

Desarrollo de Software 20/21

Práctica 1

Uso de patrones de diseño en orientación a objetos

Víctor José Rubia López y Claudia Salado Méndez

23/03/21

1	SESIÓN 1ª: PATRÓN OBSERVADOR	2
1.1	DESCRIPCIÓN INFORMAL.....	2
1.1.1	<i>Requisitos obligatorios.....</i>	2
1.1.2	<i>Requisitos opcionales.....</i>	2
1.2	DIAGRAMA DE CLASES.....	4
1.2.1	<i>Antes de hacer el paso a código.....</i>	4
1.2.2	<i>Tras hacer el código volvemos a pasar a diagrama.....</i>	4
2	SESIÓN 2ª: EJERCICIO 1 P. FACTORÍA ABSTRACTA Y MÉTODO FACTORÍA	5
2.1	DIAGRAMA DE CLASES.....	5
2.2	BREVE DESCRIPCIÓN INFORMAL	5
3	SESIÓN 2ª: EJERCICIO 2 P. FACTORÍA ABSTRACTA CON P. PROTOTIPO	6
3.1	DIAGRAMA DE CLASES.....	6
3.2	DESCRIPCIÓN INFORMAL.....	6

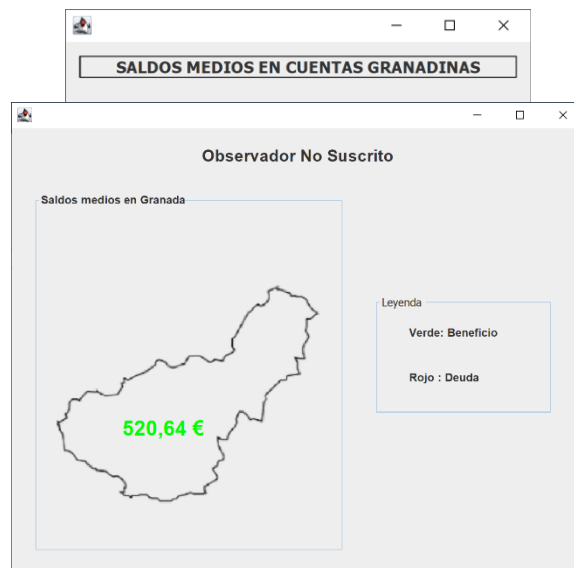
1 SESIÓN 1ª: PATRÓN OBSERVADOR

1.1 DESCRIPCIÓN INFORMAL

Nuestro tema escogido en prácticas es el diseño de un sistema de gestión para bancos. Por ello, hemos escogido para ilustrar bien el patrón observador, que sea el saldo de las distintas provincias donde se encuentra nuestro banco lo que observemos. Existe una hebra que cada cierto tiempo modifica el saldo de las distintas provincias.

1.1.1 Requisitos obligatorios

1. El primer observador definido es el suscrito convencional. Se trata de la clase *miGrafica*. Esta clase se comunica mediante *push* y el método *notifyObservers*. Hemos diseñado una gráfica que se complementará con el siguiente observador definido, indicando, mediante una barra de progreso a modo de gráfica, el saldo medio de la provincia.
2. El segundo observador definido es el no suscrito. Se trata de la clase *ObservadorNoSuscrito*. Esta clase se comunica mediante *pull*. Hemos diseñado un JFrame que muestra la provincia de Granada y el saldo medio de

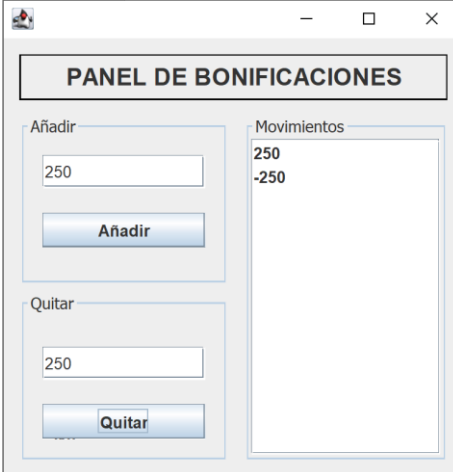


las cuentas de esta provincia mediante un texto centrado en la provincia y dos colores, verde para saldos positivos y rojo para negativos.

1.1.2 Requisitos opcionales

- El tercer observador definido es un observador suscrito que puede producir cambios en el modelo observado. La clase *BotonSaldo* se encargará de esto. Se trata de un JFrame en el que se muestran dos campos de texto con sus respectivos botones cada uno. En estos campos de texto el usuario puede introducir una cantidad y

posteriormente añadir o quitarle dicha cantidad al modelo observado (provincia de Granada). Mediante un JList se puede seguir cada vez que se le ha añadido o quitado saldo mediante este método.



Panel de Bonificaciones

Añadir

250

Añadir

Quitar

250

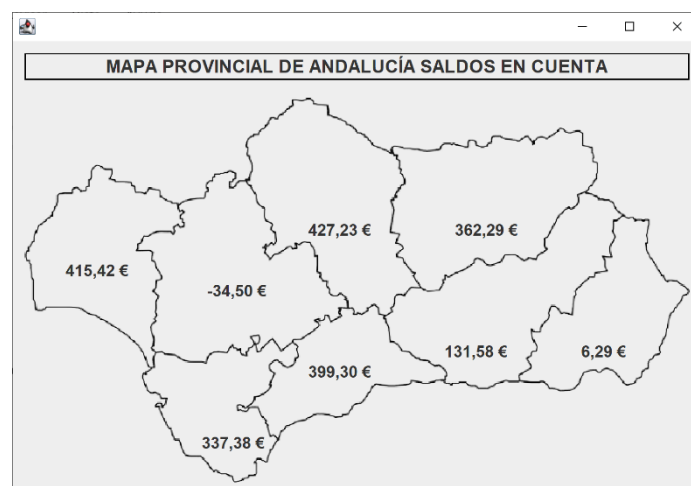
Quitar

Movimientos

250

-250

- El cuarto observador es la clase llamada *Mapa*. Para este, hemos usado un mapa en el que representamos en cada provincia de Andalucía los saldos medios en cuentas.

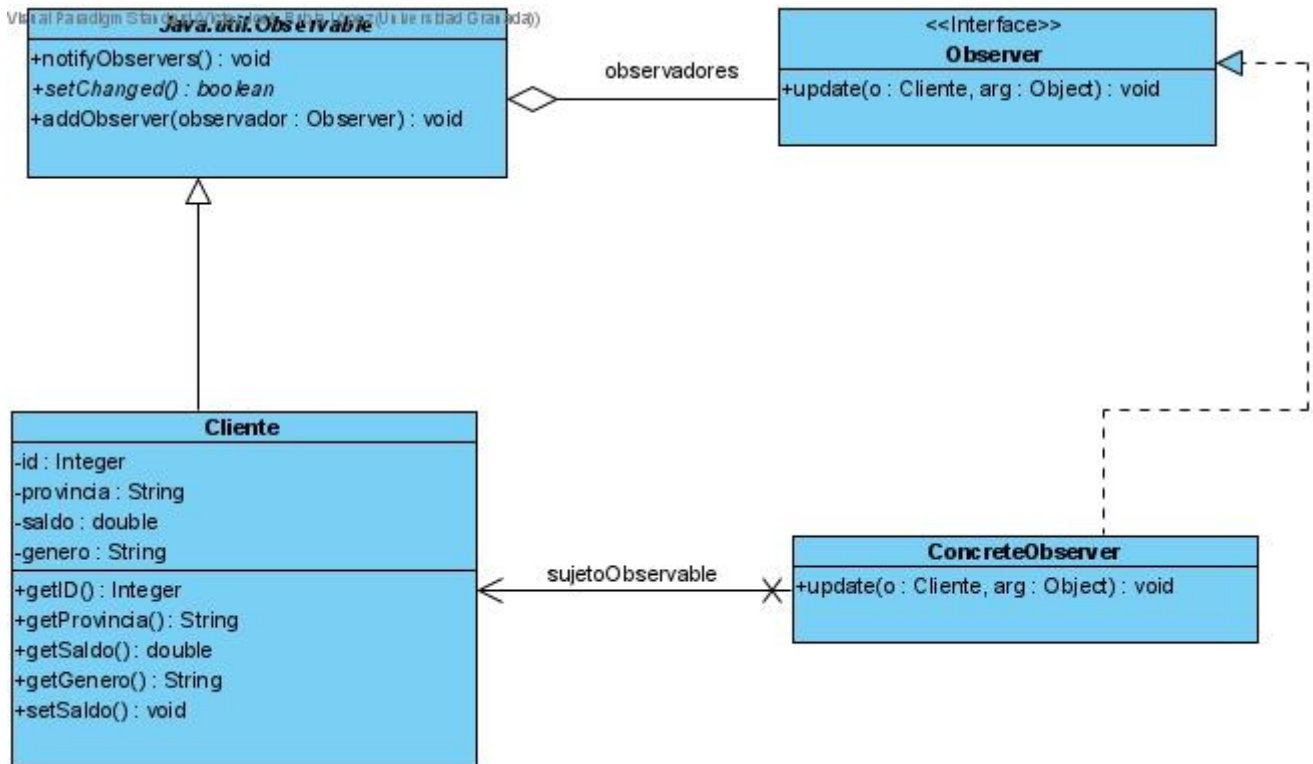


- El quinto observador es la clase *Compuesto* que resulta de combinar la clase *miGrafica* y *BotonSaldo*.

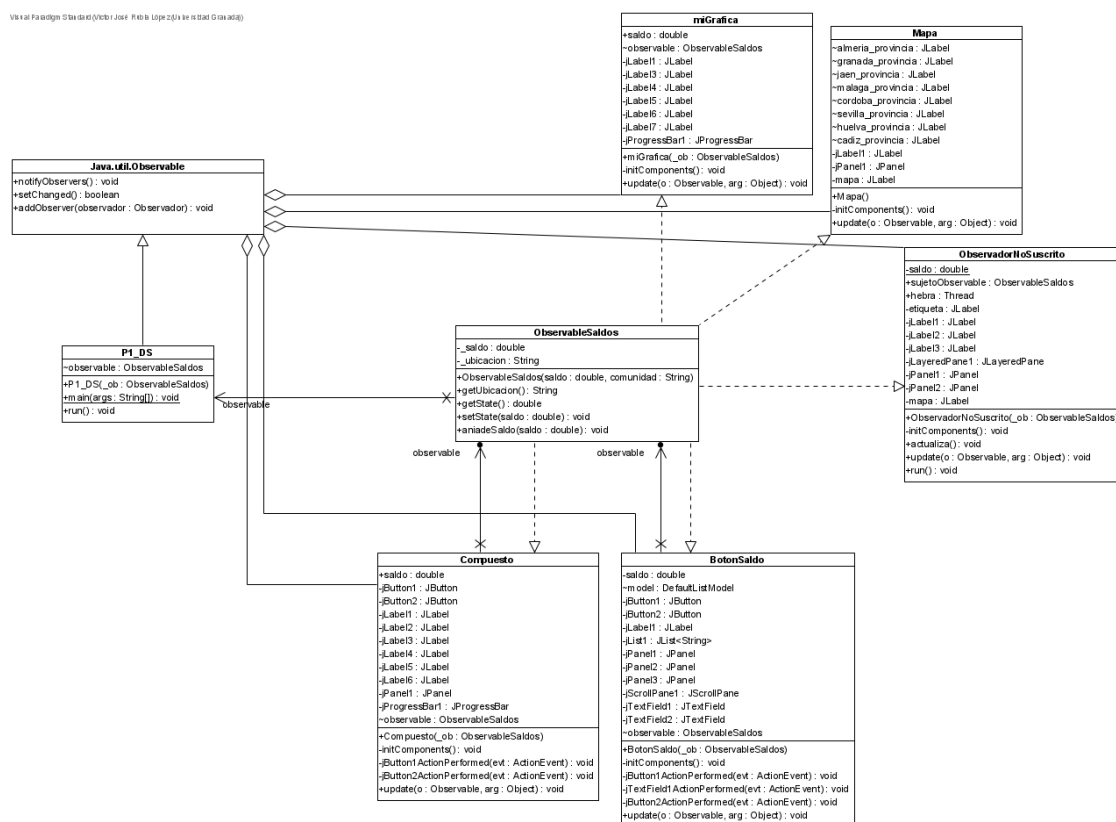


1.2 DIAGRAMA DE CLASES

1.2.1 Antes de hacer el paso a código

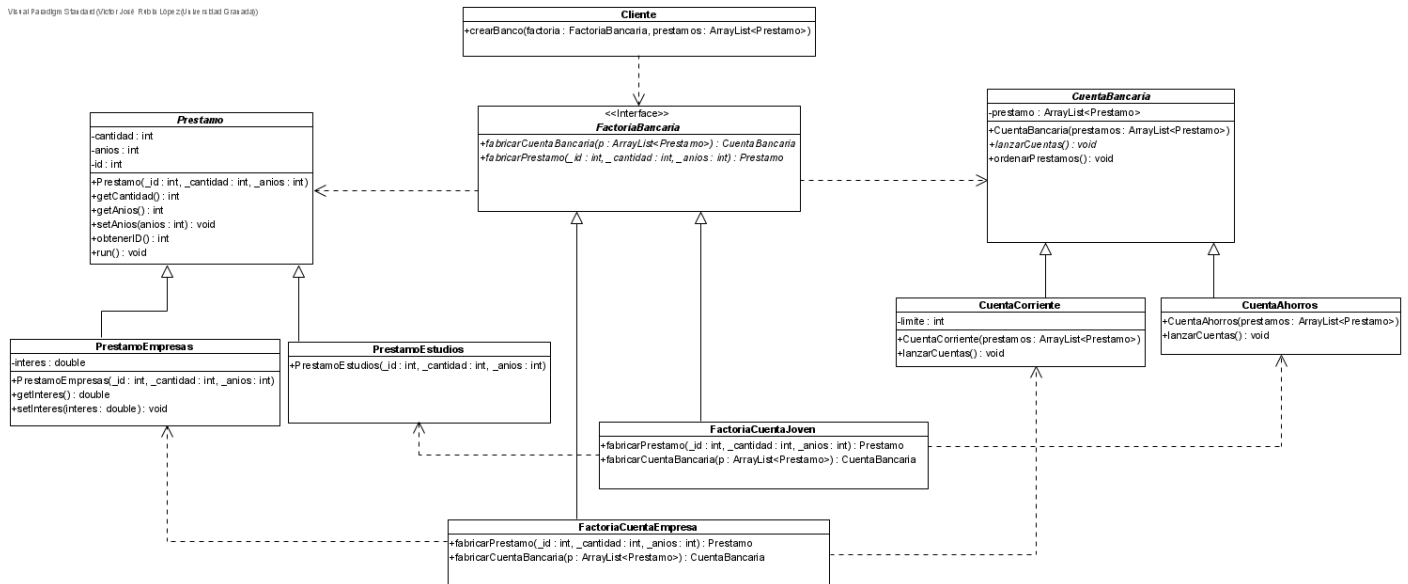


1.2.2 Tras hacer el código volvemos a pasar a diagrama



2 SESIÓN 2ª: EJERCICIO 1 P. FACTORÍA ABSTRACTA Y MÉTODO FACTORÍA

2.1 DIAGRAMA DE CLASES



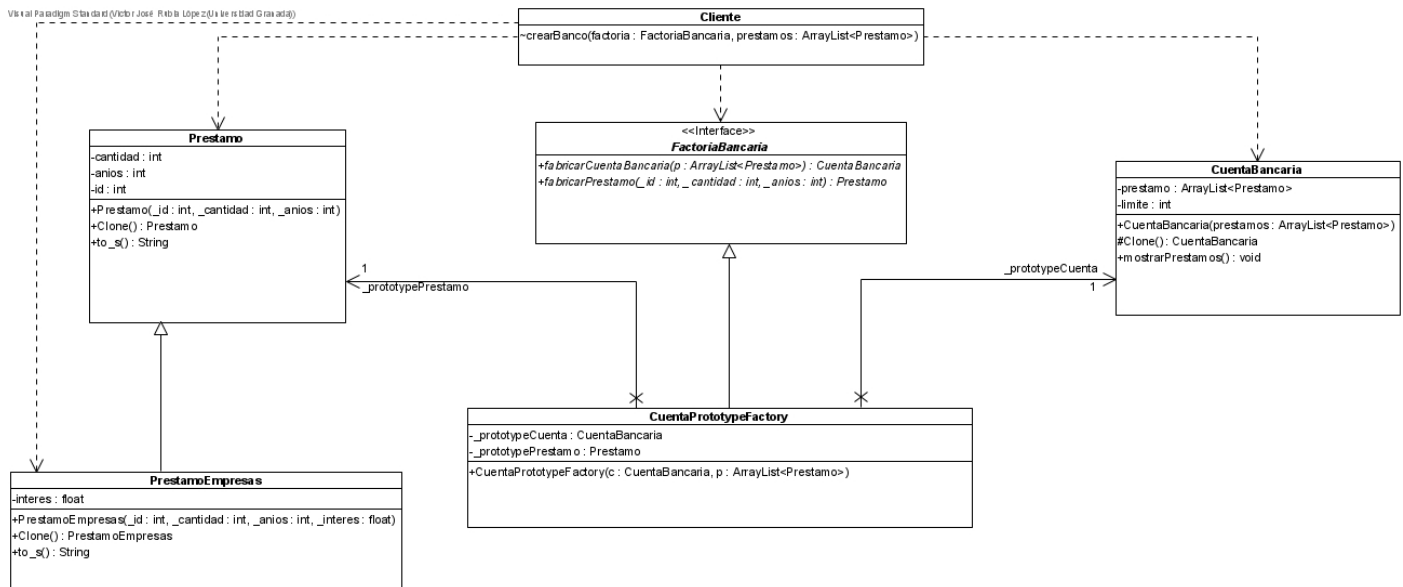
2.2 BREVE DESCRIPCIÓN INFORMAL

El programa realiza una simulación de, a medida que vayan pasando los años (hecho por una hebra) se van venciendo los préstamos. Tenemos tanto préstamos de Estudios, como préstamos de empresas (que poseen además un interés).

```
[P. EMPRESAS PENDIENTES] en el año 2025
Prestamo 1 quedan 0 años.
Prestamo 3 quedan 0 años.
Prestamo 0 quedan 5 años.
Prestamo 2 quedan 10 años.
Prestamo 4 quedan 11 años.
[P. EMPRESAS] Su prestamo con id 3 y cantidad 7294 ha vencido su plazo.
[P. EMPRESAS] Su prestamo con id 1 y cantidad 1913 ha vencido su plazo.
```

3 SESIÓN 2ª: EJERCICIO 2 P. FACTORÍA ABSTRACTA CON P. PROTOTIPO

3.1 DIAGRAMA DE CLASES



3.2 DESCRIPCIÓN INFORMAL

Hemos entendido el ejemplo propuesto de Maze en teoría y hemos aplicado estos dos patrones en nuestro sistema de la forma que mejor hemos entendido. Como no nos queda claro del todo si lo hemos implementado adecuadamente, usaremos la clase de la última semana para darle el toque final preguntándole a la profesora en horario de tutoría o en clase.

El funcionamiento de la aplicación es bastante básico y lo hemos hecho así para ejemplificar y entender el uso del patrón Prototipo y el Patrón Factoría Abstracta de una forma sencilla. La clase **CuentaPrototypeFactory** se encargará de crear Cuentas y Prestamos según el prototipo, mediante clonación.

Por tanto, en el método **crearBanco** tenemos una factoría de prototipos “Simples” en el que tenemos un prototipo de **CuentaBancaria** que posee un solo préstamo a 3 años y de 1500€. Por otro lado tenemos que el préstamo prototipo “Simple” sean de 1000€ y a 3 años.

Por otro lado, tenemos una factoría de prototipos de “Empresas”. Estos crean prototipos de **CuentaBancaria** con un préstamo de 10.000€ a 10 años y al 10% de interés y prototipos de **PrestamoEmpresas** de 5000€ a 5 años y al 5% de interés.