

Conceptes Avançats de Programació

Práctica Q1 2014-2015

Blink

Ferrer González, Javier
García i Oto, Pol

2015-01-15

Contenidos

[Contenidos](#)

[Resumen enunciado](#)

[Conceptos Doctor Who](#)

[Conceptos práctica](#)

[Conceptos aprendidos](#)

[Smalltalk+GitHub](#)

[Punto de guardado de contexto](#)

[Descripción del concepto aprendido](#)

[Ejemplo del concepto](#)

[Pila de contextos %función%ContextsStack vacía](#)

[Descripción del concepto aprendido](#)

[Ejemplo del concepto](#)

[Tratamiento de la pila de contextos %función%ContextsStack](#)

[Descripción del concepto aprendido](#)

[Ejemplo del concepto](#)

[Metodología aplicada](#)

[Control de versiones](#)

[Tests](#)

[TDD](#)

[Entrega](#)

Resumen enunciado

Conceptos Doctor Who

- *Tardis*: Máquina del tiempo en la que viaja el Doctor Who.
 - Se pueden trasladar otros acompañantes y objetos a diferentes momentos del tiempo.
- *Senyors del temps*: Son los que dominan la tecnología de la Tardis. Originarios del planeta Gallifrey.
- *Àngels plorangers*: Se alimentan de la energía residual de los viajes en el tiempo. Para sobrevivir, envían personas a momentos aleatorios en el tiempo con sólo tocarlos. El Doctor Who tiene que ir arreglando los desperfectos que éstos provoquen.

Conceptos práctica

Tendremos una clase `Tardis` con 3 métodos principales:

- `#fita: anExpression`
 - Guarda el contexto actual para poder volver a él en un momento determinado vía llamada al método `#angelPloraner`.

- Retornar el valor de la evaluación de la expresión recibida como `anExpression`.
- `#angelPloraner: anExpression`
 - Guardar el contexto actual para poderlo restaurar vía llamada al método `#doctor`.
 - Establecer el contexto actual al contexto guardado por la última llamada a `#fita` no usada previamente (en caso de que exista).
 - Retornar el valor de la evaluación de la expresión recibida como `anExpression` con el contexto establecido en el paso anterior.
- `#doctor: anExpression`
 - Restaurar el contexto actual al siguiente guardado en la pila de contextos donde se guardan los contextos por las llamadas a `#angelPloraner` (en caso de que exista).
 - Retornar el valor de la evaluación de la expresión recibida como `anExpression` con el contexto establecido en el paso anterior.

Línea cronológica de ejecución del programa



Conceptos aprendidos

Smalltalk+GitHub

Tal y como se describe en el [apartado de Metodología aplicada > Control de versiones](#), por primera vez hemos jugado con el desarrollo en equipo con Smalltalk y, gracias a ello, nos hemos forzado a integrarlo con un repositorio en GitHub para poder compartir los avances entre nosotros.

Punto de guardado de contexto

Descripción del concepto aprendido

El punto o momento donde se guarda el contexto de ejecución es en `thisContext sender`, es decir, en la parte del código donde se llama a la función `#fita: anExpression`. Eso implica que la función `#fita` solamente tiene que guardar el contexto en la pila `fitesContextsStack` y evaluar la expresión `anExpression`.

En el caso de los contextos para los ángeles, es lo mismo pero con su propia pila (`angelsPlorannersContextsStack`) y en su propia llamada, es decir: la función `#angelPloraner: anExpression`. Ambas funciones siempre apilan el contexto de llamada cuando son llamadas, sea cual sea el caso de su ejecución.

Ejemplo del concepto

```
fita: anExpression
"a -fita- is a time mark available in order to return to it
in terms of application context"
    fitesContextsStack push: ( Continuation fromContext:
        thisContext sender ).

    ^ anExpression
```

Pila de contextos %función%ContextsStack vacía

Descripción del concepto aprendido

Cuando las pilas `fitesContextsStack` y `angelsPlorannersContextsStack` estan vacías la funciones `#angelPloraner` y `#doctor` evalúan su expresión y la devuelven. El estado vacío de estas pilas implica que no quedan fitas a las que saltar o destrozos de angeles por reparar, respectivamente para cada pila.

Ejemplo del concepto

```
"Firstly, check if the stack of angels ploranners is empty
in order to do not make any action and only evaluate the
received expression"
    ( angelsPlorannersContextsStack isEmpty )
        ifTrue: [
            ^ anExpression
        ].
```

Tratamiento de la pila de contextos

%función%ContextsStack

Descripción del concepto aprendido

Las pilas `fitesContextsStack` y `angelsPloranersContextsStack` se consultan en las funciones `#angelPloraner` y `#doctor` respectivamente. Se consulta el valor de encima de la pila y ese valor se desempila, para que no se pueda volver a ejecutar dicho contexto. Seguidamente se ejecuta la continuación (o salto temporal) hacia dicho contexto pero con la nueva expresión `anExpression`.

Ejemplo del concepto

```
"Also, recover the previously saved context and evaluate
the expression on it."
    lastFita := fitesContextsStack pop.
    lastFita value: anExpression.
```

Metodología aplicada

Control de versiones

Al querer usar Git como control de versiones, en un primer momento pensamos en usar `smalltalkhub.com` por su integración con Pharo, no obstante, por haberse convertido casi en un estándar de mercado, creemos que nos ha sido más fácil trabajar con GitHub por su soporte de aplicaciones con las que ya habíamos tratado (SourceTree) y por sus funcionalidades.

Para saber cómo integrarlo todo, nos hemos basado en el vídeo: "Pharo and Github using Sourcetree": <https://www.youtube.com/watch?v=n2WNYDtO0cE>

Descarga de los cambios realizados por el compañero:

- Pull de los cambios en el repositorio remoto (GitHub) desde SourceTree.
- Carga de los cambios en el paquete desde Pharo:
 - World -> Monticello Browser
 - Repositorio de tipo "filetree" donde está ubicado nuestro paquete -> Open
 - Selección del sub-paquete `Blink.package` -> Load

Actualización de los cambios realizados por nosotros para hacerlos disponibles:

- Guardado de los cambios en el paquete desde Pharo:
 - World -> Monticello Browser

- Repositorio de tipo "filetree" donde está ubicado nuestro paquete -> Open
- Selección del sub-paquete Blink -> Save
- Introducimos un "Log Message" descriptivo de nuestros cambios y lo copiamos al portapapeles -> Accept
- Cerramos ventana informativa de la versión de nuestro paquete
- Push de los cambios en el repositorio remoto (GitHub) desde SourceTree:
 - Botón Commit
 - Mover todos los cambios que nos interesa subir al staging area
 - Introducir "Commit Message" (pegar del portapapeles)
 - Botón Commit
 - Botón Push

Tests

Para la realización de los tests nos hemos basado en el principio de declaración semántica en cuanto a nomenclatura. Este principio se puede ver explicado en este post:

<http://codurance.com/2014/12/13/naming-test-classes-and-methods/>

No obstante, para poder seguir el estándar conocido en Pharo, las clases de tests sí que las hemos denominado con el sufijo "Test", ejemplo: `TardisTest`. Además, también hemos añadido el prefijo "test" a los métodos de testing unitario para, nuevamente, seguir lo que hemos visto que es común usar en Pharo, ejemplo:

```
testFitaShouldReturnExpressionEvaluation.
```

TDD

La metodología de desarrollo que hemos seguido ha sido la de "Test Driven Development". Para ello, primeramente hemos construido la lista de tests que queremos que nuestra aplicación implemente:

```
Tardis>>fita
```

- ✓ Retorna el resultado de la evaluación de la expresión recibida como parámetro -> `#testFitaShouldReturnExpressionEvaluation`

```
Tardis>>angelPloraner
```

- ✓ Al invocar a este método sin existir llamadas previas a `Tardis>>fita`, hace de función de identidad.
-> `#testAngelPloranerShouldReturnExpressionEvaluation`
- ✓ Al invocar con llamadas previas a `Tardis>>fita`, establece el contexto guardado en la última llamada a `#fita` y retorna el resultado de la evaluación de la expresión recibida como parámetro en el contexto de ésta. -> `#testAngelPloranerShoudGoBackInContext.`

- ✓ Al invocar con llamadas previas a `Tardis>>fita` y retornar al contexto de la llamada a `#fita`, ese contexto se elimina de la pila y no se vuelve a ejecutar.
-> `#testAngelPloranerShoudNotRepeatTheSameFita`
- ✓ Dos invocaciones consecutivas establecen contextos diferentes, yendo primero al contexto de la última llamada a `#fita`. -> `#testAngelPloranerShoudGoBackToTheLastFitaFirst`

`Tardis>>doctor`

- ✓ Al invocar a este método sin existir llamadas previas a `Tardis>>angelPloraner`, hace de función de identidad. -> `#testDoctorShouldReturnExpressionEvaluation`
- ✓ Al invocar con llamadas previas a `Tardis>>angelPloraner`, establece el contexto guardado en la última llamada a `#angelPloraner` y retorna el resultado de la evaluación de la expresión recibida como parámetro en el contexto de ésta. -> `#testDoctorShouldRestoreContext`

Una vez con esta lista de tests por implementar, los hemos ido implementando siguiendo un orden basado en la dificultad (de más fáciles a más difícil). De esta forma, las estructuras y demás conceptos complejos han ido saliendo a medida que íbamos avanzando.

Por último, hemos introducido el test del enunciado para comprobar su salida por el Transcript (`testDelivery`) junto con los otros dos tests adicionales recibidos por email (`#testAngelPloranerShouldNotPermanentlyReplaceTheSameFita` y `#testAngelPloranerShouldSaveTheContextWithoutPreviousFitaCalls`).

Entrega

Adjunto a este PDF, se encuentra el fichero `Blink.st`. Simplemente se debe importar a la imagen de Pharo por defecto y ya se pueden ejecutar todos los tests.

Adicionalmente, el repositorio de GitHub utilizado: <https://github.com/JavierCane/CAP-SmallTalk-Blink> se hará público a partir del último día para poder entregar la práctica (si no hay ningún problema) y, en caso de disponer de un usuario de GitHub, podríamos dar acceso antes para que fuera revisado.