Trabajo Práctico 1 — Smalltalk

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

Alumno:	Izquierdo, Stephanie
Número de padrón:	104196
Email:	sizquierdo@fi.uba.ar

Contents

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 que permite calcular el mejor presupuesto de un pintor de acuerdo a las necesidades del usuario en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2 Supuestos

Los supuestos realizados fueron:

- Todos los pintores que se registren son distintos.
- No se ingresaran números inferiores a 0
- Se puede ingresar un pintor cuyo valor por hora sea cero (el usuario así tendra posibilidad de registrar un pintor que haga un trabajo gratis)
- No se ingresaran manos de pinturas iguales a 0 o inferior a 0.
- No se ingresaran metros igual o inferior a 0

3 Diagramas de clase

En este diagrama podemos observar las relaciones entre AlgoFix y las otras clases:

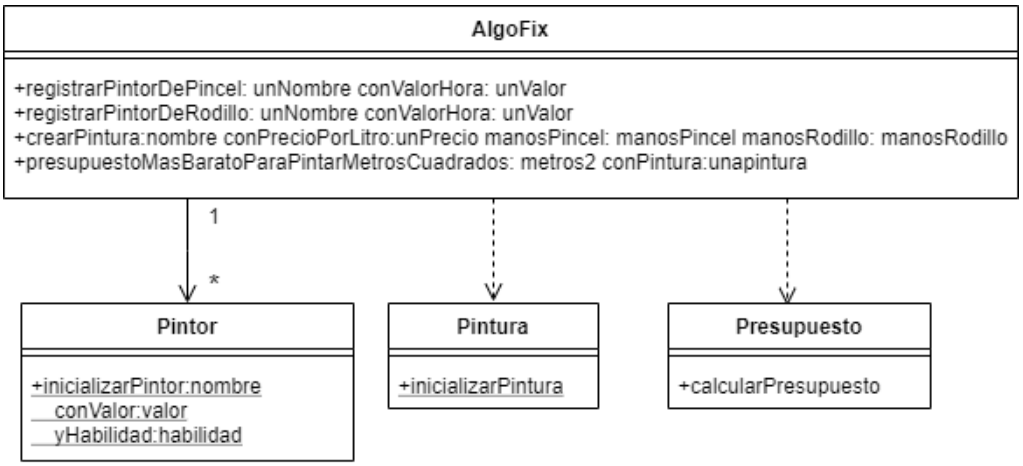

Figure 1: Diagrama de relaciones de AlgoFix.

En el siguiente diagrama podemos observar como algoFix conoce a uno o mas instancias de Pintor y a su vez como cada pintor posee un atributo de habilidad.

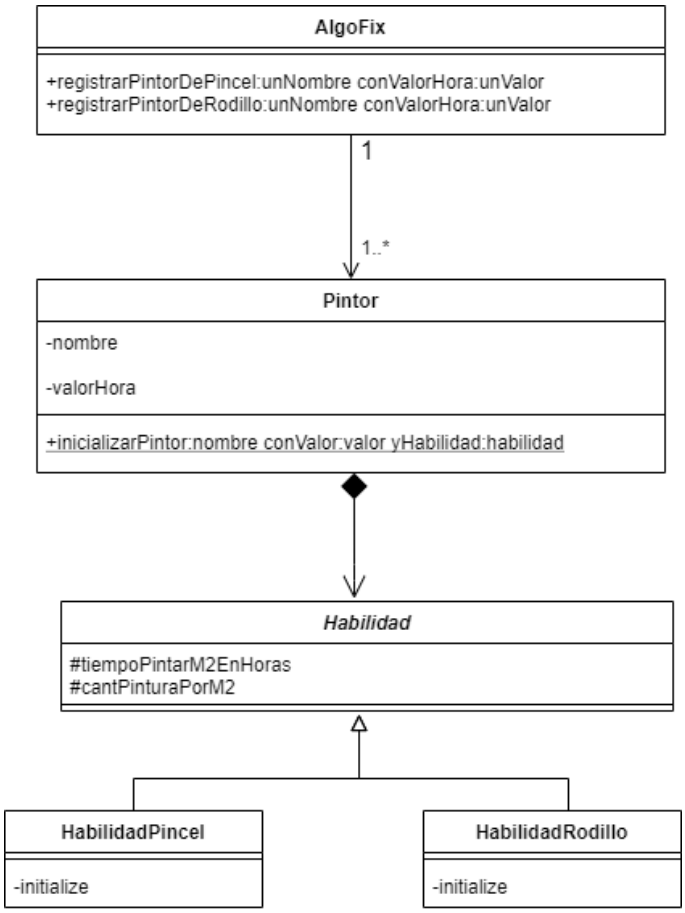

Figure 2: Relacion entre AlgoFix y Pintor.

En el siguiente diagrama se puede observar como se relacionan las distintas clases entre si para calcular un presupuesto mínimo.

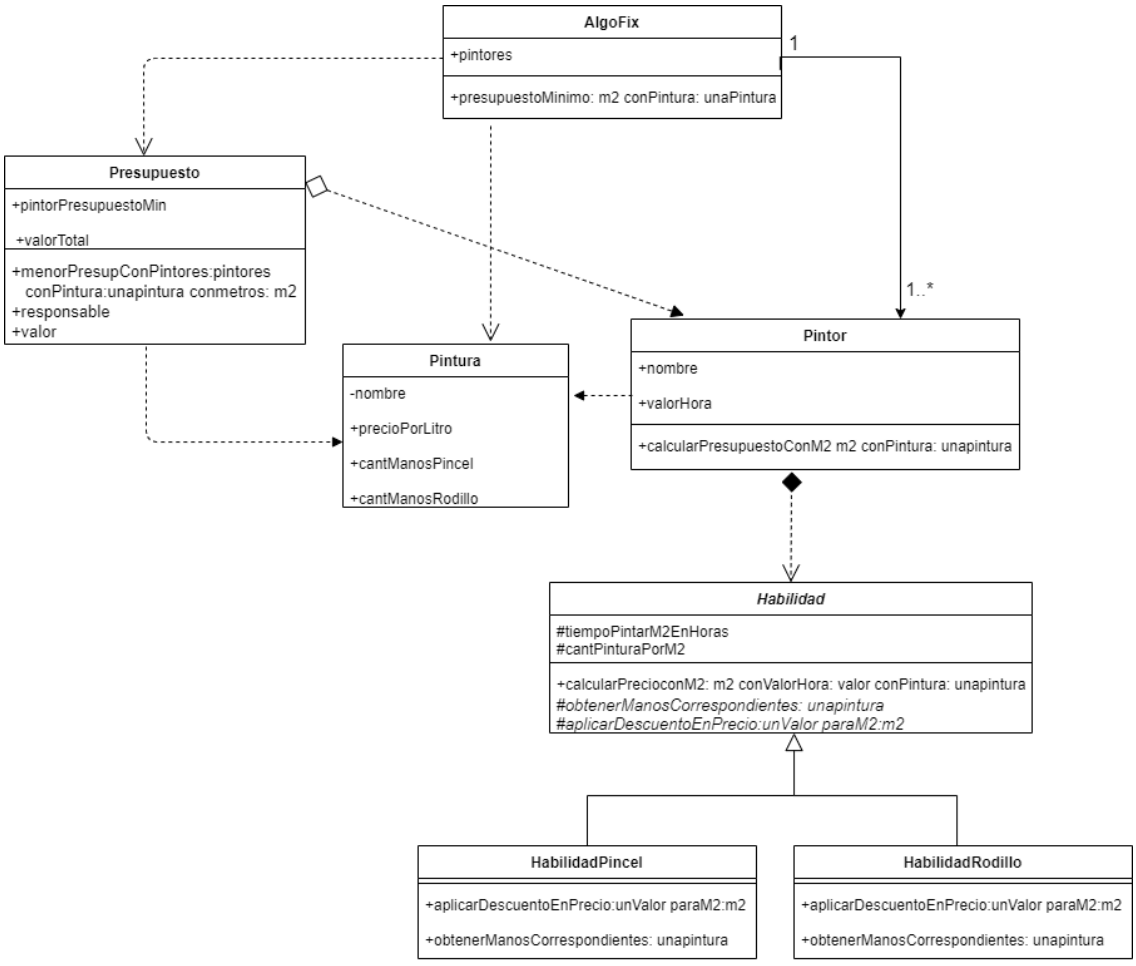

Figure 3:

"presupuestoMinimo: m2 conPintura: unaPintura" refiere al metodo: "presupuestoMasBaratoPara-PintarMetrosCuadrados: metros2 conPintura:unapintura"

"menorPresupConPintores:pintores conPintura:unapintura conmetros: m2" refiere al metodo: "menor-PresupuestoConPintores: pintores conPintura: unapintura conmetros: metros2"

4 Detalles de implementación

4.1 Elecciones de diseño

En mi modelo AlgoFix se encarga de delegar el comportamiento de sus mensajes a las otras entidades.

Cada instancia de Pintor, tiene los atributos: nombre, valorHora, habilidad. Este ultimo atributo puede ser una instancia de HabilidadPincel o Habilidad Rodillo que se determinara cuando se invoquen los metodos "registrarpintordePincel(...)" o "registrarPintordeRodillo(...)".

Para el atributo "habilidad" decidí usar polimorfismo con herencia de forma tal que "HabilidadPincel" y "HabilidadRodillo" hereden de la clase abstracta "Habilidad".

Dado que se cumple la relación "es un" y el comportamiento de "HabilidadPincel" y "HabilidadRodillo" difieren en los valores de los atributos de instancia y la cantidad de manos a utilizar de una pintura para realizar el cálculo de mano de obra más materiales, decidí crear la clase abstracta "Habilidad" para que el comportamiento similar quede en ella, y en las clases hijas se implemente aquello que difiere. Esto simplificó el código y se pudo evitar el código repetido sin necesidad de realizar un modelo más rebuscado usando exclusivamente delegación. A su vez, si bien una relación de herencia es muy fuerte, para la problemática del trabajo práctico no resulta inconveniente alguno porque, para este caso, se cumple dicha relación.

Dado que decidí delegar el cálculo de mano de obra y materiales a la "habilidad" para simplificar el código y cumplir con el encapsulamiento, la clase "Pintura" solo posee métodos de tipo "getter" (ademas del constructor). Entonces así el cálculo de mano de obra y materiales queda encapsulado en la habilidad.

4.2 Cálculo del presupuesto

Para el cálculo del presupuesto más barato, lo que se realizo fue buscar dentro de la coleccion ordenada "pintores", al pintor cuyo cálculo de mano de obra más materiales fuera mínimo mediante el método "detectMin".
Luego de que este método haya retornado la instancia de la clase Pintor, se calcula sobre este último el valor de mano de obra más materiales para que el objeto correspondiente a la clase Presupuesto instancie su atributo "pintorMasBarato" y "valorTotal"

```
menorPresupuestoConPintores: pintores conPintura: unapintura conmetros: metros2
| pintor  valor |

  pintor:= pintores detectMin:[:unpintor | unpintor calcularprecioHoraporM2: metros2
              conPintura: unapintura].

  valor:= pintor calcularPresupuestoConM2: metros2 conPintura:unapintura.

  self inicializarConPintor: pintor yconValorTotal: valor.

  ^self
```

5 Excepciones

Excepcion 1: En caso de que se pida un realizar un presupuesto sin haber registrado al menos un pintor, se lanzara una excepcion dado que no se podra calcular el valor del presupuesto mínimo sin instancias de la clase Pintor.

Excepcion 2: Para realizar un presupuesto la pintura debera ser valida, es decir, debe haber sido creada anteriormente para poder elaborar un presupuesto.

Excepcion 3: No se aceptará registrar pintores, tanto de pinceles como de rodillo, sin un nombre con motivo de que si luego se solicita un presupuesto, no se podrá saber el responsable.

Excepcion 4: No se crearan pinturas sin nombre.

6 Diagramas de secuencia

Secuencia al solicitar el mínimo presupuesto ya habiendo registrado un pintor y creado una pintura:

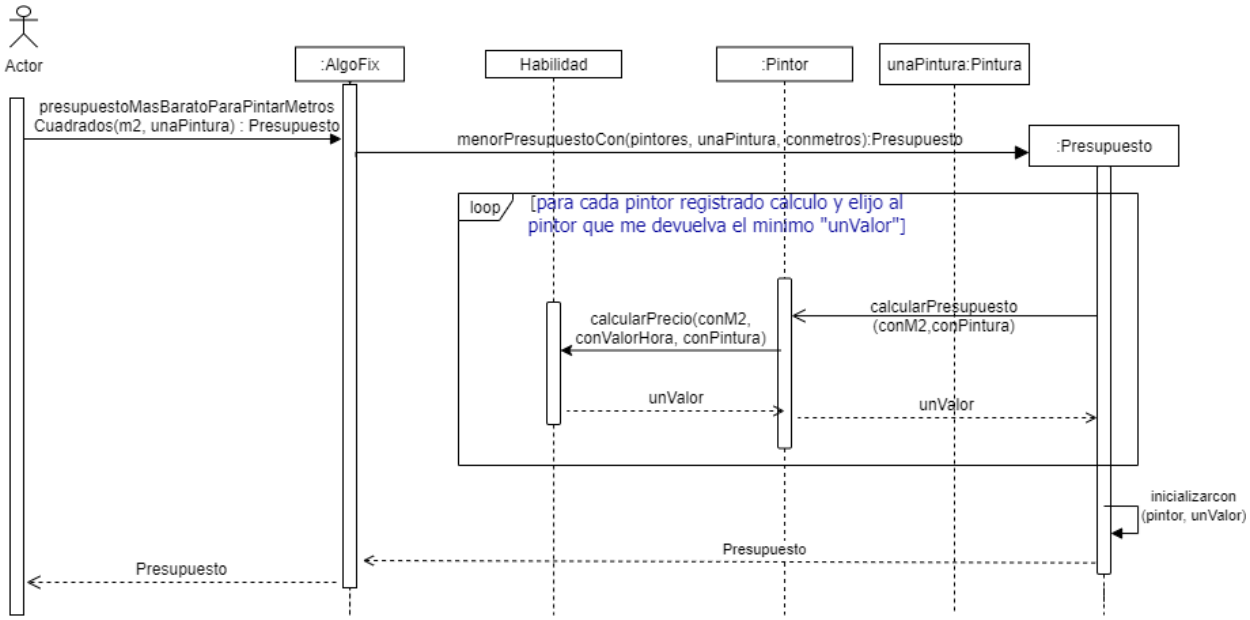

Figure 4: Flujo al calcular un presupuesto.

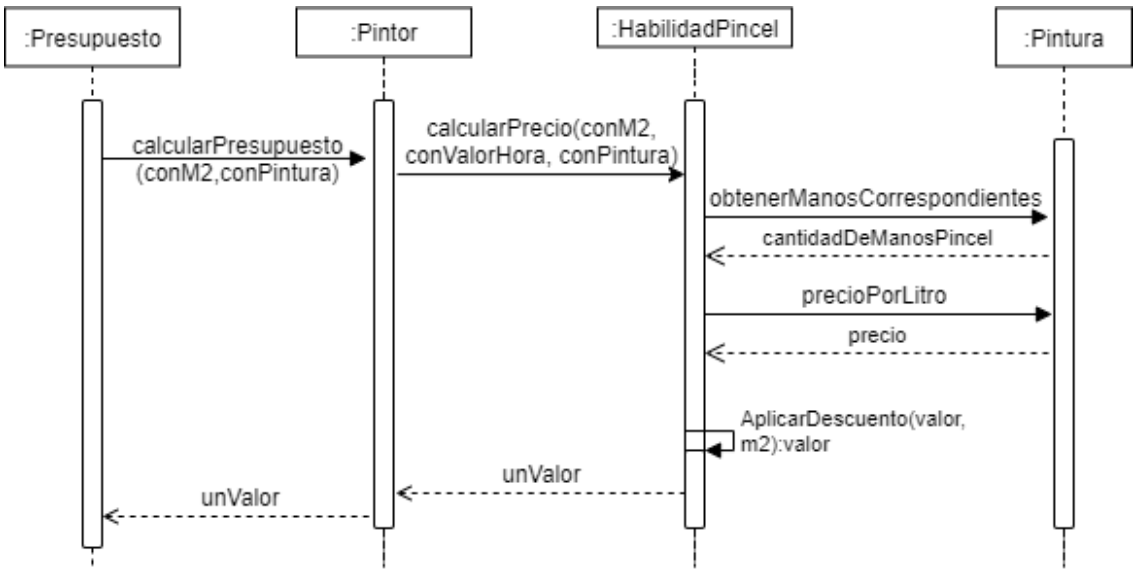

Figure 5: Caso en el cual el pintor posee habilidad pincel.

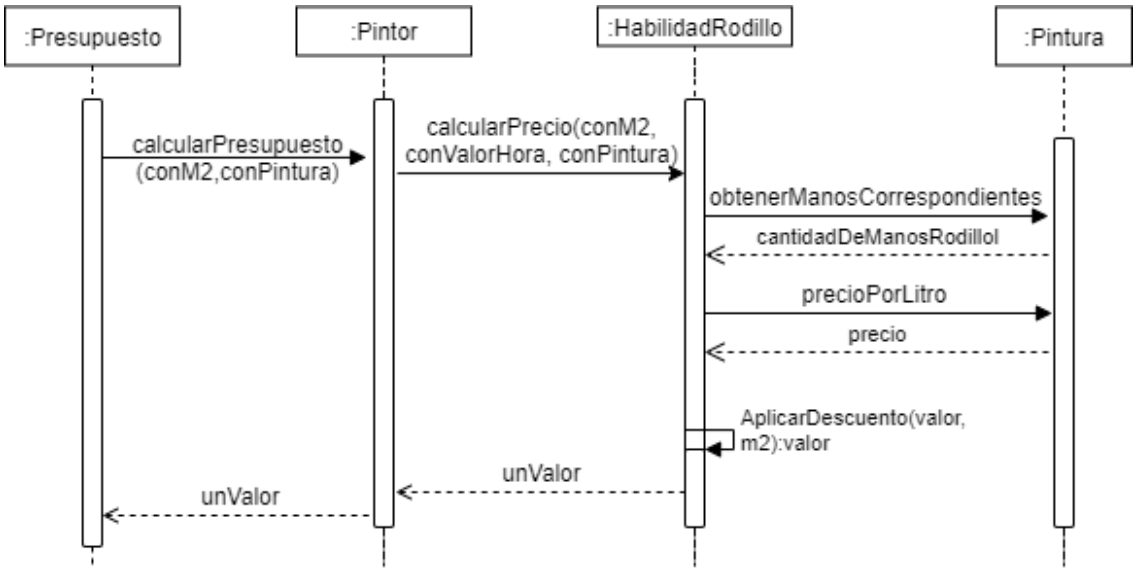

Figure 6: Caso en el cual el pintor posee habilidad rodillo.

Aclaración: En estos diagramas se mostró con énfasis el flujo al seleccionar un pintor habiendo previamente instanciado AlgoFix, registrado un pintor y creado una pintura.

En el siguiente diagrama de secuencia se muestra el caso en el cual se pide calcular un presupuesto sin haber registrado al menos un pintor.

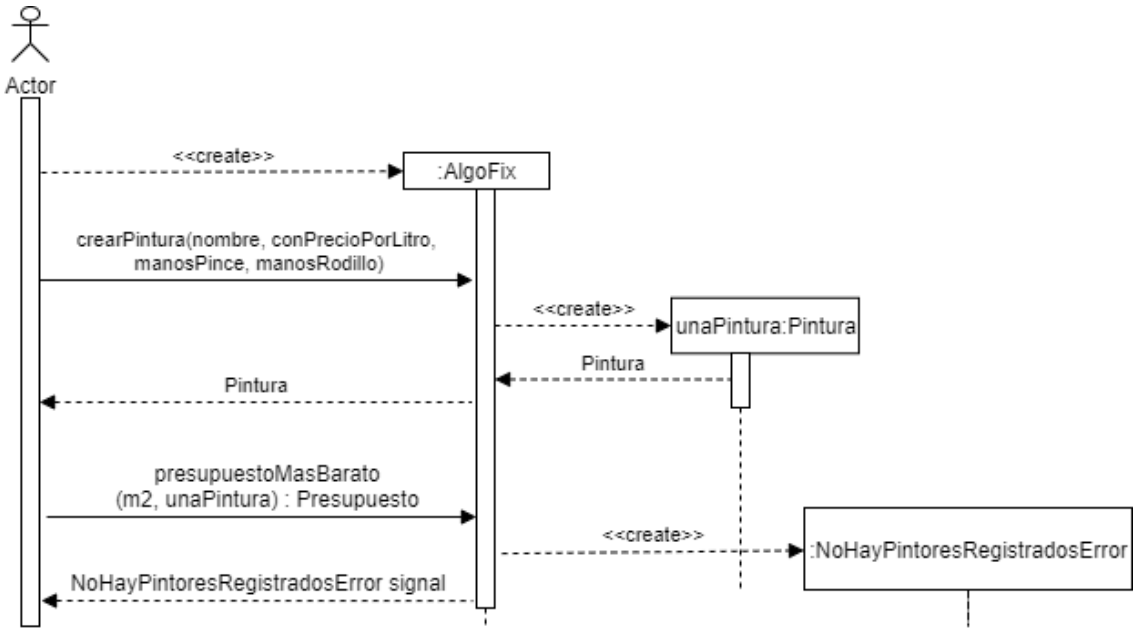

Figure 7: excepcion