Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 2
Primer cuatrimestre de 2020

Alumno:	CORREA, Valentina Laura
Número de padrón:	104415
Email:	vcorrea@fi.uba.ar

Índice

1. Introducción

El presente informe tiene como objetivo exponer los conocimientos adquiridos en relación a los pilares de la programacion orientada a objetos. Para esto, se propone una solucion al primer Trabajo Practico de 'Algoritmos y Programacion III', la cual esta realizada en el lenguaje Smalltalk, y resuelve el funcionamiento de una empresa de pintores utilizando como herramienta su implementación en Pharo.

2. Supuestos

El enunciado no presentó una gran cantidad de inconvenientes pero pese a esto se resolvieron algunas situaciónes particulares de la siguiente manera.

- Si un pintor posee la tecnica de pintar con pincel, será agregado a la lista de pintores. Si luego este aprende a pintar utilizando un rodillo, el mismo se tendrá que volver a registrar. De esta forma, la cantidad de pintores anotados en la lista no equivale a la cantidad total ya que esta puede contener repetidos. Esto se eligio asi para lograr una mejor y mas visual implementacion.
- Si la pintura que se quiere registrar, posee un nombre invalido, se determino que en esas condiciones no habra forma de distinguirla, por ende no se podra utilizar. Por otro lado, se asume que la cantidad de manos necesarias, tanto para utilizar rodillo como para pincel, siempre seran mayores estrictas a 0.

3. Diagramas de clase

En esta sección se encuentran los diagramas de todas las clases pertenecientes al modelo. Para una mejor visibilidad de éstos, se organizó de la siguiente manera: en un primer lugar, se encuentra un diagrama de todas las clases que componen el modelo, mostrando unicamente las relaciones entre ellas. Luego de este, se van a mostrar en cada detalle como está compuesta cada una; teniendo en cuenta el diagrama general para evitar el exceso de notación.

3.1. Diagramas

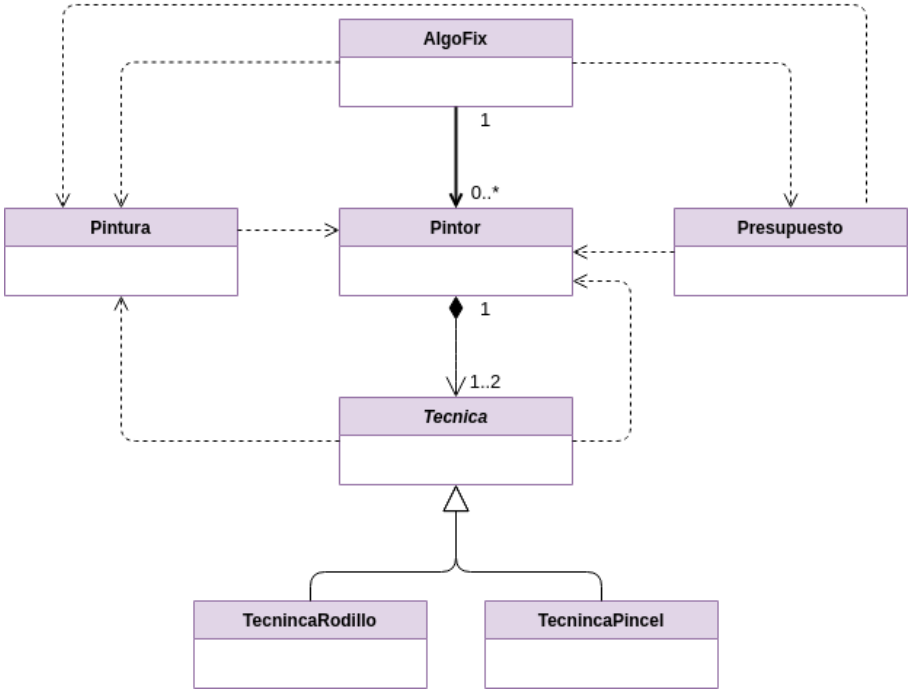

Figura 1: Diagrama general con todas las relaciones entre las clases.

Figura 2: Diagrama detallado de la clase AlgoFix.

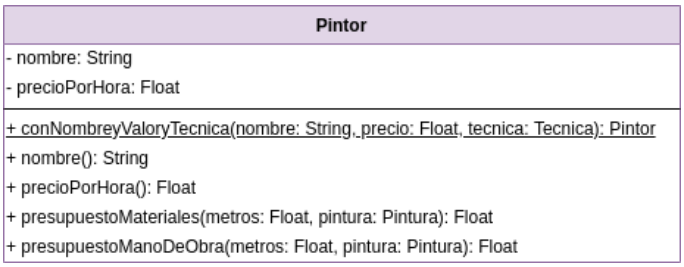

Figura 3: Diagrama detallado de la clase Pintor.

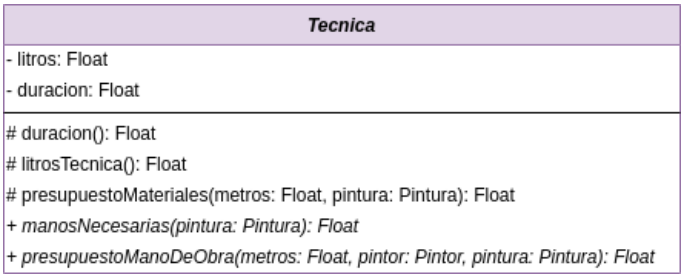

Figura 4: Diagrama detallado de la clase abstracta Tecnica.

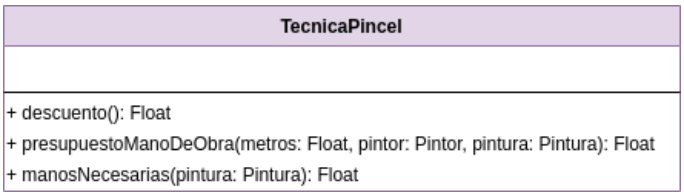

Figura 5: Diagrama detallado de la clase TecnicaPincel.

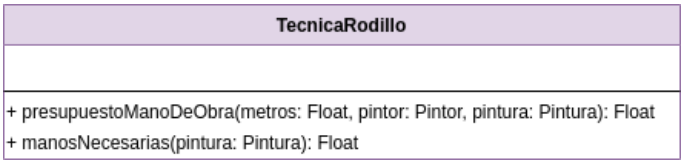

Figura 6: Diagrama detallado de la clase TecnicaRodillo.

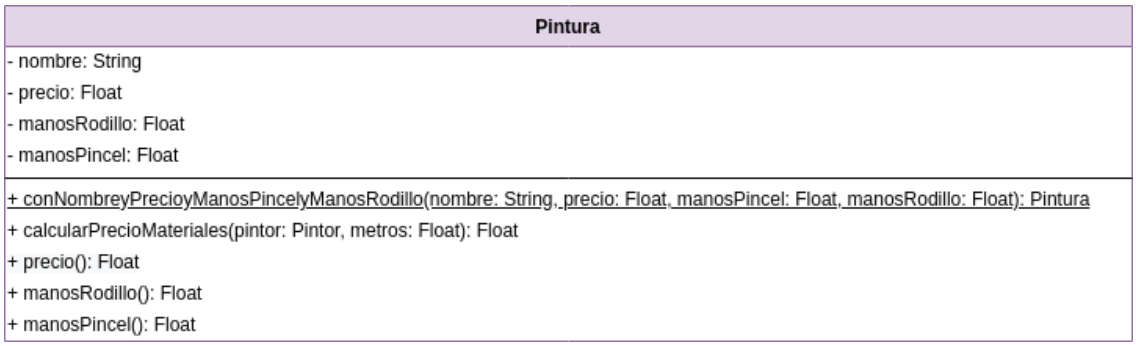

Figura 7: Diagrama detallado de la clase Pintura.

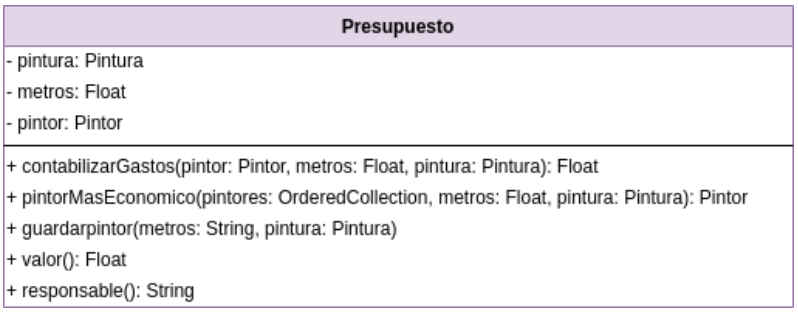

Figura 8: Diagrama detallado de la clase Presupuesto.

3.2. Consideraciones

En los diagramas particulares se excluyeron algunos métodos no muy relevantes a lo que nos interesaba ver, como por ejemplo, algunos setters; todos los demás metodos se creyeron interesantes para mostrar. Tambien se exluyeron atributos en ciertos lados, como por ejemplo tecnica en Pintor, o pintores en AlgoFix, ya que están incluidos como clases en el diagrama general.

Con respecto a las relaciones que se forman entre los objetos, tanto AlgoFix como Pintor, *usan los servicios(métodos) proporcionados por el objeto al cual señalan* (con línea negra sólida). También cumplen la relación de que AlgoFix *tiene* pintores, o un pintor *usa* una determinada tecnica, por lo cual nos dice que aquella relación es de asociación. Además, desde pintor hacia técnica se incluyó una relación de composicón, ya que no tiene sentido que exista una técnica si no hay alguien que la pueda poseer.

Por otro lado, si observamos a tecnica, notamos que representa una clase abstracta, por ende de ella heredan los dos tipos de tecnicas posibles en el modelo.

Finalmente, las flechas punteadas hacen referencia a una relación de dependencia ya que *no se tiene como atributo al señalado* pero si *se lo recibe como parametro.*

4. Detalles de implementación

Dentro de este apartado, se va a hacer mención de algunos algoritmos interesantes sobre los cuales se tomaron ciertas decisiones; y también sobre algunas decisiones de diseño que fueron tomadas a lo largo del trabajo.

4.1. Algunas implementaciones

El algoritmo que resulta mas complejo en el modelo resulta ser el que se encuentra dentro del objeto Presupuesto y calcula el mismo, el cual esta notado como **contabilizarGastosPara: unPintor conMetros: unosMetros yPintura: unaPintura**. Lo que hace este metodo es, dado un conjunto de pintores, cierta pintura y una cantidad de metros cuadrados a pintar, determinar cual es el pintor que ofrece el mejor precio.

Para mantener el encapsulamiento, se opto por que cada objeto se calcule su propio presupuesto, ya que cada pintor cuenta con la suma del precio de una mano de obra (calculada por cada pintor) y el costo de los materiales (calculados por la pintura). A su vez, dentro de cada uno de ellos se hace la delegación de estos calculos hacia la tecnica, ya que es la unica que cuenta con los datos restantes, referidas al estilo de la misma. Estas consideraciones dan como resultado que el metodo pueda implementarse de la siguiente manera:

```
contabilizarGastosPara: unPintor conMetros: unosMetros yPintura: unaPintura

    ^ (unPintor presupuestoManoDeObraParaMetros: unosMetros conPintura: unaPintura)
+ (unaPintura calcularPrecioMaterialesDe: unPintor paraMetros: unosMetros).
```

4.2. Cuestiones de diseño

A continuación se enuncian algunas decisiones que se creyeron mejores para el modelado:

- Respecto a la tecnica del pintor: En un primer lugar, se incluyeron solamente dos objetos diferentes, TecnicaPincel y TecnicaRodillo, y de esta forma se mantuvo hasta que el programa pase las tests proporcionadas por la catedra. El pintor siempre realizo delegaciones hacia su tecnica, y gracias al polimorfismo aplicado no fue necesario preguntar a cual se le estaba mandando; lo cual tambien fue una de las razones por las cuales el codigo repetido fue en aumento. Por todo esto, se evaluo la utilizacion de una clase abstracta la cual pueda abarcar ambas. Pese a que se esta estableciendo una relacion fuerte, ambas comparten los mismos mensajes y entre cada una y la madre se establece la relacion de *es un*. Por estos motivos fue que se decidio utilizar la clase abstracta Tecnica de la cual hereden ambas.

- Todas las excepciones inicialmente se incluyeron en la clase AlgoFix, pero para mantener el encapsulamiento, luego se decidio que debian estar en la clase que no iba a poder proceder con sus metodos particulares si esta se lanzaba, la que se iba a ver particularmente afectada.

- Tanto en el diagrama de clases como en el de secuencia, se redujeron los nombres de los metodos para lograr mas simplicidad en los mismos y para que queden mas genéricos y adaptables para poder ser implementados en cualquier lenguaje.

- Respecto a las tests, se intento mantener en cada una una separacion del *arrange, act, assert*, para lograr que quede una mejor visualizacion y quede mas organizado y explicito.

  Por otro lado, tambien se utilizo el SetUp, para garantizar que las pruebas sean completamente unitarias y el foco este en ese objeto de la clase que esta siendo testeada, y los que necesite para algun mensaje queden en segundo plano.

  Por ultimo, las mismas fueron organizadas por orden de creacion y tambien subdivididas en categorias segun la clase bajo test.

- Respecto a las pruebas de integracion propias (en algoFixTest) se decidio hacer pocas pero abarcativas e integradoras como se pide que sean. Con estas y las pruebas unitarias que solo prueban comportamiento, se logro hacer una **cobertura del 100 % del programa (excluyendo los tests de la catedra)**.

5. Excepciones

A lo largo del modelo se notaron convenientes incluir las excepciones a continuación

NoHayPinturaParaCalcularPresupuesto Esta excepcion se llama dentro del objeto Presupuesto, antes de que realice el calculo de cual es el pintor mas economico; ya que si no se tiene una pintura valida (nil) no se puede realizar dicho cálculo.

NoSeRegistroNombrePintura Esta excepcion se lanza si al momento de instanciar una Pintura, dentro de su método de clase, descubre que su nombre es nil. Dadas estas condiciones el programa responde que no se pueden registrar.

NoSeEnvioUnPunteroDePintores Nuevamente dentro del objeto Presupuesto, si al querer calcular el pintor mas barato, no recibimos un puntero de pintores, el cálculo no podrá ejecutarse, ya que no tenemos forma de interactuar con ningún pintor.

NoSePuedeRegistrarPintorSinNombre Dentro del método de clase de Pintor, al igual que con pintura, si no se registra un nombre del mismo no se tiene forma de distinguirlo, por ende, no se podrá cargar a algoFix tampoco.

NoSeRegistraronPintores Una vez mas en Presupuesto, si al querer realizar el cálculo nos encontramos con que no tenemos ningun pintor cargado, nuevamente no se podra concretar dicho método.

6. Diagramas de secuencia

En esta sección se incluirán algunos diagramas de secuencias interesantes de ver en el codigo, a partir del lenguaje UML.

6.1. Consideraciones

Para una mejor visibilidad de la imagen, se opto por reducir los nombres de los métodos mas extensos. Al pie de la imagen se podrán encontrár los nombres reales. Respecto a los metodos de clase para la creacion de algunos objetos, tambien fueron excluidos del diagrama y reemplazados por un *«create»* dado que no se veia de un aporte considerable a lo que se quiere mostrar.

6.2. Test 01 de AlgoFix

6.2.1. Diagrama General

En el siguiente diagrama se verá el seguimiento de las partes mas representativas del código, según el test bridado por la catedra:
test01PresupuestoPintorUtilizaPincelConPinturaAlbaOfreceElMenorPresupuesto.

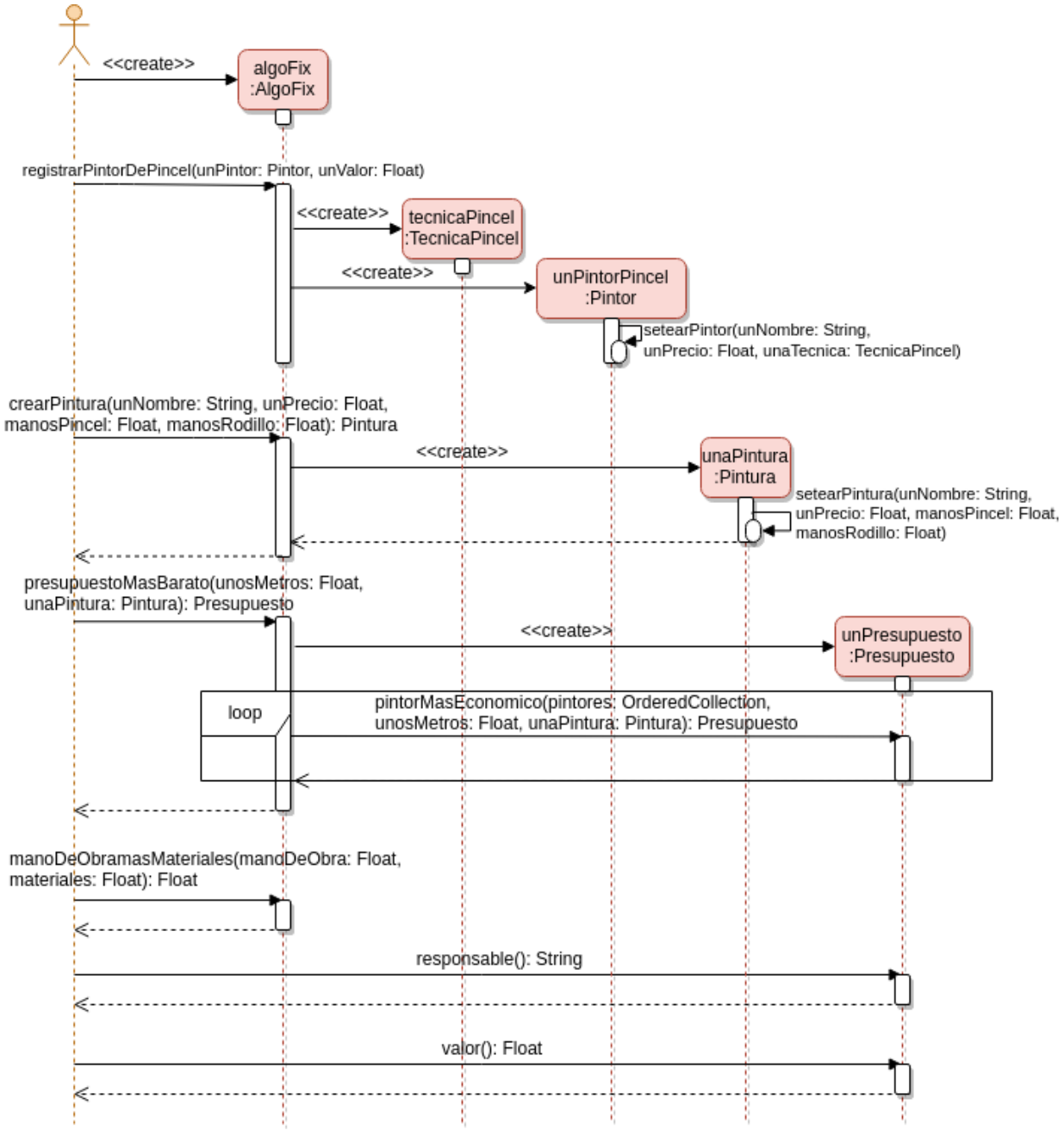

Figura 9: Diagrama general del test 01.

Como se ve en el diagrama, para determinar el calculo del pintor mas economico se genera un loop hasta encontrarlo. En este caso, al ser un solo pintor, este se realiza una sola vez.

A continuacion, se escribirán nuevamente los nombres reales de los métodos que los tienen reducidos(en orden de llamado, tal como figura en el diagrama):

- registrarPintorDePincelconValorHora(unPintor: Pintor, unValor: Float)
- setearNombreyPrecioyTecnica(unNombre: String, unPrecio: Float, unaTecnica: TecnicaPincel)

- crearPinturaconPrecioPorLitromanosPincelmanosRodillo(unNombre: String, unPrecio: Float, manosPincel: Float, manosRodillo: Float)
- setearNombreyPrecioyManosPincelyManosRodillo(unNombre: String, unPrecio: Float, manosPincel: Float, manosRodillo: Float)
- presupuestoMasBaratoParaPintarMetrosCuadradosconPintura(unosMetros: Float, unaPintura: Pintura)
- pintorMasEconomicoDePintoresparaMetrosconPintura(pintores: OrderedCollection, unosMetros: Float, unaPintura: Pintura): Presupuesto

6.2.2. Diagrama particular del calculo del presupuesto

Si se quisiera ver con mas detalle el seguimiento del método *pintorMasEconomicoDePintoresparaMetrosconPintura...* , podemos observar el siguiente diagrama. El mismo respeta los ordenes en el cual los objetos fueron creados.

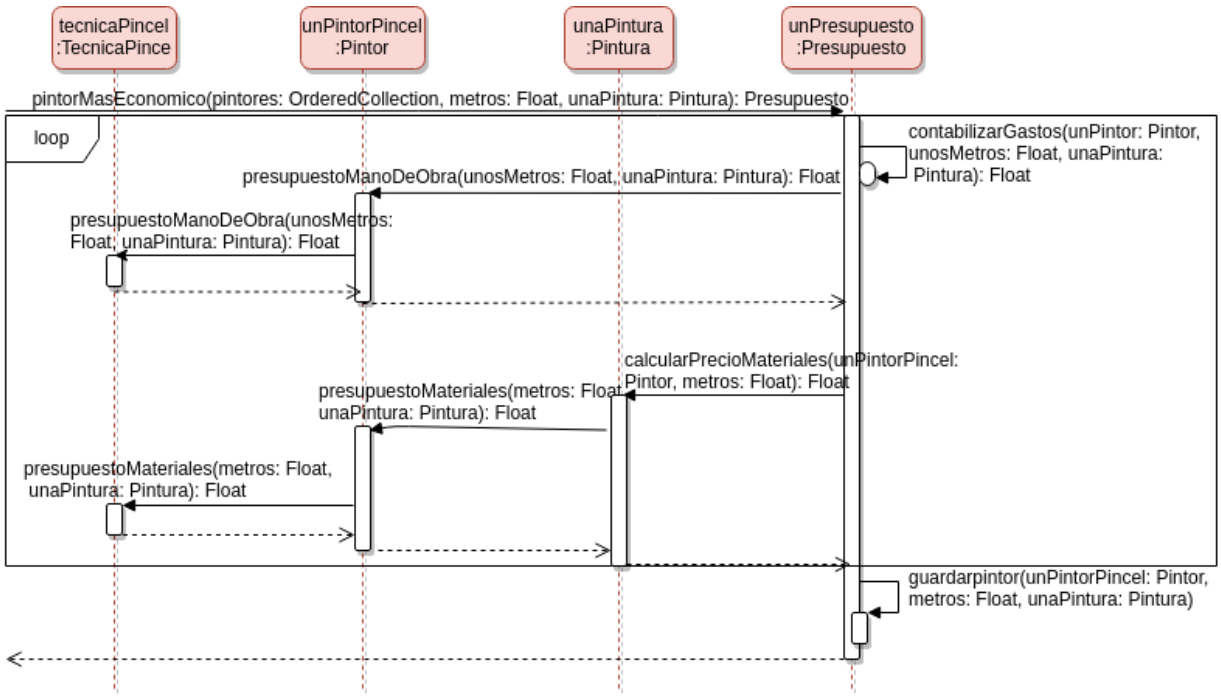

Figura 10: Diagrama particular de la funcion que retorna un Presupuesto en el test 01.

Nuevamente, se detallan a continuación los metodos que fueron abreviados:

- pintorMasEconomicoDePintoresparaMetrosconPintura(pintores: OrderedCollection, unosMetros: Float, unaPintura: Pintura): Presupuesto
- contabilizarGastosPara(unPintor: Pintor, metros: Float, unaPintura: Pintura): Float
- presupuestoManoDeObraParaMetrosconPintura(metros: Float, unaPintura: Pintura): Float
- presupuestoManoDeObraParaMetrosconPintoryPintura(metros: Float, unPintorPincel: Pintor, unaPintura: Pintura): Float

- calcularPrecioMaterialesDeparaMetros(unPintorPincel: Pintor, metros: Float): Float
- presupuestoMaterialesParaMetrosyPintura(metros: Float, unaPintura: Pintura): Float
- presupuestoMaterialesParaMetrosyPintura(metros: Float, unaPintura: Pintura): Float

6.2.3. Diagrama particular del calculo de la mano de obra

Si se quisiera ver aún mas en detalle la forma en que, en este caso TecnicaPincel, calcula los presupuestos, es la siguiente:

Para la mano de obra del pintor se hizo este seguimiento

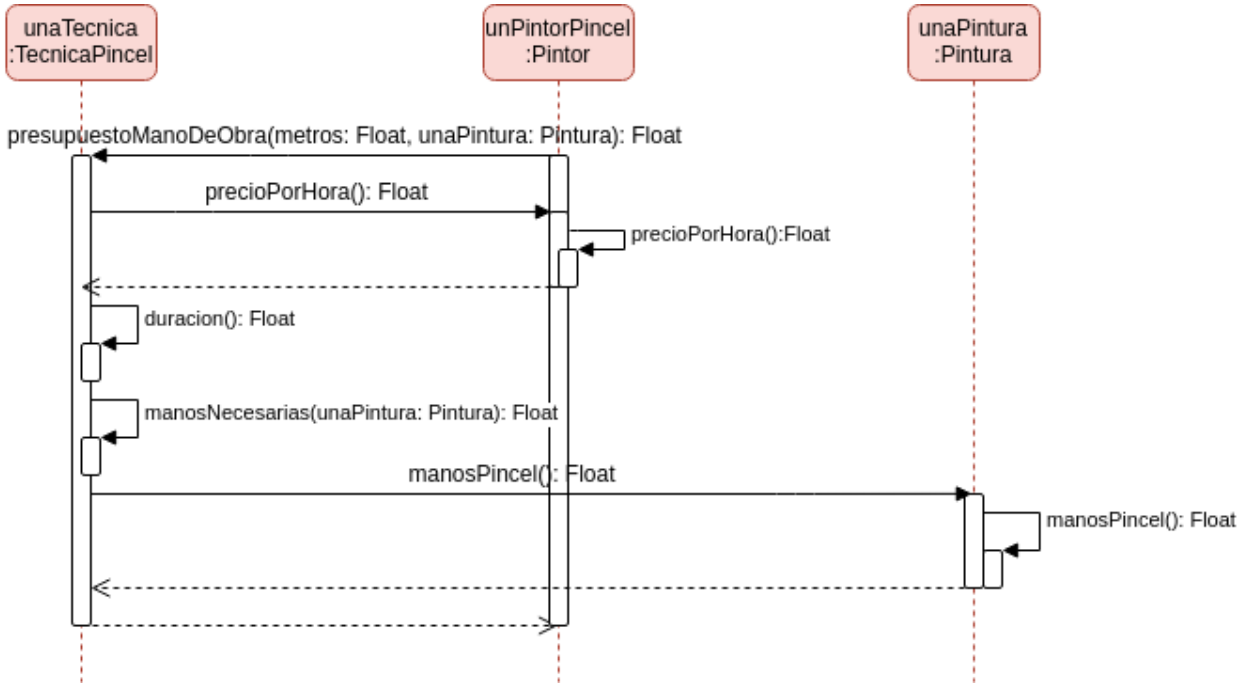

Figura 11: Diagrama particular del calculo de materiales que realiza la Tecnica, en el test 01.

Los métodos que fueron abreviados en este diagrama son:

- presupuestoManoDeObraParaMetroconPintura(metros: Float, unaPintura: Pintura): Float
- manosNecesariasConPintura(unaPintura: Pintura): Float

6.2.4. Diagrama particular para el calculo de los materiales

Por otro lado, para el cálculo de materiales se observa lo siguiente

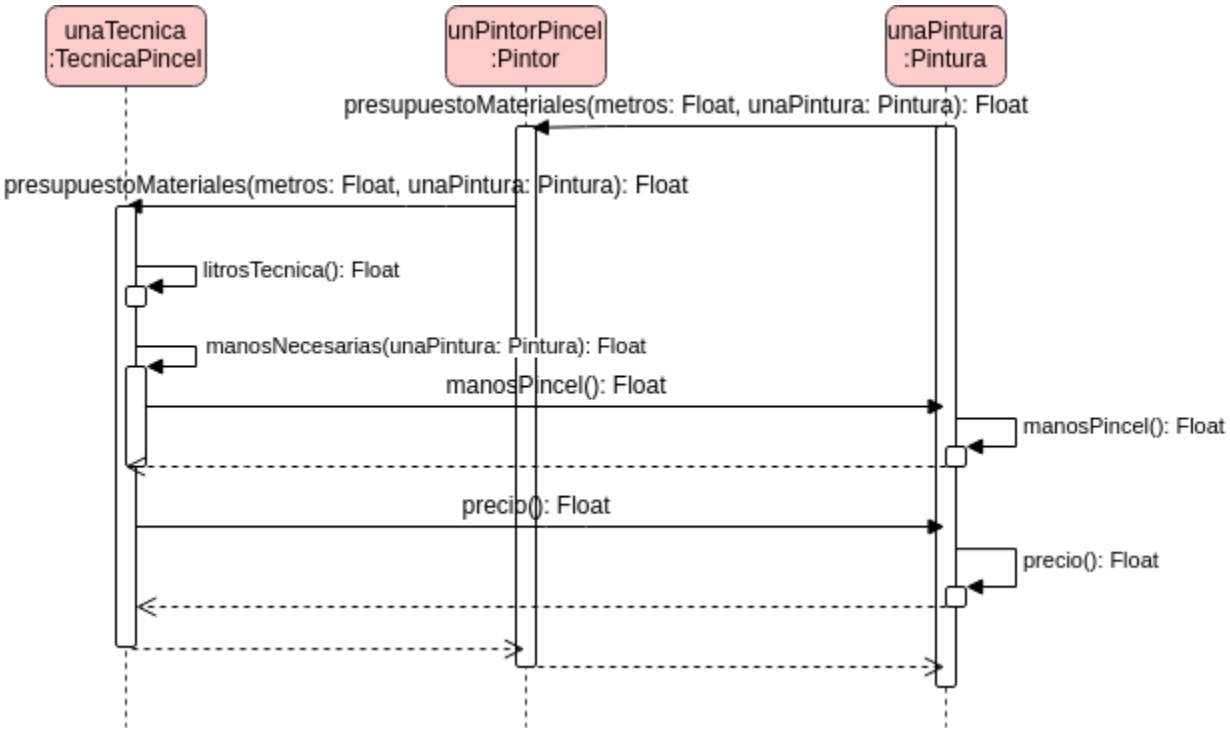

Figura 12: Diagrama particular del calculo de la mano de obra que realiza la Tecnica, en el test 01.

En éste se optó por abreviar

- presupuestoMaterialesParaMetros(metros: Float, unaPintura: Pintura): Float
- presupuestoMaterialesParaMetrosyPintura(metros: Float, unaPintura: Pintura): Float
- manosNecesariasConPintura(unaPintura: Pintura): Float

De esta forma, completamos el seguimiento de todo el test 01, pasando por la totalidad de sus mensajes.

Los restantes 7 tests de integracion de la catedra no presentan mayores variaciones por lo que no se considera de gran necesidad realizaros, dada la semejanza con el presentado.