paquete Java con utilidades de la asignatura

2 Datos disponibles

En este proyecto trabajaremos sobre datos de personajes de comic. En estos datos encontramos solo un tipo de entidad:

- **Personaje:** contiene información relativa a un personaje de comic

Los datos están disponibles en formato CSV. En la siguiente figura se muestran las primeras líneas del fichero de datos.

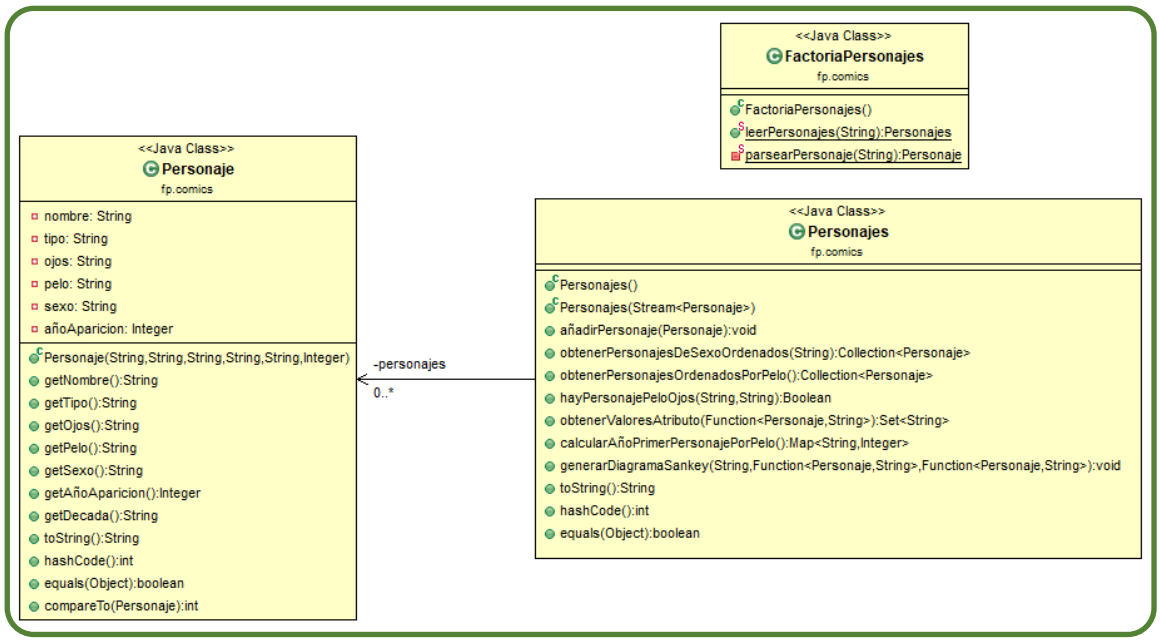
Datos de *personajes*

	A	B	C	D	E	F
1	name	align	eye	hair	sex	year
2	Batman	Good Characters	Blue Eyes	Black Hair	Male Characters	1939
3	Superman	Good Characters	Blue Eyes	Black Hair	Male Characters	1986
4	Green Lantern	Good Characters	Brown Eyes	Brown Hair	Male Characters	1959
5	James Gordon	Good Characters	Brown Eyes	White Hair	Male Characters	1987
6	Richard Grayson	Good Characters	Blue Eyes	Black Hair	Male Characters	1940
7	Wonder Woman	Good Characters	Blue Eyes	Black Hair	Female Characters	1941
8	Aquaman	Good Characters	Blue Eyes	Blond Hair	Male Characters	1941
9	Timothy Drake	Good Characters	Blue Eyes	Black Hair	Male Characters	1989
10	Dinah Laurel Lance	Good Characters	Blue Eyes	Blond Hair	Female Characters	1969
11	Flash	Good Characters	Blue Eyes	Blond Hair	Male Characters	1956

3 Modelo

En el siguiente diagrama se muestran todos los elementos que habrá que implementar en este proyecto. Todos ellos se incluirán en el paquete **fp.comics**. Los aspectos más destacables del modelo son:

- **Personaje**: clase para implementar el tipo básico.
- **Personajes**: tipo contenedor que incluye, además, algunos métodos de consulta basados en tratamientos secuenciales.
- **FactoriaPersonajes**: clase para dar soporte a la creación de objetos **Personaje** y **Personajes** a partir de datos en un fichero CSV.



Este diagrama ha sido generado con el plugin de Eclipse ObjectAid
URL de instalación: http://www.objectaid.com/update/current

4 Ejercicios

EJERCICIO 1

Crear la clase **Personaje** con los siguientes atributos

- **nombre:** atributo *String* con el nombre del personaje
- **tipo:** atributo *String* con el tipo del personaje (héroe, malvado, …)
- **ojos:** atributo *String* con el color de ojos del personaje
- **pelo:** atributo *String* con el color de pelo del personaje
- **sexo:** atributo *String* con el género del personaje
- **añoAparición:** atributo *Integer* con el año de la primera aparición del personaje en un comic. Este año debe ser igual o posterior a 1930

EJERCICIO 2

Crear los siguientes métodos de la clase **Personaje** comprobando las restricciones de los atributos en los casos en los que sea necesario

- **Personaje:** constructor de la clase a partir de los atributos en el orden que se indica en el ejercicio anterior
- Métodos *getters*: para todos los atributos de la clase
- **Personaje::getDecada:** propiedad derivada con década en la que apareció el personaje. Será un *String* con el primer año de la década (p.e. "1930" o "2000").
- **Personaje::toString:** mostrando todos los atributos
- **Personaje::equals:** usando el atributo **nombre** para determinar la igualdad
- **Personaje::hashCode:** usando la misma selección de atributos que el método **equals**
- **Personaje::compareTo:** para ordenar de forma natural por **añoAparicion**

EJERCICIO 3

Crear la clase **Personajes** con los siguientes atributos y métodos

- **personajes:** atributo con una lista de objetos **Personaje**

- **Personajes:** constructor vacío de la clase **Personajes**
- **Personajes:** constructor de la clase **Personajes** a partir de un *Stream* de **Personaje**
- **Personajes::añadirPersonaje:** método para añadir un **Personaje** a la lista de personajes
- **Personajes::toString:** mostrando todos los atributos
- **Personajes::equals:** usando el atributo **personajes** para determinar la igualdad
- **Personajes::hashCode:** usando la misma selección de atributos que el método **equals**

- **Personajes::obtenerPersonajesDeSexoOrdenados:** filtra la lista de personajes por sexo y ordena el resultado mediante el orden natural
- **Personajes::obtenerPersonajesOrdenadosPorPelo:** ordena la lista de personajes mediante el orden alfabético del color de pelo
- **Personajes::hayPersonajePeloOjos:** comprueba si hay una determinada combinación de color de ojos y color de pelo en algún personaje
- **Personajes::obtenerValoresAtributo:** dado un método *getter* (que devuelva un *String*), obtiene todos los valores distintos, encontrados en la lista de personajes, para el correspondiente atributo
- **Personajes::calcularAñoPrimerPersonajePorPelo:** calcula un diccionario, cuyas claves son los valores posibles de color de pelo, con el primer año en el que apareció un personaje con ese color de pelo

EJERCICIO 4

Crear la clase **FactoriaPersonajes** con los siguientes métodos estáticos

- **FactoriaPersonajes::parsearPersonaje:** método privado para construir un objeto **Personaje** a partir de una línea CSV del fichero de entrada
- **FactoriaPersonajes::leerPersonajes:** método que devuelve un objeto **Personajes** a partir de la ruta del fichero en el que se encuentran los datos de los personajes

EJERCICIO 5

Implementar el siguiente método de la clase **Personajes** que genera un informe gráfico usando la API de **Google Charts**

- **Personajes::generarDiagramaSankey:** genera un diagrama *Sankey* que refleje la relación entre los valores de dos atributos categóricos (los que devuelven *String*, salvo **nombre**). El método recibirá los siguientes parámetros:
  - ficheroSalida: un *String* con el nombre del fichero **HTML** en el que se generará la salida
  - Dos *getters* (de tipo *Function<Personaje,String>*), uno para cada uno de los atributos de la comparativa

  El proceso será el siguiente:
  - Construir el diccionario **atributos2Poratributo1** cuyas claves sean todos los posibles valores del "atributo 1", y los valores sean listas de los valores del "atributo 2" que han coincidido con la clave en algún personaje
  - Generar la salida **HTML** en un fichero de la carpeta **/out** con la siguiente instrucción:

    GraphTools.sankeyChart("out/"+ficheroSalida, atributos2Poratributo1);