

Trabajo Práctico 1 — Smalltalk

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

Alumno:	KOVNAT, Thiago
Número de padrón:	104429
Email:	tkovnat@fi.uba.ar

Índice

1. Introducción	2
2. Supuestos	2
3. Diagramas de clase	2
4. Detalles de implementación	2
4.1. AlgoFix	2
4.2. Pintor	3
4.3. Pintura	3
5. Excepciones	3
6. Diagramas de secuencia	4

1. Introducción

El presente informe reúne la documentación de la solución del primer trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un sistema de gestión de costos de una pinturería en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Durante la implementación, la principal duda que surgió fue que hacer en caso de que haya dos o mas pintores que presenten el mismo presupuesto mínimo. Debido a la forma en la que esta implementado el método que devuelve dicho presupuesto, la resolución de esta duda quedó sujeta a la implementación propia del método *detectMin* por lo que se devuelve el primer pintor que haya presentado dicho presupuesto mínimo. También decidí que un pintor no es identificado unívocamente por su nombre, es decir, se puede registrar mas de un pintor con un mismo nombre y costos por hora distintos. La otra duda que surgió fue que hacer en caso de que no haya un pintor registrado a la hora de pedir un presupuesto mínimo. Para dicho caso, se creó una excepción que esta explicada en su debida sección.

3. Diagramas de clase

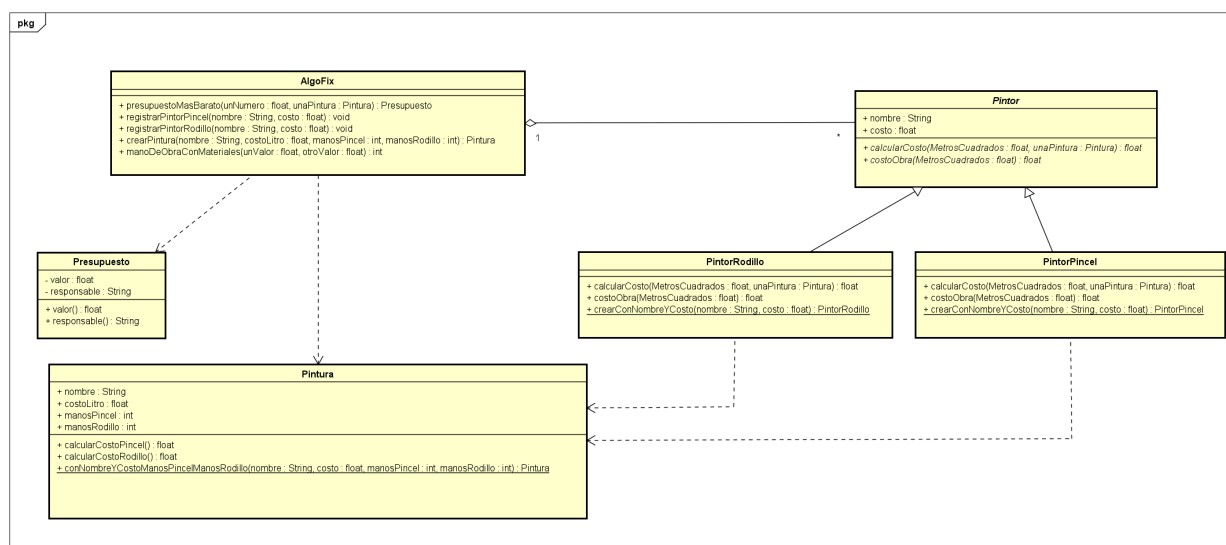


Figura 1: Diagrama de Clases de la solución planteada.

4. Detalles de implementación

4.1. AlgoFix

El método principal del trabajo práctico era la implementación del método *presupuestoMasBarato*. En mi implementación, los pintores que se hayan creado en *AlgoFix* se guardaban en una misma *OrderedCollection*, indistintamente de que tipo de pintores este registrando. Lo que me permite esto es que yo en un futuro puedo crear cualquier cantidad de tipo de pintores distintos sin tener

que modificar el funcionamiento de la clase `AlgoFix`, ya que la misma trabaja bajo la única suposición de que el pintor registrado entienda el mensaje `calcularCostoConM2yPintura`. Dentro del método `presupuestoMasBarato`, `AlgoFix` hace uso del método `detectMin` de `OrderedCollection` y delega el cálculo del presupuesto a cada pintor y este a su vez delega una parte del cálculo a la instancia de la clase `Pintura`.

En cuanto a la creación de pintores, cuando se le pide a `AlgoFix` registrar un pintor este crea una instancia del tipo de pintor que se haya solicitado registrar y se lo guarda en la colección de pintores. Para la pintura, cuando se le pide a `AlgoFix` crear una pintura este simplemente la instancia con los valores solicitados y la devuelve, ya que no es necesario guardar la pintura dentro de la instancia de `AlgoFix` debido a que, cuando se le pide el presupuesto, se le envía la instancia de la clase `pintura` con la cual quiero calcular dicho presupuesto.

4.2. Pintor

Decidí tener una clase abstracta llamada `Pintor`, con dos métodos abstractos que cada clase hija debiera implementar. Dichos métodos son los pilares fundamentales del cálculo del presupuesto, y dado que cada tipo de pintor lo calcula de una forma diferente (Ya que puede aplicar descuentos u en un futuro otros beneficios), me resultó lógico tener una clase abstracta y que cada tipo distinto de pintor herede de esta clase e implemente los métodos que calculan el presupuesto. Debido a que en un futuro puedo decidir implementar muchos otros tipos de pintores, decidí usar herencia ya que permite la reutilización de código y además cada clase hija cumple con la relación *es un* con la clase padre.

Para el cálculo del presupuesto, cada clase hija de `Pintor` tiene implementado el método `calcularCostoConM2 yPintura`. Dentro de dicho método, el `Pintor` hace una simple cuenta: $(M2 * cantidadManos) * (costoHoraPintor * HoraPorCadaM2 + costoLitroPintura * cantidadLitroPorM2)$ La cantidad de manos la consigue bajo un método `getter` de la clase `Pintura`, mientras que el costo operativo por parte del pintor se calcula en un método llamado `costoObra` y el cálculo del costo de la pintura se le es delegado a la instancia de la clase `Pintura`.

4.3. Pintura

La clase `Pintura` tiene como única responsabilidad el cálculo de el costo de pintar con cada tipo de pintado. Es decir, la pintura sabe cuantos litros consume cada tipo de pintado y cuando se le pide calcular el costo con cierto tipo lo que hace es devolver el costo por litro multiplicado por la cantidad de litros que requiere ese tipo de pintado. Ese valor representa el costo total de la pintura por cada M2 que uno quiera pintar. Por como esta implementado la solución general del trabajo práctico, si en un futuro se quisiera implementar mas tipos de pintado, se debería agregar un método en la clase `pintura` que calcule dicho costo.

5. Excepciones

Excepción `NumeroInvalidoError`: excepción creada para cuando se inserta un número inválido para cierto parámetro, por ejemplo: cantidad de metros cuadrados.

Excepción `SinPintoresDisponiblesError`: excepción creada para cuando se pide el cálculo de un presupuesto mínimo pero no hay ningún pintor registrado actualmente.

6. Diagramas de secuencia

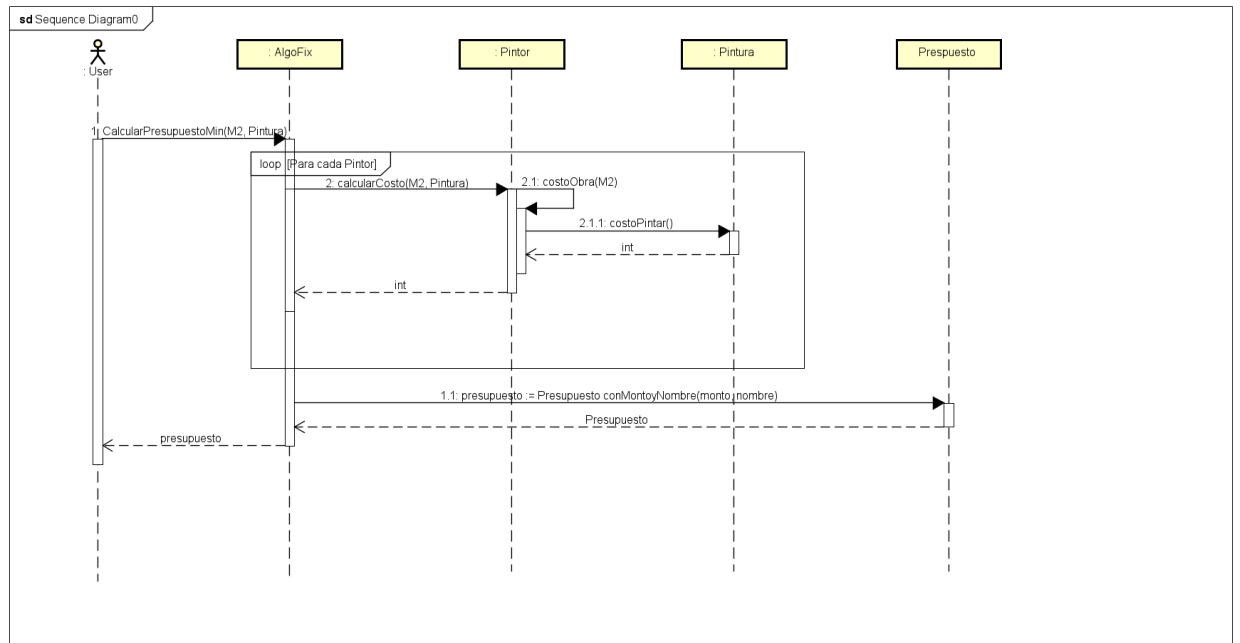


Figura 2: Calculo del presupuesto mínimo.

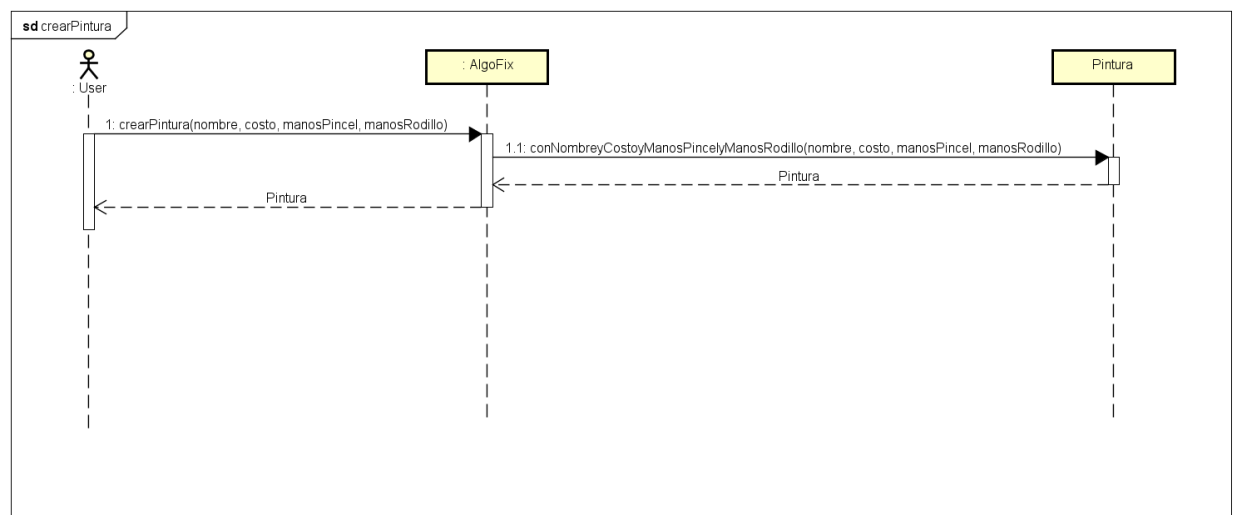


Figura 3: Creación de una pintura.

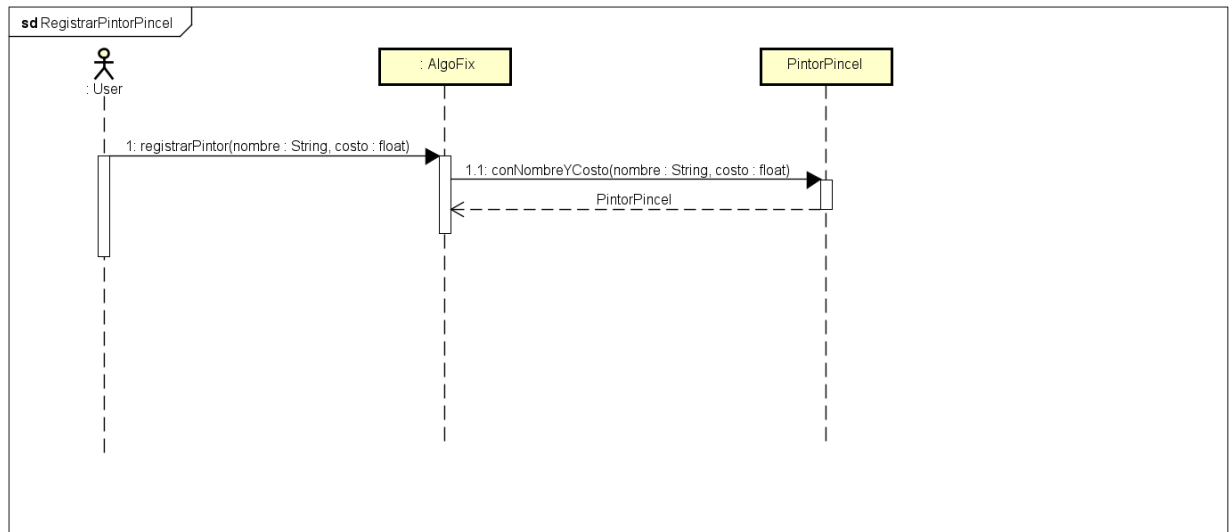


Figura 4: Registro de un pintor de pincel.

En caso de querer registrar un pintor de rodillo, el diagrama sería exactamente igual salvo que se instanciaría un pintor de tipo rodillo.