Trabajo Práctico 3
Programación Orientada a Objetos

Paradigmas de Lenguajes de Programación — 2º cuat. 2011

Fecha de entrega: 10 de Noviembre

En este trabajo práctico se implementará un sistema que permita el armado de cuestionarios y el llenado sucesivo de los mismos. En los cuestionarios de la vida se pueden encontrar distintos tipos de preguntas que esperan su correspondiente tipo de respuesta, por ejemplo: algún texto, número, fecha, si/no, direcciones, etc. No se van a abarcar todas estas opciones pero el sistema a armar dispondrá (como se verá a lo largo del enunciado) de un diseño que permitirá incorporar cuánto tipos de preguntas quieran.

Los cuestionarios dispondrán de un código de referencia que será usado para referenciarlos y usarlos para su confección o llenado. El llenado se podrá realizar mediante algunos cómodos métodos o gracias algún código que se encuentra más adelante, por medio de dialogos interactivos (si es que usan Pharo).

A pesar de la interfáz de usuario, como programadores, siempre ~~pueden~~ deberían testear.

Todas las implementaciones se deben realizar en una única categoria, por ejemplo Tp2011c2.

Ejercicio 1

Crear la clase ApplicationForm que representará un cuestionario. Las instancias de esta clase dispondrán de un código code que se indicará únicamente al momento de la creación y deberá ser único entre todas las instancias. Para esto resolver que:

- ApplicationForm >> code devuelva el código de la instancia.
- ApplicationForm class >> withCode: aCode devuelva, si existe, el cuestionario de código indicado. Sino nil.
- ApplicationForm class >> newWithCode: aCode cree un nuevo cuestionario con dicho código, pero falle (enviando el mensaje #error:) si el código ya está usado.

Nota: Sucesivas evaluaciones de ApplicationForm newWithCode: '42' deberían fallar.

Ejercicio 2

Agregar a ApplicationForm la capacidad de contener preguntas. Resolverlo usando alguna colección que respete el orden en que fueron insertados los elementos. Para esto resolver que:

- ApplicationForm >> addQuestion: aQuestion agregue la pregunta al cuestionario.
- ApplicationForm >> countOfQuestions devuelva cuántas preguntas tiene el cuestionario.
- ApplicationForm >> questionAt: index devuelva la pregunta de la posición indicada (el primer elemento está en la posición 1).

Ejercicio 3

Crear la clase Question que representará una pregunta. Debe disponer de una propiedad text que se inicializa únicamente al instanciar el objeto. Ésta contendrá el texto de la pregunta. Por ejemplo: (Question new: 'Nombre: ') text evalua a 'Nombre: '

Ejercicio 4

Crear la clase Answer que representará una respuesta a una pregunta. Debe disponer de una propiedad question. Por ejemplo: (Answer new question: q1) question evalua a q1.

Ejercicio 5

Crear la clase Submission que representará un cuestionario completo. Una instancia de esta clase tendrá una referencia al ApplicationForm y almacenará una lista con instancias de Answer. Para esto resolver que:

- Submission >> form y Submission >> form: anApplicationForm retorne y asigne el cuestionario correspondiente a la instancia.
- Submission >> addAnswer: anAnswer agregue la respuesta a la instancia.
- Submission >> answerFor: aQuestion retorne la respuesta que corresponde a la pregunta en cuestión.

Para soportar tipos de preguntas se deberán armar subclases de Question y Answer. Dependiendo del caso, en las subclases de Answer se almacenará la respuesta de una u otra manera. Ahora bien, como se va permitir completar un cuestionario, en definitiva, via texto cual consola interactiva dispondremos de otros elementos en nuestro modelo que estarán encargados del nexo entre esta interacción y el tipo de pregunta. Habrá un elemento por cada tipo de pregunta y serán subclases de una clase a crear llamada QuestionShellHelper.

Ejercicio 6

Crear la clase QuestionShellHelper que será la clase base para los *helpers* de los distintos tipos de preguntas. Implementar los siguientes items que serán de utilidad para las futuras subclases y el correcto funcionamiento del sistema. Los *helpers* contendrán métodos de clase solamente ya que se van a usar las clase como receptoras de los mensajes.

- Impedir la creación de instancias de QuestionShellHelper.
- Crear un método de clase QuestionShellHelper >> handle: aQuestion que deberá ser redefinido por las subclases. En las subclases este método deberá retornar true si el *helper* en cuestión es el relacionado con el tipo de pregunta del parámetro. Es decir, más adelante BooleanQuestionShellHelper handle: BooleanQuestion new deberá evaluar a true.
- Crear un método de clase QuestionShellHelper >> thatHandles: aQuestion que retorne la subclase de QuestionShellHelper para la cual el mensaje #handle: evalua a true con argumento aQuestion.
- Crear un método de clase QuestionShellHelper >> print: aQuestion que retorne el texto de la pregunta. Este método será usado para mostrar las preguntas y que cada tipo pueda por ejemplo agregar texto al mostrarse (ej: [Y/N] para las preguntas si/no).

Para crear un nuevo tipo de pregunta, por ejemplo para preguntas si/no será necesarias las siguientes clases: BooleanQuestion, BooleanAnswer y BooleanQuestionShellHelper cada con las responsabilidad que indica su sufijo. Como la relación entre estás clases ya está fija se dispondrá de

un método que permitirá crear todos estas clases y ya generará el código mínimo para que operen. Luego el programador deberá adaptar el código para usos específicos, pero de esta manera no se comenzará de cero en cada caso.

Ejercicio 7

Se pide implementar un método de clase Question >> buildQuestionType: prefix que realice las siguientes tareas:

- Crear la clase [prefix]Question subclase de Question
- Crear la clase [prefix]Answer subclase de Answer
- Crear la clase [prefix]QuestionShellHelper subclase de QuestionShellHelper
- Agregar un método de instancia [prefix]Question >> buildAnswer que retorne una nueva instancia de [prefix]Answer relacionada con la pregunta [prefix]Question que recibió el mensaje.
- Agregar un método de clase [prefix]QuestionShellHelper >> parse: theUserInput for: aQuestion que use el mensaje #buildAnswer del item anterior para retornar una instancia de Answer vinculada a aQuestion. Asumir que aQuestion será del tipo [prefix]Question.

Ejercicio 8

Usar el mensaje #buildQuestionType: con prefijo 'Text' para representar preguntas de texto. Además completar la implementación realizando los siguientes items:

- Agregar a TextAnswer los mensajes #value y #value:
- Cambiar el método TextQuestionShellHelper >> parse: theUserInput for: aQuestion para que retorne un TextAnswer con el valor indicado por theUserInput. Pero en los casos donde este argumento sea nil o vacío debe generar una excepción (usando el mensaje #error:).

Ejercicio 9

Usar el mensaje #buildQuestionType: con prefijo 'Boolean' para representar preguntas si/no. Además completar la implementación relizando los siguientes items:

- Agregar a BooleanAnswer los mensajes #value y #value:
- Cambiar el método BooleanQuestionShellHelper >> parse: theUserInput for: aQuestion para que retorne un BooleanAnswer con el valor indicado por theUserInput: 'y' o 'yes' será true y 'n' o 'no' será false, en cualquier otro caso se debe generar una excepción (usando el mensaje #error:).
- Sobrescribir el método BooleanQuestionShellHelper >> print: aQuestion para que al resultado de la clase base le concate el texto ' [Y/Yes/N/No]'.

Se incorporará una clase que tendrá la responsabilidad de, para un cuestionario dado, guiar el proceso de responder todas las preguntas. Esta clase permitirá obtener una instancia de Submission con dichas respuestas. Esta clase implementará una especie cursor que marcará la pregunta actual.

Ejercicio 10

- Crear una clase ShellHelper que deberá recibir al momento de creación un ApplicationForm.
- Agregar el método ShellHelper >> start que inicializa un llenado del cuestionar. Apunta el cursor a la primer pregunta del cuestionario y retorna el texto que se debe mostrar al usuario (mensaje #text: del QuestionShellHelper que corresponda).
- Agregar el método ShellHelper >> receive: theUserInput que usa el QuestionShellHelper de la pregunta actual para obtener un Answer y agregarlo al Submission. En caso de éxito se avanza a la siguiente pregunta y se retorna el texto a mostrar al usuario. En caso de que theUserInput no se corresponda con una respuesta, no se avanzará de pregunta y se retornará el texto de la misma a mostrar al usuario (ver mensaje #on:do: de los bloques).
- Agregar el método ShellHelper >> submission que retorne el Submission con las respuestas completadas.
- Agregar el método ApplicationForm >> buildShellHelper que retorne un ShellHelper inicializado para el receptor del mensaje.

Para poder probar de forma interactiva el llenado de un cuestionario se puede agregar el siguiente método:

```
ShellHelper >> startUI
  | output input |
  output := self start.
  [ output isNil ] whileFalse: [
    input := UIManager default request: output.
    output := self receive: input.
  ].
```

Para poder mostrar un Submission en pantalla se sugiere sobrescribir los mensajes de instancia #printOn: que reciben un stream en donde se va a imprimir el contenido del receptor. Los streams cuentan con un mensaje #nextPutAll: que reciben un String. Luego, a cualquier objeto se le puede enviar el mensaje #asString para ver su representación en String.

Ejercicio 11

Definir #printOn: para Submission, TextAnswer y BooleanAnswer.

Pautas de entrega

Se debe entregar un archivo con la implementación de las clases pedidas (un `.cs` si usan `Pharo` o `Squeak`, un `.st` si usan `VisualWorks`). No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Sí deben entregar una versión impresa (legible) del código, e indicar qué intérprete utilizaron para desarrollarlo.

Los objetivos a evaluar en la implementación de las clases son:

- Corrección.
- Declaratividad.
- Manejo adecuado de conceptos del paradigma y herramientas particulares de Smalltalk.