Trabajo Práctico 1 — Smalltalk

[7507/9502] Algoritmos y Programación III
Curso 2
Segundo cuatrimestre de 2018

Alumno:	CHIAPPE, Ignacio
Número de padrón:	90340
Email:	nacho.chiappe@gmail.com

Índice

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 un calendario en Pharo utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

El principal supuesto que se adoptó fue que no pueden existir eventos con el mismo nombre en el calendario. Esta determinación se tomó principalmente teniendo en cuenta que el método removerEvento() mencionado en las pruebas toma únicamente como parámetro un nombre. Esto permite utilizar una estructura de diccionario donde la clave sea el nombre del evento.

3. Modelo de dominio

Calendario La clase principal del trabajo práctico y sobre la cual las pruebas envían la mayoría de los mensajes, actuando como intermediario entre los demás objetos. Está compuesta por invitados y eventos.

Evento Uno de los objetos que compone el calendario. Puede ser de tipo *Simple* o *Semanal*. Está compuesto por un nombre, invitados y la o las fechas en las que se lleva a cabo el mismo.

Invitados Los invitados forman parte del calendario, y pueden ser *Personas* o *Recursos*. Las *Personas* pueden superponerse en distintos eventos que se lleven a cabo al mismo tiempo, pero no así los *Recursos*, los cuales sólo pueden formar parte de un único evento a la vez.

4. Diagramas de clase

En el diagrama de clase que se muestra a continuación se puede ver la clase Calendario, y las clases abstractas Eventos e Invitados.

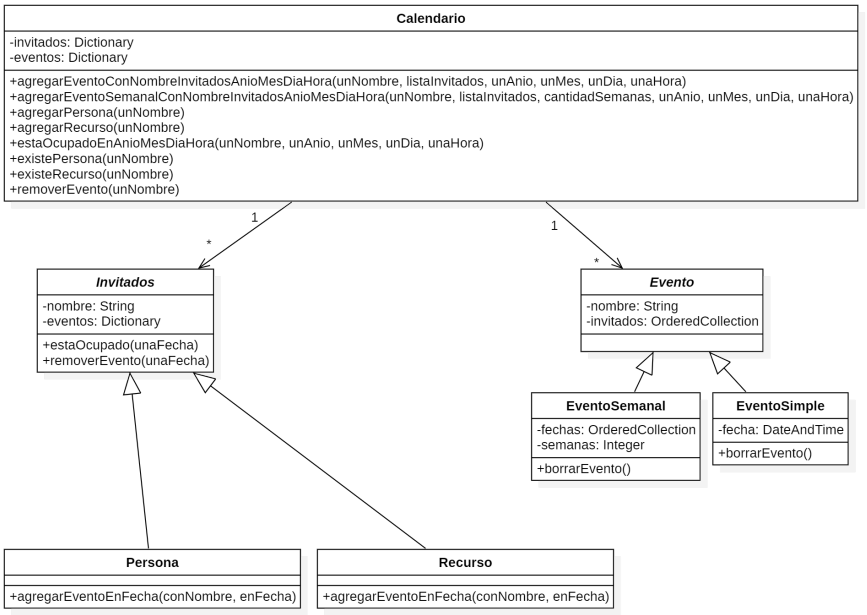

Figura 1: Diagrama de clase.

5. Detalles de implementación

5.1. Calendario

Los métodos más complejos del trabajo fueron los correspondientes a agregar eventos al calendario. Se resolvió utilizando dos tipos de evento: Simples y Semanales. La implementación de los dos métodos (agregarEventoConNombre y agregarEventoSemanalConNombre) es muy parecida. A continuación se detalla el código para eventos simples:

```
agregarEventoConNombre: unNombre invitados: listaInvitados enAnio: unAnio mes: unMes
dia: unDia hora: unaHora

| unaFecha evento coleccionInvitados |
(eventos includesKey: unNombre) ifTrue: [ NombreDeEventoYaExisteError signal ].
unaFecha := DateAndTime year: unAnio month: unMes day: unDia hour: unaHora minute: 0.
coleccionInvitados := OrderedCollection new.
listaInvitados do: [ :invitado |
(invitados includesKey: invitado) ifFalse: [ InvitadoNoExisteError signal ].
coleccionInvitados add: (invitados at: invitado) ].
evento := (EventoSimple new) conNombre: unNombre conInvitados: coleccionInvitados
enFecha: unaFecha.
eventos at: unNombre put: evento.
```

Para el caso de eventos semanales, la principal modificación radica en el mensaje que se le envía a *evento*:

```
evento := (EventoSemanal new) conNombre: unNombre conInvitados: coleccionInvitados
enFecha: unaFecha duranteSemanas: cantidadSemanas.
```

5.2. Evento

Luego de enviado el mensaje correspondiente según se indica en la sección anterior, cada clase agrega el evento a cada invitado, utilizando para ello un diccionario donde la clave es la fecha, lo cual es útil para determinar si una persona o recurso se encuentra ocupado en dicha fecha. En el caso de eventos simples, la implementación es la que se detalla a continuación:

```
conNombre: unNombre conInvitados: coleccionInvitados enFecha: unaFecha

nombre := unNombre.
invitados := coleccionInvitados.
fecha := unaFecha.
invitados do: [ :invitado | invitado agregarEvento: unNombre enFecha: unaFecha ].
```

En el caso de eventos semanales, se realiza una iteración entre 1 y la cantidad de semanas pasadas por parámetro, incrementando el valor de la fecha en 7 días (1 semana) en cada ciclo:

```
(1 to: cantidadSemanas) do: [ :each |
invitados do: [ :invitado | invitado agregarEvento: unNombre enFecha: fecha ].
fechas add: fecha.
fecha := fecha + (Duration weeks: 1) ].
```

6. Excepciones

CantidadSemanasMenorAUnoError Al crear un evento semanal, si el parámetro *semanas* es menor a 1 (uno), lanzará esta excepción.

EventoInexistenteError Esta excepción se lanzará si se intenta remover un evento con un nombre inexistente.

InvitadoNoExisteError Al intentar agregar un evento nuevo para un invitado, o al verificar si un invitado está ocupado, se lanzará esta excepción si dicho invitado no existe.

NombreDeEventoYaExisteError Se lanzará esta excepción si se quiere crear un nuevo evento con un nombre de evento ya existente.

PersonaYaExisteError Si se intenta agregar una persona ya existente al calendario se lanzará esta excepción.

RecursoOcupadoError Esta excepción se lanzará si se intenta agregar un evento al calendario, incluyendo en el mismo un recurso que ya está siendo utilizado para esa misma fecha.

RecursoYaExisteError Si se intenta agregar un recurso ya existente al calendario se lanzará esta excepción.

7. Diagramas de secuencia

En la Figura 2 se muestra el proceso de agregar un evento simple al calendario. Se puede ver como en el punto 2 se genera una nueva instancia de la clase EventoSimple, la cual en su creación agrega el evento para cada persona o recurso.

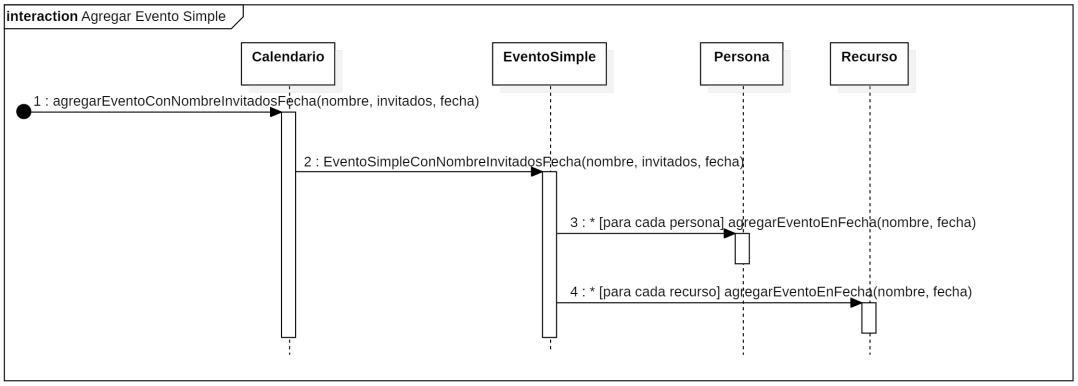

Figura 2: Agregar Evento Simple.

En la Figura 3 se puede ver que la diferencia de los eventos semanales con los simples radica en que se instancia un objeto de la clase EventoSemanal, la cual luego realiza una doble iteración, sobre cada invitado (persona o recurso) durante la cantidad de semanas pasadas como parámetro.

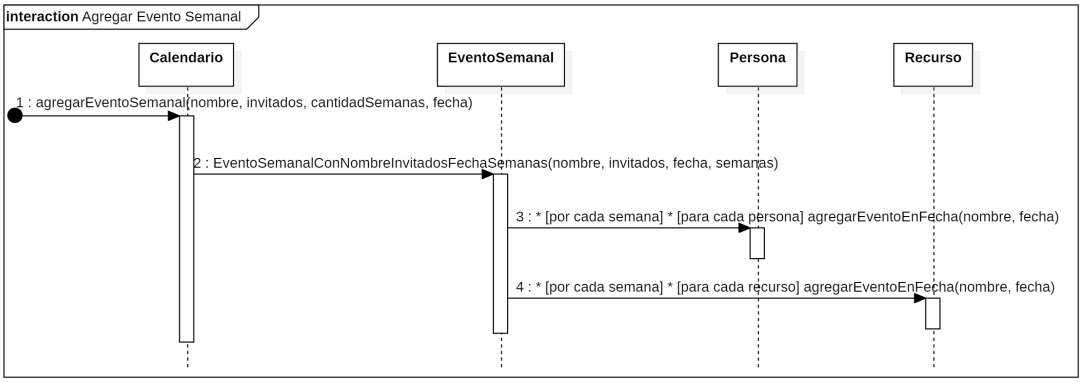

Figura 3: Agregar Evento Semanal.