Orientación a Objetos 1 - 2012
Práctica 4

En esta práctica: definirá nuevas clases, utilizará variables temporales, verificará el funcionamiento de los objetos a través del inspector y de una GUI, y realizará diagramas de clases.

Ejercicio 1: DanceCouple

Una DanceCouple (pareja de baile) está formada por dos robots, uno conocido como `he` y el otro conocido como `she`. Luego de crear una instancia de DanceCouple, se le deben pasar los dos robots como parámetros a través de un mensaje `#he:she:`.

1. Defina la clase `DanceCouple` (utilice el botón Agregar Clase), cree una instancia en el workspace (la parte del ambiente del robot donde se evalúan expresiones), y cree dos robots distintos para que sean los bailarines. Inspeccione el resultado.

2. Para hacer pruebas en el workspace con instancias de `DanceCouple`, deberá utilizar variables temporales. ¿En qué se diferencian las variables temporales de las variables de workspace? ¿Cómo se declaran?

Ejercicio 2: DanceCouple con comportamiento

En este ejercicio agregaremos comportamiento a `DanceCouple` con la siguiente definición de mensajes:

```
#reset
"Ambos bailarines se posicionan enfrentados a distancia 20 del home (uno mira al south y el otro al north)"


#tangoStep
"he hace un cuadrado de lado 100 rotado 45 grados, she hace un cuadrado de lado 50."


#doTheTango
"Los bailarines repiten tangoStep 5 veces, pero luego de cada una se corren 10 hacia el este."
```

1. Realice el diagrama clases.

2. Implemente los tres métodos (no olvide agregar los comentarios a cada método). Y pruebe valiéndose del Workspace y el Inspector que funcionen correctamente.

Orientación a Objetos 1 - 2012
Práctica 4

Ejercicio 3: ToDoItem

En este ejercicio abandonamos el mundo del robot y nos manejamos con el Browser del Sistema. Vamos a comenzar el desarrollo de una aplicación que continuaremos a lo largo de varias prácticas. Se trata de un manejador de tareas (que llamaremos To-Do List). Definimos el objeto tarea (ToDoItem) con los siguientes atributos: un texto que describe la tarea, una prioridad (1 a 10) y un estado (si fue completada o no).

Defina la clase `ToDoItem` en Smalltalk, en un package "ToDo-Model", con los siguientes mensajes:

```
#text
"Retorna el texto descriptivo de la tarea"

#text: aString
"Setea el texto descriptivo de la tarea"

#priority
"Retorna la prioridad"

#incrementPriority
"Incrementa la prioridad en uno. Si ya es 10, no hace nada"

#decrementPriority
"Decrementa la prioridad en uno. Si ya es 0, no hace nada"

#isCompleted
"Retorna true si la tarea ya fue completada, false en caso contrario"

#toggle
"Cambia el estado de completada a no completada y viceversa"

#initialize
"Inicializa el estado de las variables de instancia del ToDo Item. Luego de
la invocación el toDoItem debe tener como texto: 'Undefined ToDoItem', debe
estar en estado no completado y su prioridad debe ser 0."
```

Utilice el test provisto por la cátedra para comprobar que su implementación de `ToDoItem` es correcta

Ejercicio 4: Ventana del ToDoItem

En este ejercicio podrá interactuar con el ToDoItem a través de una GUI.

Orientación a Objetos 1 - 2012
Práctica 4

Figura 1: Ventana para interactuar con el ToDo item.

1. Cargue el parcel `Objetos - Practica 04` provisto en el material adicional de la práctica.

2. Para demostrar que su `ToDoItem` funciona, se lo vamos a pasar a una ventana como la que muestra la figura (una instancia de `ToDoChangerView`). Normalmente el usuario interactúa con los objetos a través de ventanas como esta. Si su `ToDoItem` implementa bien sus responsabilidades, la ventana debería funcionar sin problemas. Uno de los botones de la ventana le permite inspeccionar el objeto con el que la ventana trabaja.

3. Vea cómo el estado del objeto cambia a medida que usted interactúa con la ventana.

4. ¿Qué cree que sucede cuando oprime el botón “Inc”? ¿Y cuando oprime el botón “Dec”?

Para probar sus objetos en el workspace puede usar el siguiente script:

```
|anItem changer|
"creamos el modelo"
anItem := ToDoItem new.

"inicializamos el modelo"
anItem initialize.

"creamos el view"
changer := ToDoChangerView new.

"le damos un modelo al view"
changer item: anItem.

"desplegamos la view"
changer open.
```

Ejercicio 5: Evaluación de expresiones

¿Qué devuelve Smalltalk cuando se evalúan las siguientes expresiones? Realice el ejercicio en papel, tenga en cuenta el estado del ambiente producido por las evaluaciones previas.

```
x := 2 * 5 factorial
y := x + 1
|n| n := n+1
|n m| n := 4. m := 1. ^(n+m+x+y)
[x. y] value
```

Orientación a Objetos 1 - 2012
Práctica 4

```
duplicar := [:n | 2*n]
duplicar value:5
('objeto' at:2) isVowel ifTrue: [^2 * 3] ifFalse: [^4 * 2]
[x> 0] whileTrue: [x := x - 1]
5 timesRepeat: [x := x + y]
```

Luego de hacer las evaluaciones en papel, compruebe los resultados copiando las expresiones en un workspace y evaluándolas con VisualWorks

Ejercicio 6

Se tienen definidas en Smalltalk las clases `A`, `B` y `C` con el siguiente comportamiento

```
Object subclass: A
#print
      Transcript show: 'Soy la clase A'.
#hacerAlgo
      self print.

A subclass: B
#print
      Transcript show: 'Soy la clase B'.

B subclass: #C
#print
      Transcript show: 'Soy la clase C'.
#hacerAlgo
      super print.
```

¿Qué mensajes se imprimen en el Transcript, si dentro de una ventana de Workspace se evalúan las siguientes expresiones?

1. `b := B new. b hacerAlgo.`
2. `c := C new. c hacerAlgo.`