

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
2.1. AlgoFix	2
2.2. Pintor	2
2.3. Pintura	2
3. Diagramas de clase	3
4. Detalles de implementación	3
4.1. AlgoFix	3
4.2. Cálculo del Presupuesto	4
4.3. Pintor	4
4.4. Pintura	4
5. Excepciones	5
5.1. Responsabilidades en la comprobación de excepciones	5
6. Diagramas de secuencia	5

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

2.1. AlgoFix

En caso de que haya dos pintores que presenten el mismo presupuesto mínimo, se devuelve el primer pintor que se haya registrado. Además, si no hay un pintor registrado a la hora de pedir un presupuesto mínimo se lanza una excepción explicada en su debida sección.

2.2. Pintor

Un pintor no es identificado unívocamente por su nombre, es decir, se permite que se registre mas de un pintor con el mismo nombre y costos por hora distintos. El costo por hora se permite que sea 0 (mano de obra gratuita) y no se permite como nombre un string vacío. No se permite costos por hora negativos. La cantidad de horas que le lleva a un pintor pintar para cada tipo de pintado es constante.

2.3. Pintura

Al igual que un pintor, una pintura no es identificada unívocamente por su nombre, se pueden crear varias pinturas de mismo nombre y costos distintos. El costo por hora se permite que sea 0 (Pintura gratuita). La cantidad de manos no puede ser 0 ni negativo. El costo por litro tampoco puede ser negativo. No se permite un string vacío como nombre. La cantidad de litros que consume la pintura para cada tipo de pintado es constante.

3. Diagramas de clase

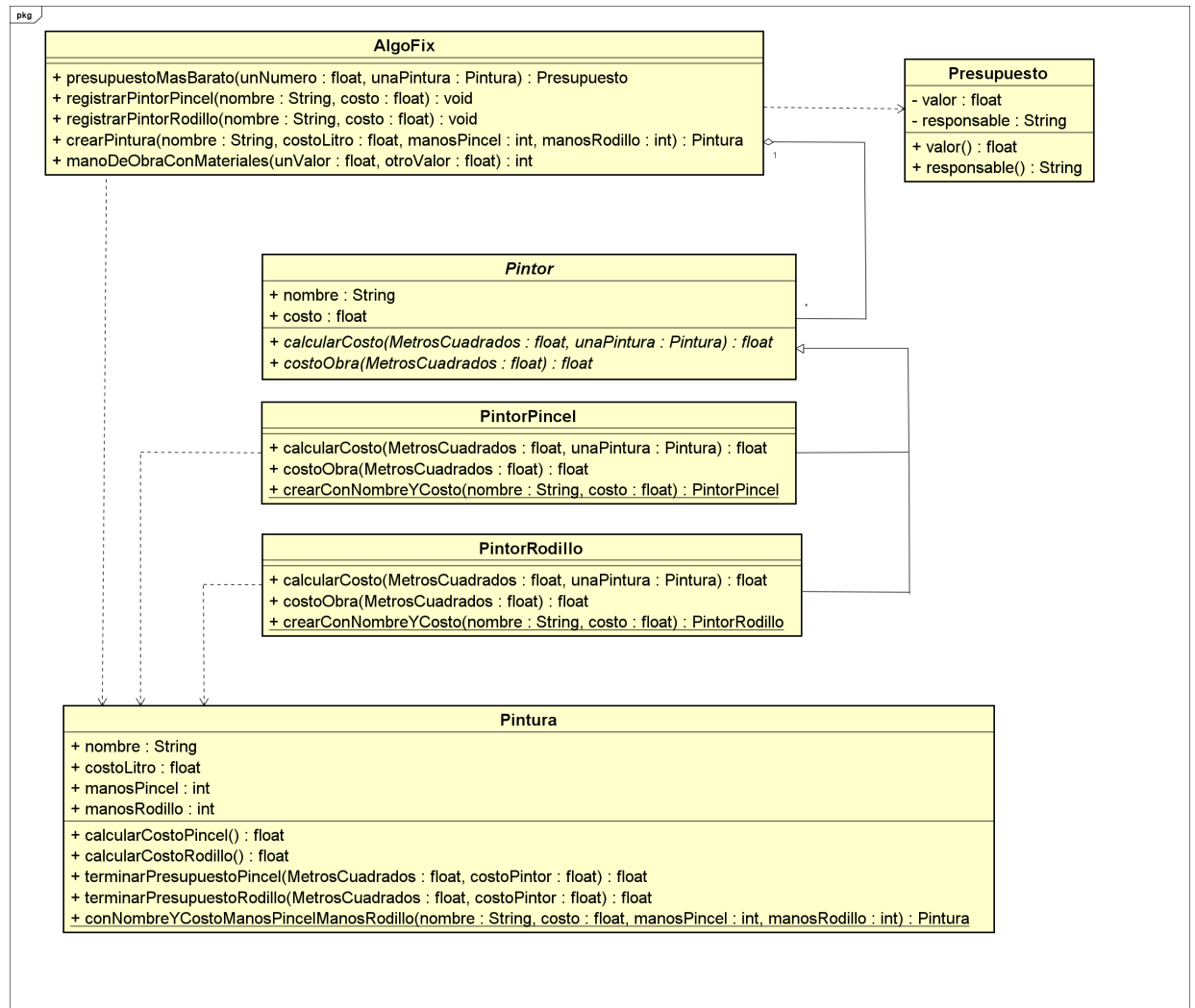


Figura 1: Diagrama de Clases de la solución planteada.

4. Detalles de implementación

4.1. AlgoFix

El método principal del trabajo práctico era la implementación del método *presupuestoMasBarato*. En esta implementación, los pintores que se hayan creado en *AlgoFix* se guardan en una misma *OrderedCollection*, indistintamente de que tipo de pintores este registrando. Lo que permite esto es que en un futuro se puede decidir 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. 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 litrosM2 a la cantidad de litros necesarios para pintar un metro cuadrado, sabemos que el presupuesto se calcula de la siguiente manera:

$$(M2 \cdot \text{costoHora} \cdot \text{Manos} \cdot \text{cantHoras}) + (M2 \cdot \text{costoLitro} \cdot \text{litrosM2} \cdot \text{cantManos})$$

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

$$(M2 \cdot \text{Manos}) \cdot [(\text{costoHora} \cdot \text{cantHoras}) + (\text{costoLitro} \cdot \text{litrosM2})]$$

Finalmente, llamando costoObra a $(\text{costoHora} \cdot \text{cantHoras})$ y costoPintar a $(\text{costoLitro} \cdot \text{litrosM2})$, tenemos que el presupuesto se calcula como:

$$(M2 \cdot \text{Manos}) \cdot (\text{costoObra} + \text{costoPintar})$$

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

4.3. Pintor

Se tiene 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), resulta 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 se puede decidir implementar muchos otros tipos de pintores, se decidió 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, el Pintor se encarga del cálculo del costoObra mostrado en **4.2 Cálculo del Presupuesto**, mientras que delega el resto del cálculo a la Pintura.

4.4. Pintura

La clase Pintura tiene como única responsabilidad terminar de calcular el presupuesto para cada tipo de pintado. Es decir, la pintura sabe cuantos litros consume cada tipo de pintado. Cuando se le pida a la pintura terminar el presupuesto lo que hace es calcular el factor común de la operación mostrada anteriormente y el costoPintar. Luego, opera con los datos y devuelve el presupuesto final. Por como fue implementada la solución general del trabajo práctico, si en un futuro se quisiese agregar tipo de pintados diferentes se deberían crear métodos para dichos tipos en la clase pintura.

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 se le diese 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, se decidió 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

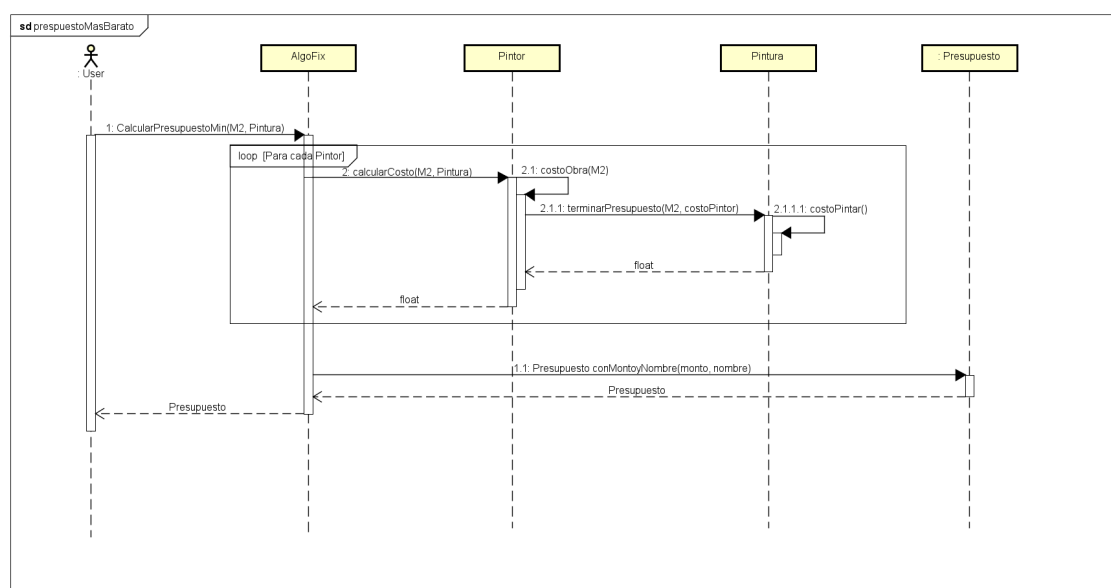


Figura 2: Calculo del presupuesto mínimo.

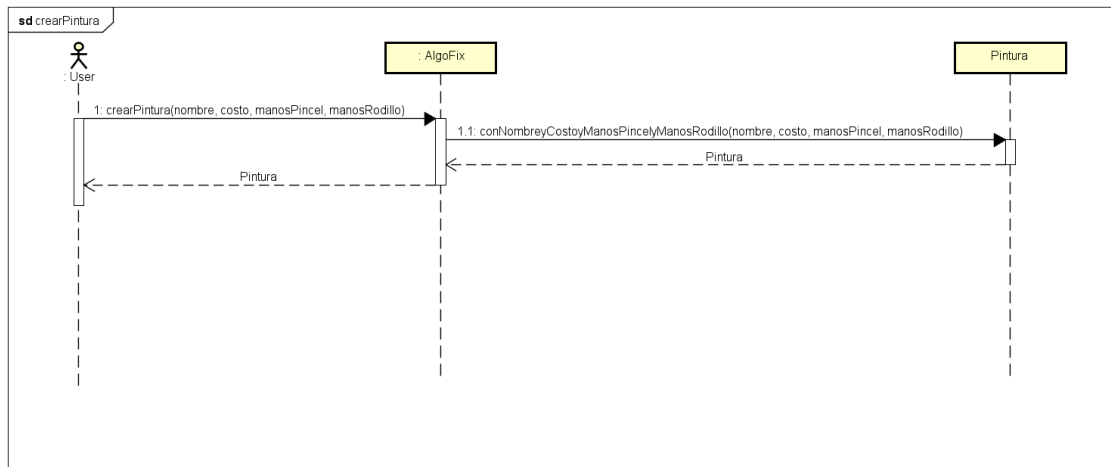


Figura 3: Creación de una pintura.

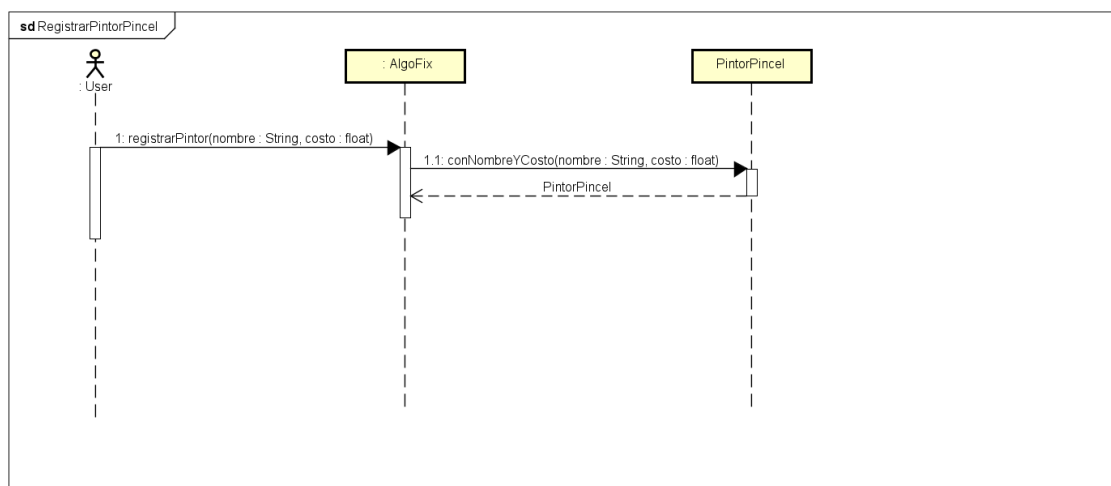


Figura 4: Registro de un pintor de pincel.

En caso de querer registrar un pintor de rodillo, el diagrama sería exactamente igual salvo que se instanciaría un pintor de tipo rodillo. Cabe destacar que una vez que AlgoFix recibe la instancia del pintor este la inserta en una `OrderedCollection` de pintores.