Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 1
Primer cuatrimestre de 2020

Alumno:	LOCATELLI, Santiago
Número de padrón:	104107
Email:	slocatelli@fi.uba.ar

Índice

1. Introducción **2**

2. Supuestos **2**

3. Diagramas de clase **3**

4. Detalles de implementación **3**
4.1. Clase Pintor, y clases hijas. 3
4.2. El Descuento para Pintores de Pincel. 4
4.3. Delegación. 4

5. Excepciones **4**

6. Diagramas de secuencia **5**

1. Introducción

El presente informe reune la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación, de lo que asumo que es una pintureria, en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Primero: Asumo que las horas por metro cuadrado son todas iguales para todos los pintores del mismo tipo, es decir, que todos los pintores de pincel tarden dos horas por metro cuadrado y los pintores de rodillo una hora. Lo interprete así, ya que en ningun momento especifica que dos pintores de pincel puedan tardar diferentes tiempos en pintar un metro cuadrado.

Segundo: Según las pruebas, asumo que los metros cuadrados no son un atributo de alguna clase, ya que por lo constructores que proponen no pertenece a alguna clase. Por esto, yo note que la cantidad de metros cuadrados va dando vuelva por todos los métodos, la verdad que eso me hizo ruido, pero lo deje así.

Tercerco: Mas que un supuesto, pareciera ser planteado así por el enunciado del TP (los tests), pero no me quedo otra que Presupuesto solo sea un contenedor de datos, lo único que pude/ocurrió fue hacer un método de clase como constructor del mismo, a parte de sus getters y setters.

Cuarto: En el caso de haya dos pintores con el mismo valor por hora, y estos sean los que proponen un presupuesto mas barato, se tomara al primer pintor de la lista.

3. Diagramas de clase

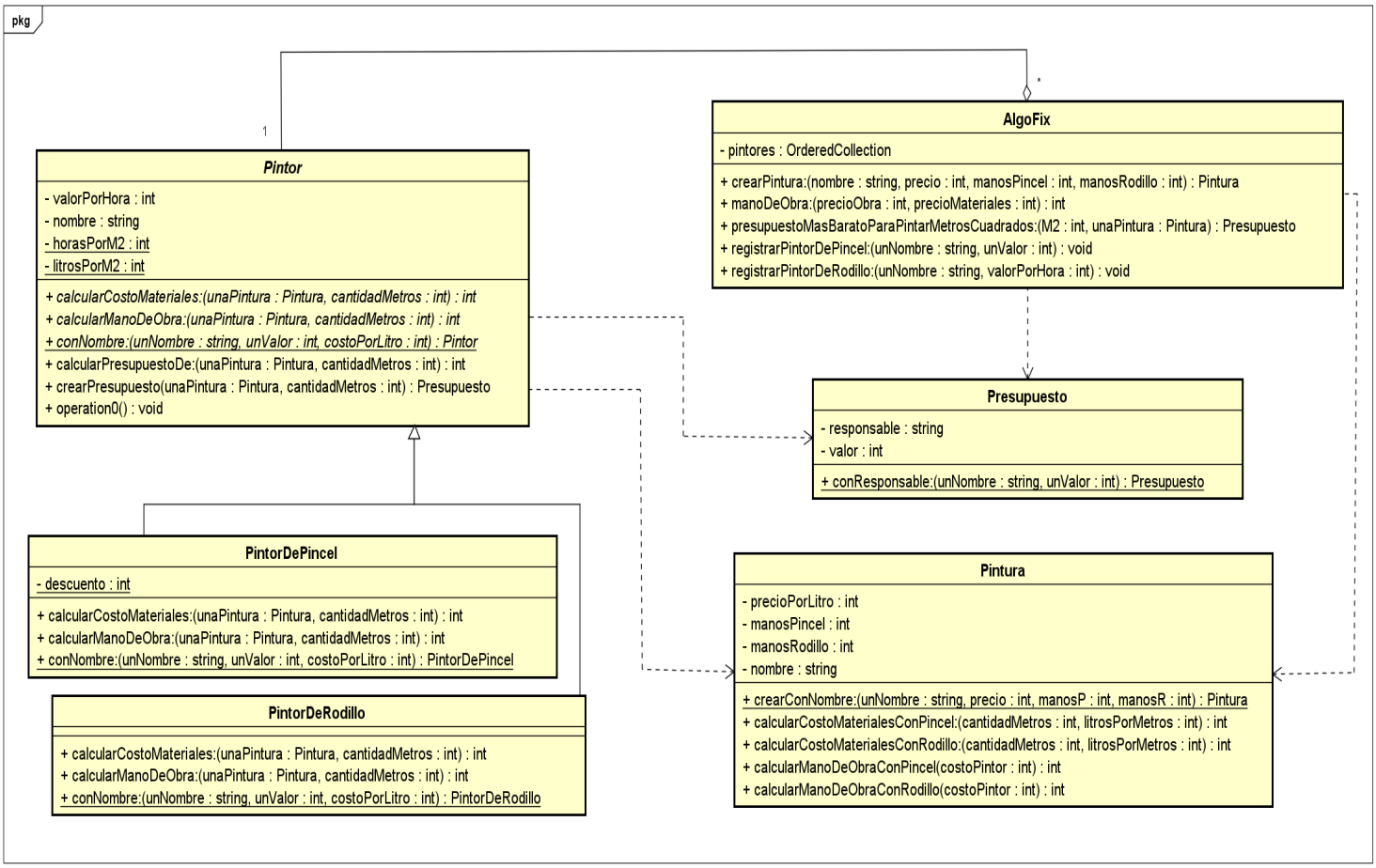

Figura 1: Diagrama de clases.

4. Detalles de implementación

4.1. Clase Pintor, y clases hijas.

Con respecto a la clase Pintor, considere que era necesaria la implementación de dos clases hijas, estas clases son, PintorDeRodillo y PintorDePincel. Estas clases hijas cumplen con la condición "es un", se ve claramente que un PintorDePincel o de Rodillo, es un Pintor, en este caso se ve fácilmente la relación de herancia, también quiero mencionar que hay métodos que redefini y otros que no. Los métodos que tuve que redefinir son:

`-calcularCostoMaterialesDe: deMetros:`
`-calcularManoDeObraDe: deMetrosCuadrados:`

Y hubo otros métodos que no necesitaron redefinición, ya que para ambas clases hijas la implementación era la misma, es por eso que estas clases hijas utilizan el método de la clase padre. Estos métodos son:

```
-calcularPresupuestoDe: deMetrosCuadrados:
-crearPresupuestoDe: deMetrosCuadrados:
```

4.2. El Descuento para Pintores de Pincel.

Al descuento para pintores de pincel, lo pensé como una variable de clase, ya que para todos los pintores de este tipo el descuento es el mismo. Si se llega al caso en el que cada pintor hace un descuento diferente, ahí ya no podría ser una variable de clase, pero, así como esta el enunciado me pareció adecuado.

4.3. Delegación.

Busque el no utilizar ningún getter para lo que es la implementación de métodos de las clases, a mi parecer con éxito. Solo los utilizo para los tests correspondientes a cada clase. Esto lo hago delegando responsabilidades a las distintas clases según el caso dado. Por ejemplo, a la clase Pintura le delego el calculo de los costos de los materiales.

5. Excepciones

NumeroIvalidoError El objetivo de esta excepción es, como bien describe su nombre, marcar que hubo un error a la hora de pasar un parámetro entero o decimal, según a criterio de cada uno, a un método. Un ejemplo de numero invalido puede ser cuando se crea un pintor sea de rodillo o de pincel, si el valor por hora pasado por parámetro es un valor negativo, esta claro que esto no puede ser así.

NoTieneElementosError Esta excepción sirve para el caso en el que un array o una lista, cualquier estructura que pueda almacenar una cadena de datos, no tenga ningún elemento guardado, por lo que si se quiere conseguir un elemento de esta no será posible. En el TP1, en el único caso que nos sirve esta excepción es con la OrderedCollection con la que implemente la clase AlgoFix.

NombreInvalidoError También pensé en agregar una excepción de nombre invalido, pero al pensar en los casos en los que se puede llegar a lanzar la excepción me parecieron bastantes, a mi por lo menos, todos los casos en los que el nombre tenga algún número podrían ser tranquilamente un nombre invalido, o que no tenga nada también puede ser otro caso, pero me termino pareciendo muy rebuscado y por eso no lo implemente.

6. Diagramas de secuencia

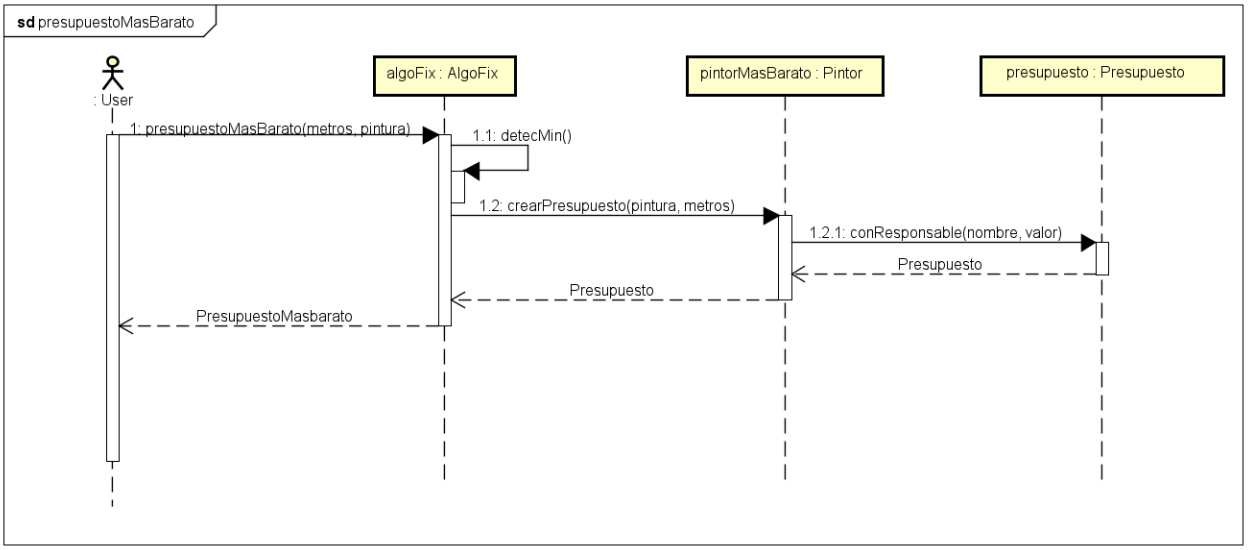

Figura 2: Presupuesto más barato.

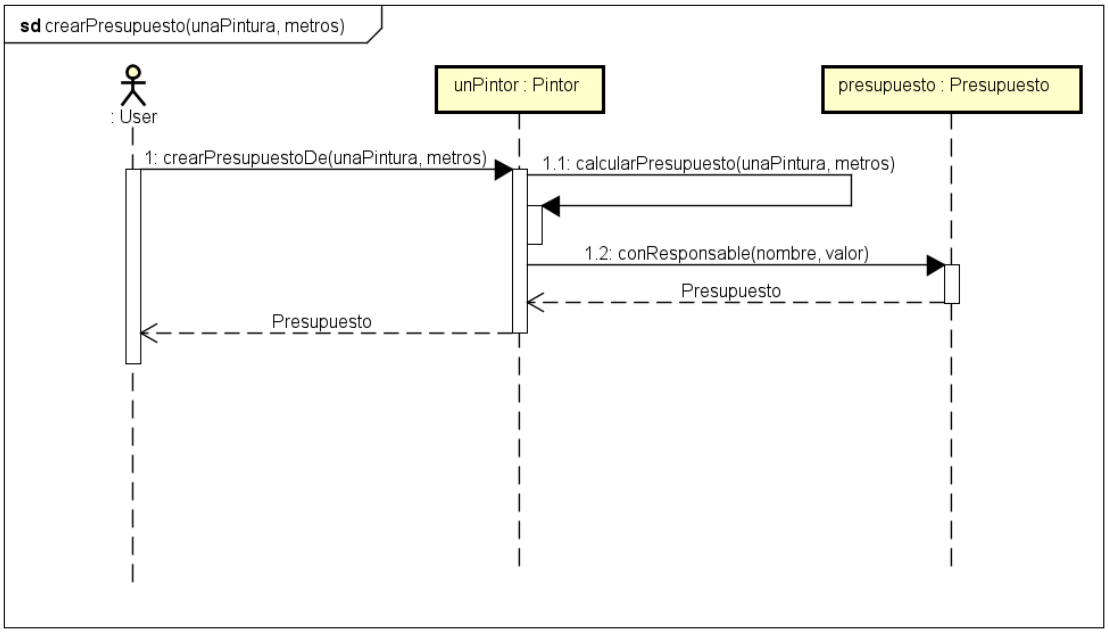

Figura 3: Crear presupuesto.

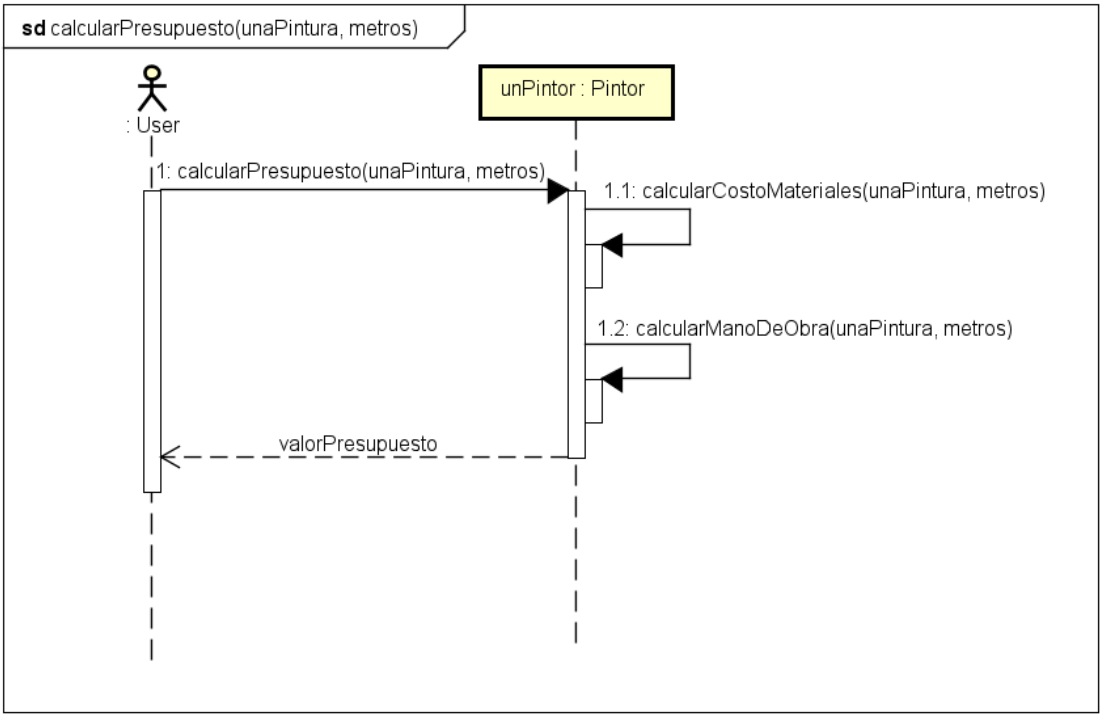

Figura 4: Calcular presupuesto.