UNIVERSIDAD DE GRANADA

Desarrollo de Software

Práctica 1.2

Granada, 24 de marzo 2021

Índice:

Raquel Molina Reche

Pablo Jiménez Jiménez

Parte Obligatoria

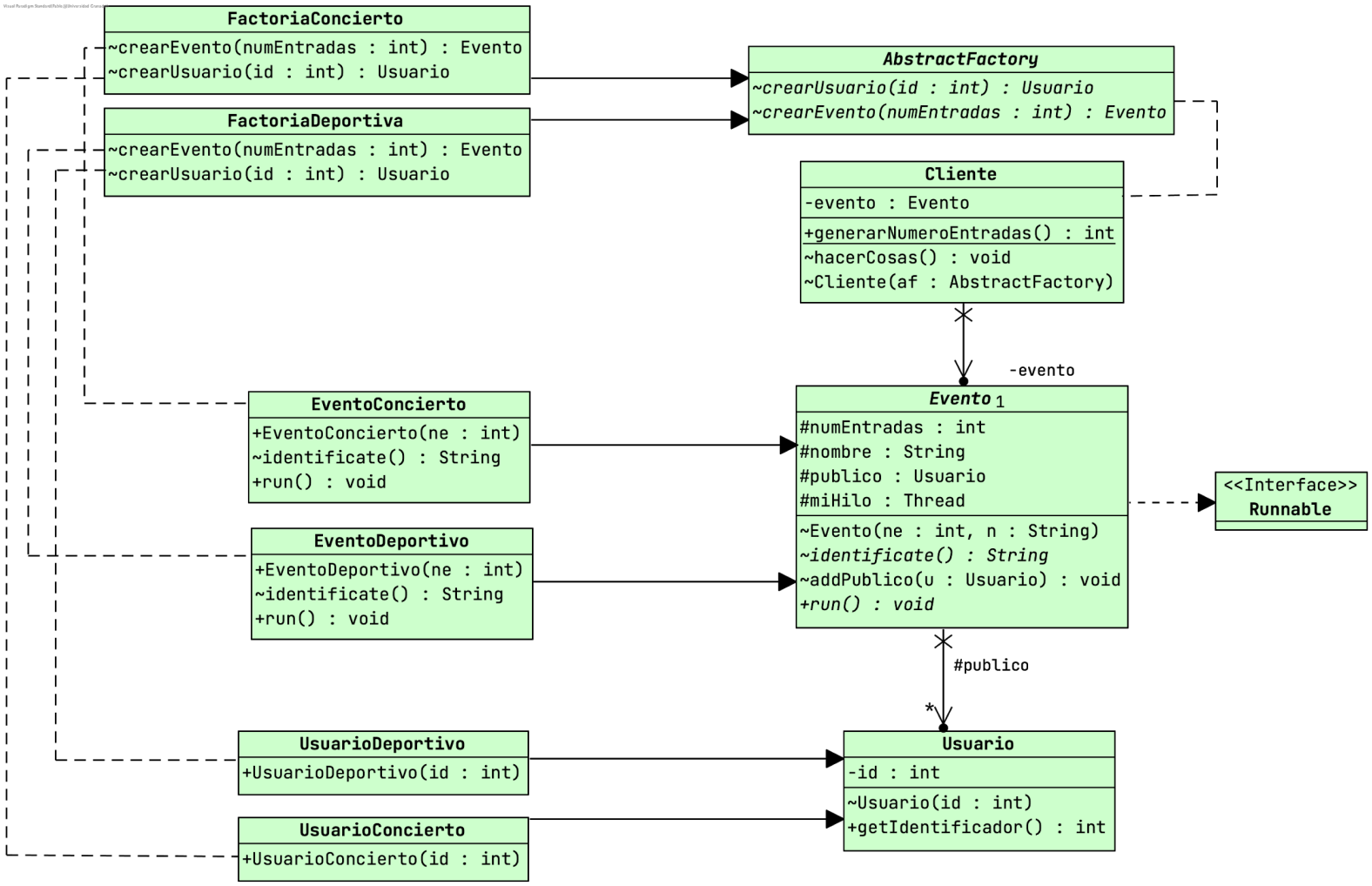
Para abordar este ejercicio hemos considerado dos familias, *Evento* y *Usuario* y estas familias tienen dos tipos de productos, productos de tipo *concierto* o productos de tipo *deportivo*.

El uso conjunto de los patrones de diseño *Factoría Abstracta* y *Método Factoría* aplicado en nuestro caso quedaría como se ve en el diagrama de clases asociado. De forma que son las factorías concretas *FactoriaDeportiva* y *FactoriaConcierto* las encargadas de crear los eventos o usuarios de los diferentes tipos cuando el cliente lo requiera. Al heredar de *AbstractFactory* , siendo esta una clase abstracta, se obliga a las diferentes factorías concretas a implementar sus métodos de forma que en el caso de necesitar más tipos de factorías concretas, por ejemplo porque el sistema crezca en funcionalidades, permite una alta adaptabilidad además de hacer el diseño mucho más intuitivo y una codificación mucho más llana.

De esta manera a la hora de crear un tipo de producto concreto, mediante el método factoría, el cliente solo necesita conocer un objeto del tipo factoría abstracta, por lo que es ajeno a los diferentes tipos de productos concretos que haya, resultando ser altamente escalable pues se pueden añadir más familias a las ya existentes y el cliente no se vería afectado por estos cambios.

En cuanto a la funcionalidad de nuestro programa, el cliente genera un evento al que se asignará un público, formado por usuarios, ambos del mismo tipo, a partir del número de entradas que el evento tenga. En la ejecución, serán hebras las que desempeñen el papel de eventos, de forma que los diferentes tipos de eventos transcurren de distinta manera o lo que es lo mismo cada uno puede llevar asociado un comportamiento diferente, pues estamos hablando de distintos tipos de eventos. En el transcurso de los eventos, las entradas serán vendidas, comenzará el evento y un porcentaje de asistentes irá abandonando los eventos según diferentes causas hasta que el evento finalice.

Diagrama de clases



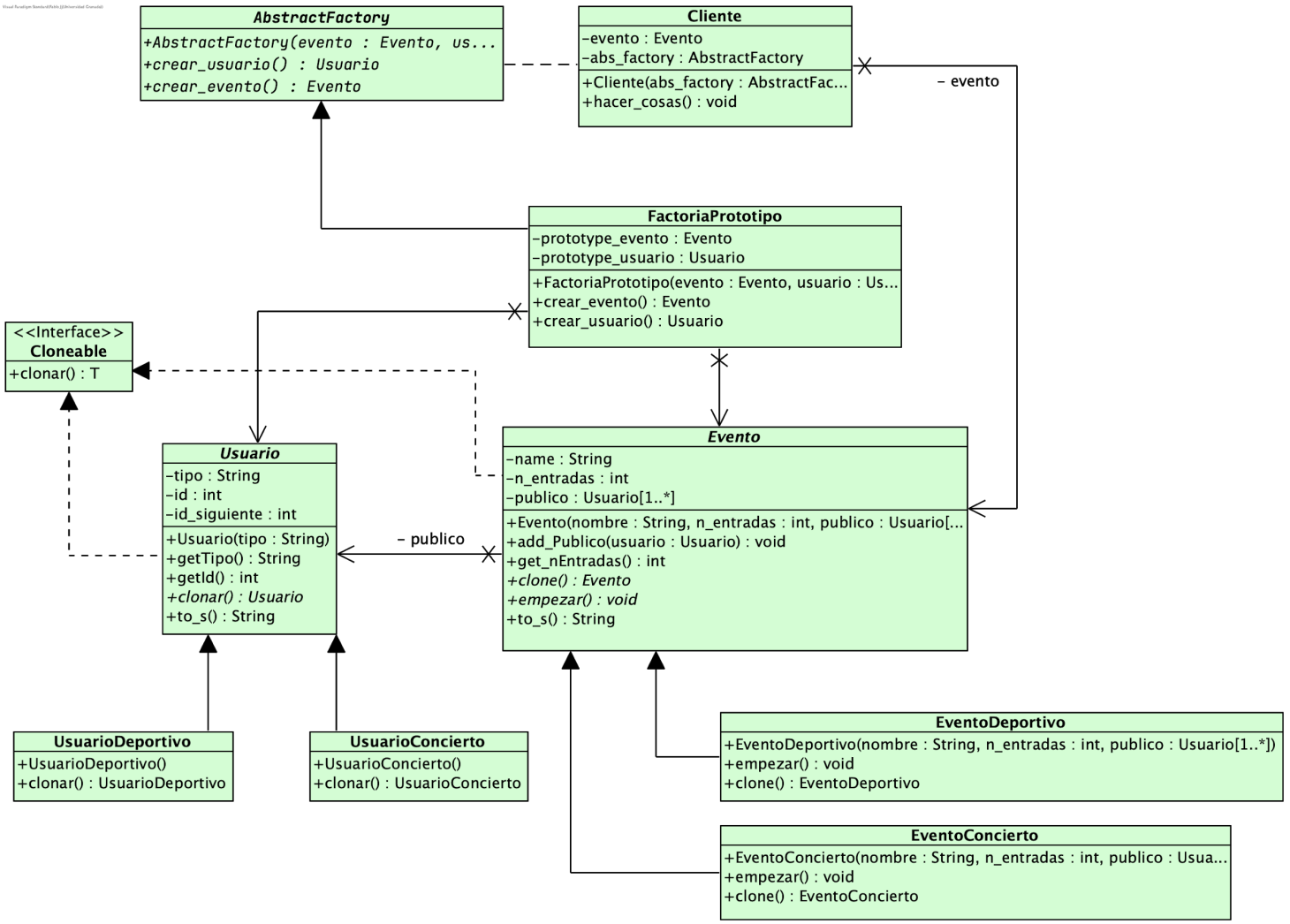
Parte Opcional

Hemos implementado la casuística anterior haciendo uso ahora del patrón creacional *Abstract Factory* con *Prototype*. En esta ocasión la factoría tiene prototipos que podrán ser de distintos tipos según cómo se cree el objeto de la factoría abstracta. Los objetos requeridos se clonan a partir de un estado base o inicial del objeto, aunque se podría discutir la necesidad de aplicación de este patrón en este ejemplo, es muy utilizado en los casos donde el coste de creación de un objeto es muy elevado (implica realizar consultas a bases de datos, inicializar grandes TDA's) por tanto aunque clonar un objeto no es una operación sencilla, es preferible a hacer una llamada al constructor.

Hemos diseñado una interfaz de forma que todos las futuras clases/modelos que quieran ser prototipos tengan que realizar la interfaz `Cloneable`. Otorgando

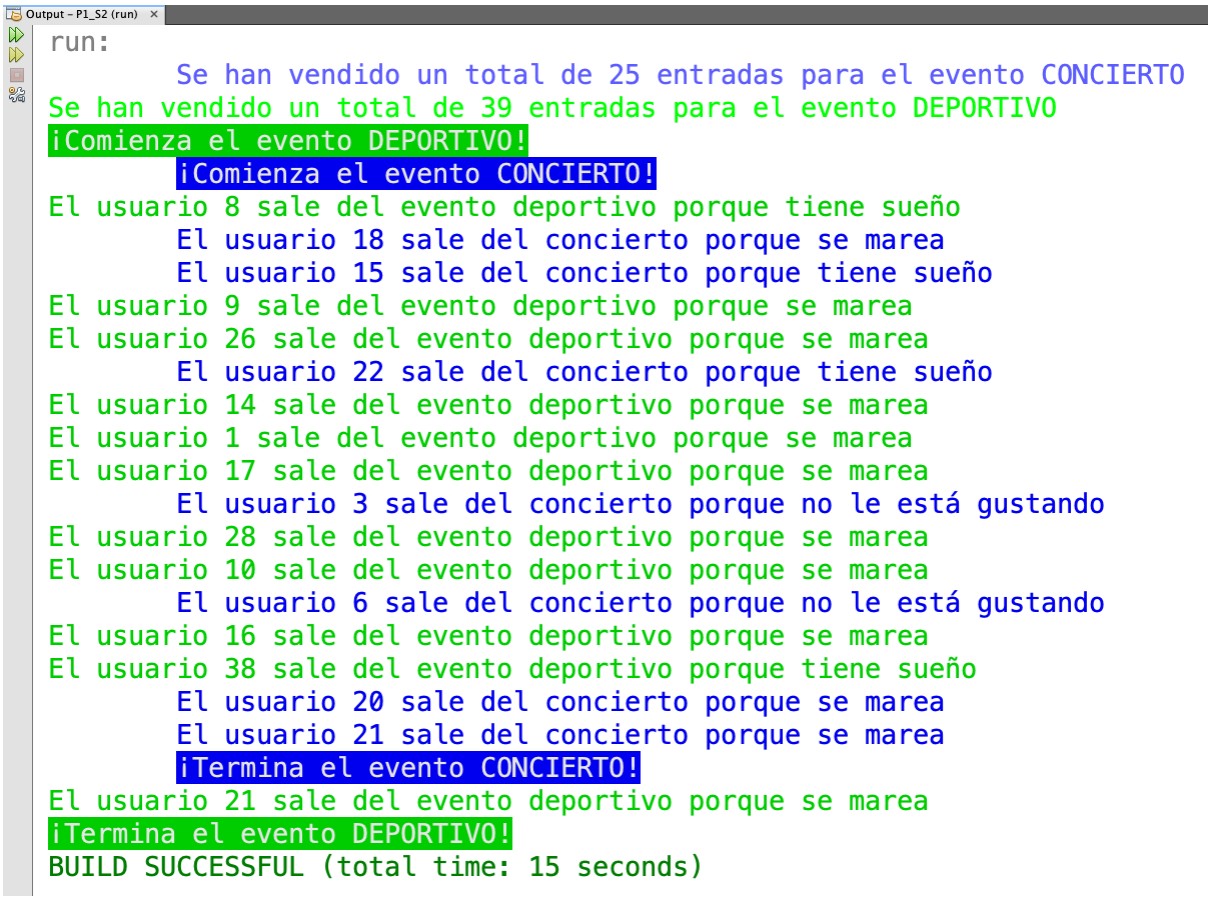
de está manera escalabilidad al ejemplo, para su implementación usando Ruby, hemos hecho uso del *mixin* para evitar que en una futura expansión pudiera haber problemas de herencia múltiple.

Diagrama de clases



Capturas de la aplicación

Parte obligatoria



Parte opcional