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

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 un sistema de gestión de costos de una pintureria 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 *d*etectMin por lo que se devuelve el primer pintor que haya presentado dicho presupuesto mínimo. Tambien 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

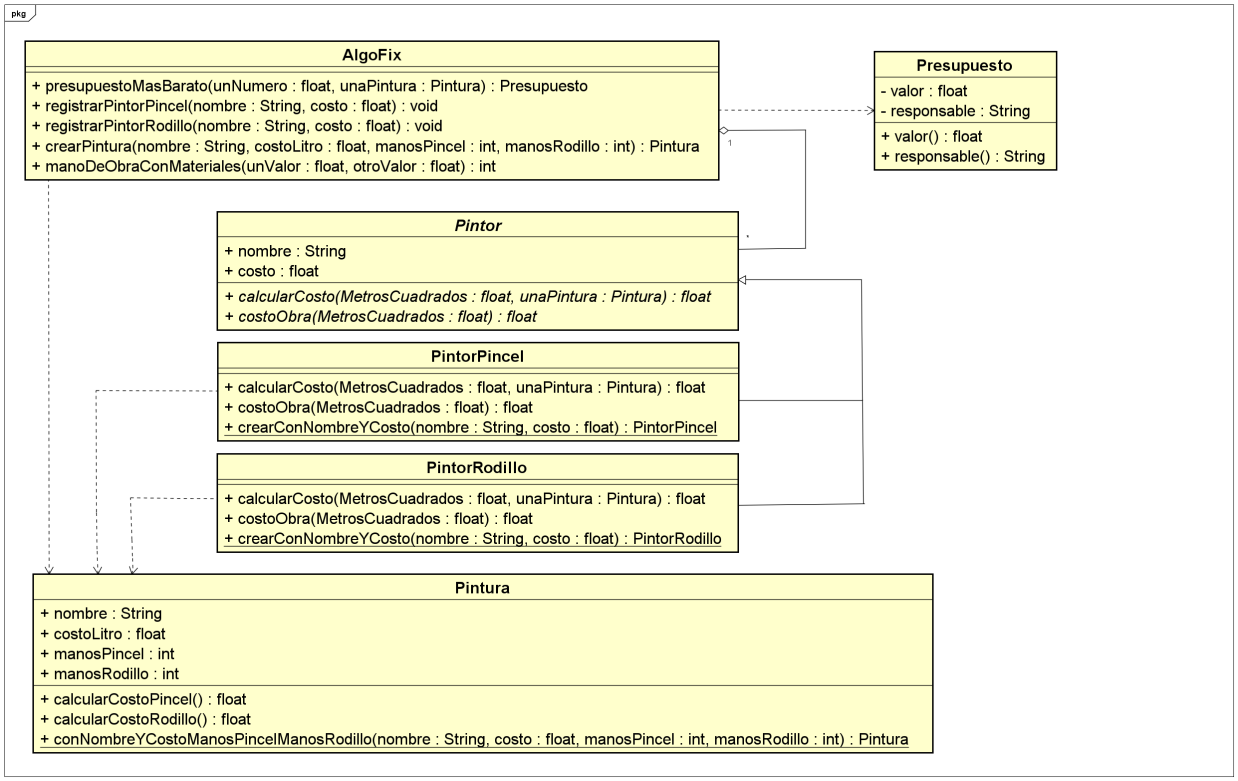

Figura 1: Diagrama de Clases de la solución planteada.

4. Detalles de implementación

4.1. AlgoFix

El método principal del trabajo práctico era la implementación del método *p*resupuestoMasBarato. 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 unica 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. Cálculo del Presupuesto

Llamando M2 a la cantidad de metros cuadrados a pintar, costoHora al costo por hora del pintor, Manos a la cantidad de manos para pintar, costoLitro al costo por litro de la pintura, cantHoras a la cantidad de horas necesarias para pintar un metro cuadrado y litroM2 a la cantidad de litros necesarios para pintar un metro cuadrado, sabemos que el presupuesto se calcula de la siguiente manera:

(M2 . costoHora . Manos . cantHoras) + (M2 . costoLitro . litroM2 . cantManos)

Sacando factor común, tenemos que el presupuesto se puede calcular como:

(M2 . Manos) . [(costoHora . cantHoras) + (costoLitro . litroM2)]

Finalmente, llamando costoObra a (costoHora . cantHoras) y costoPintar a (costoLitro . litroPorM2), tenemos que el presupuesto se calcula como:

(M2 . cantManos) . (costoObra + costoPintar)

costoObra y costoPintar representan los costos del pintor y de la pintura para pintar un único metro cuadrado.

4.3. Pintor

Decidí tener una clase abstracta llamada Pintor, con dos métodos abstractos que cada clase hija debera 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 *e*s un con la clase padre.

Para el cálculo del presupuesto, el Pintor se encarga de calcular el factor común de la operación mostrada en **4.2 Cálculo del Presupuesto** y del costoObra, mientras que delega el calculo del costoPintar a la Pintura.

4.4. Pintura

La clase Pintura tiene como única responsabilidad el calculo del 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 implementada 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 invalido 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 ningun pintor registrado actualmente.

Excepción NombreInvalidoError: Excepción creada para cuando se intenta registrar a un pintor o a una pintura sin ningun nombre, es decir, un string vacío.

5.1. Responsabilidades en la comprobación de excepciones

Hay ciertos casos en los cuales saber a quien le pertenece la responsabilidad de comprobar los datos que podrían derivar en una excepción son bastante claros, por ejemplo: la comprobación de los datos con los cuales uno instancia un pintor o una pintura le corresponde a dicha clase. Es decir, únicamente el Pintor debería saber si acepta un costo por hora negativo. De la misma manera, solo una Pintura debería saber si acepta costos o cantidad de manos negativos. Si por ejemplo yo le diera esta responsabilidad a AlgoFix, esto implicaría que AlgoFix sabe como esta implementada la clase Pintor y que valores acepta o no, por lo cual estaría rompiendo el encapsulamiento. Otro ejemplo de un caso simple es la responsabilidad de comprobar si hay o no pintores disponibles a la hora de calcular un presupuesto. Es claro que esta responsabilidad le corresponde a AlgoFix ya que es el quien contiene a los pintores disponibles y sabe si hay un pintor disponible o no.

Hay otros casos en donde esta responsabilidad no es tan clara y puede ser un poco mas subjetiva. Por ejemplo, ¿A quién le correspondería la comprobación de que la cantidad de metros cuadrados que quiero pintar no sea negativa? En este caso, yo decidí que sea el pintor quien lleve esta responsabilidad, ya que representa a una persona quien sabe que los metros cuadrados es una magnitud física que no puede ser negativa.

6. Diagramas de secuencia

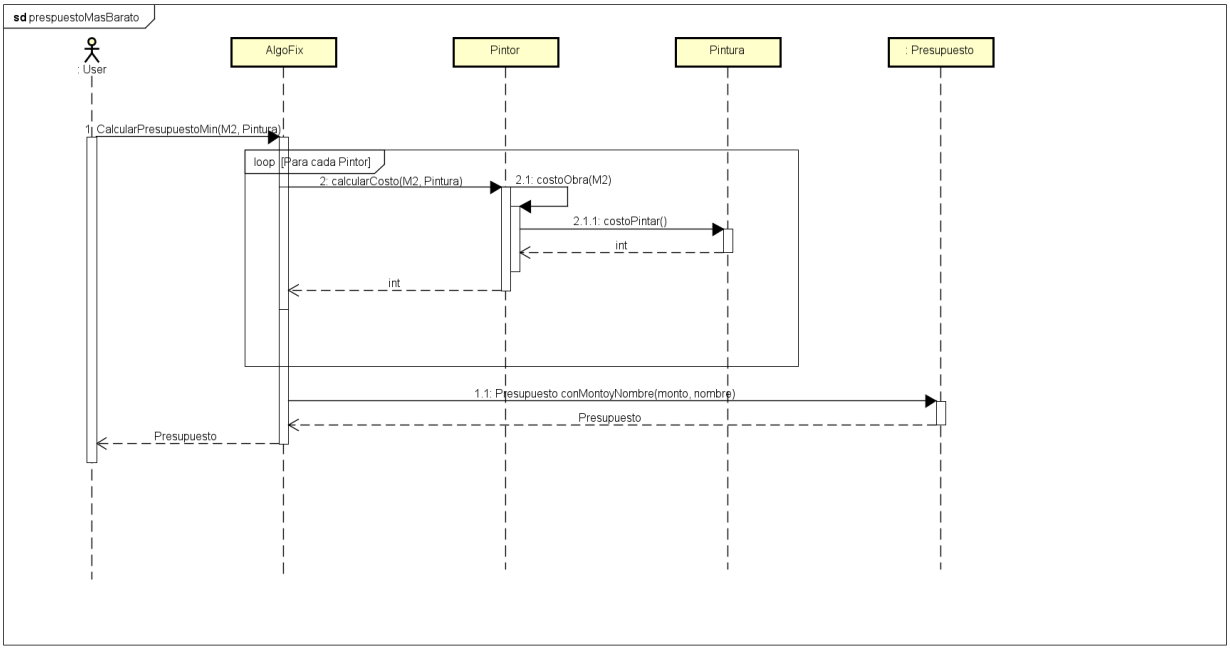

Figura 2: Calculo del presupuesto mínimo.

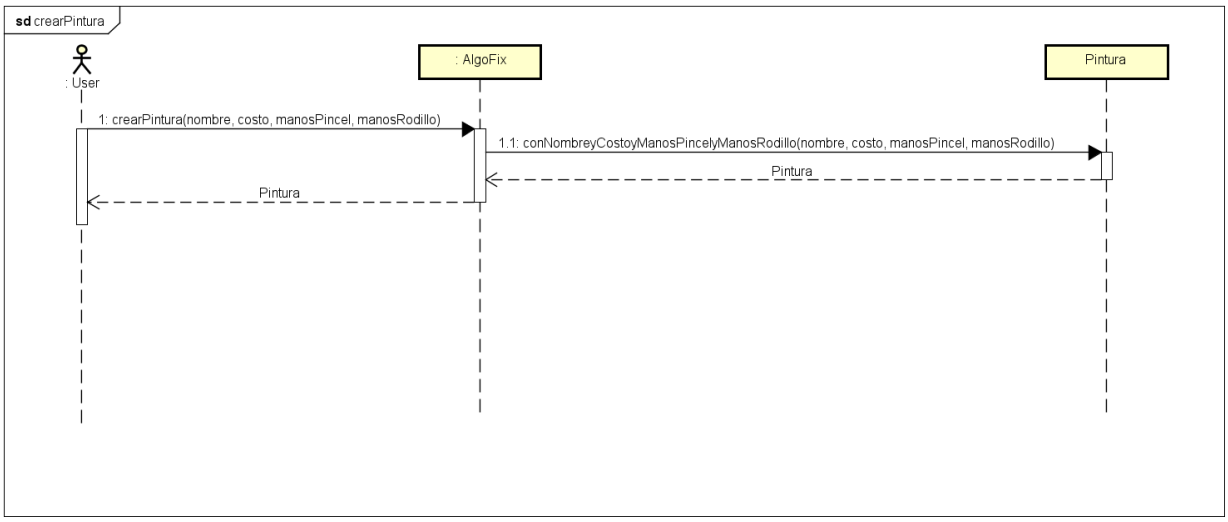

Figura 3: Creación de una pintura.

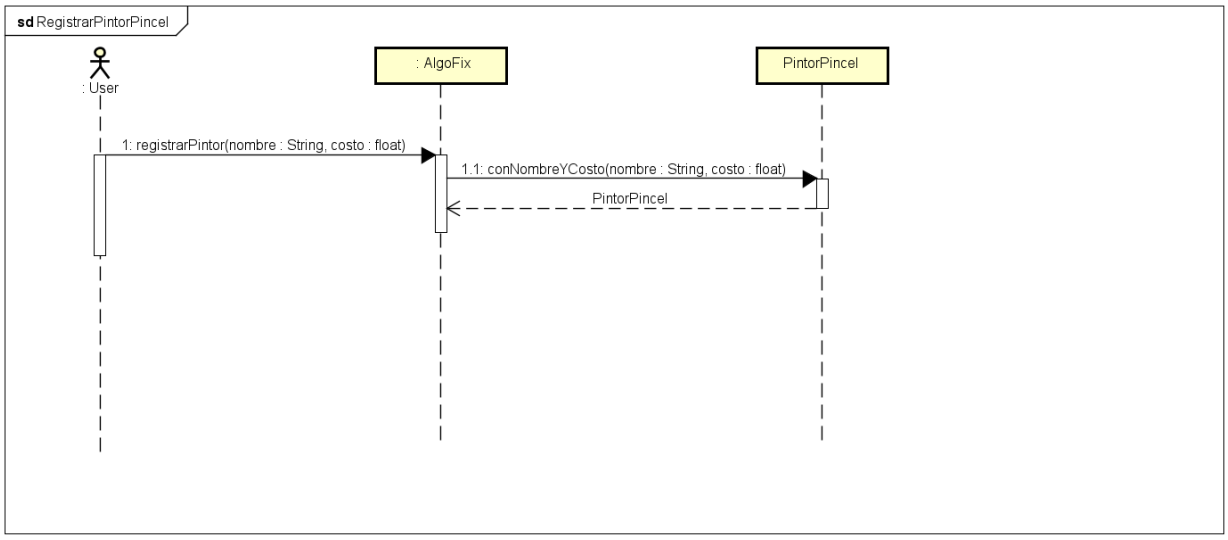

Figura 4: Registro de un pintor de pincel.

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