Ejercicio 3

Lenguajes de Dominio Específico Internos

Inicio: 13 de Febrero
Entrega: 24 de Febrero a las 11:00 a.m.

El objetivo de este ejercicio es familiarizarse con la creación de lenguajes de dominio específico internos en Ruby.

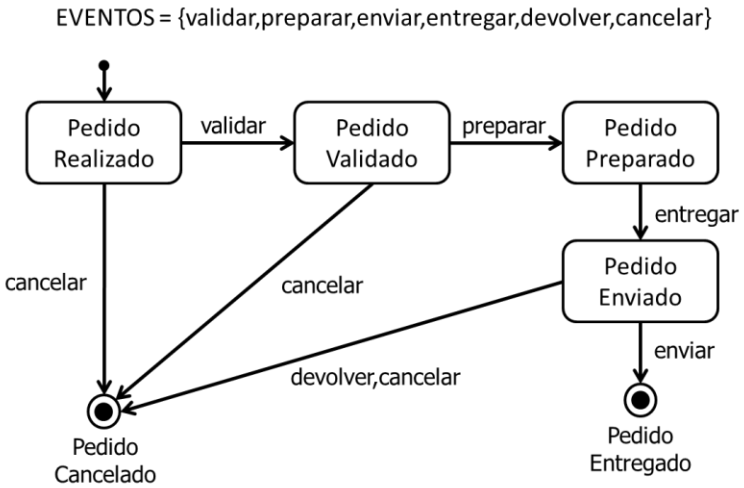
Apartado 1: Máquinas de estados (5 puntos)

1.a) 3.5 puntos.
Se pide crear un lenguaje de dominio específico para definir máquinas de estados. El lenguaje se construirá usando el primero de los enfoques vistos en clase: creando una clase de contexto a la que se añadirá un método de clase por cada *keyword* del lenguaje. De este modo, una máquina de estados se definirá mediante una subclase de la clase de contexto.

Una máquina de estados define estados conectados mediante transiciones, y admite (reacciona a) una lista de eventos que hay que definir junto con la máquina. Las transiciones pueden tener asociados uno o más de estos eventos. Los estados tienen un nombre, y pueden definir cero o más transiciones de salida. A un estado pueden llegar cero o más transiciones. Una máquina de estados puede tener un estado inicial, y cero o más estados finales.

La sintaxis concreta del lenguaje pedido queda a la elección del alumno. Se valorará que dicha sintaxis sea intuitiva y fácil de utilizar. En las transparencias de clase puedes encontrar un lenguaje para definir máquinas de estados construido en Ruby, similar al que se pide en este apartado, que (opcionalmente) puedes usar como guía.

Utiliza tu lenguaje para definir la siguiente máquina de estados, y entrégala junto con la definición de tu lenguaje.



1.b) 1.5 puntos.
Construye un simulador para tus máquinas de estados. El simulador recibirá una máquina de estados y una secuencia de eventos. La simulación comenzará en el estado marcado como estado inicial. En cada paso de la simulación, se consumirá un evento de la secuencia. Si el estado actual **no** tiene ninguna transición de salida anotada con el evento, se pasará a tratar el siguiente evento de la secuencia. Si el estado actual **sí** tiene una transición de salida anotada con el evento, el estado destino de la transición pasará a ser el estado actual. El simulador irá imprimiendo por pantalla los estados por los que va pasando. La simulación terminará cuando se llegue a un estado final, o cuando se hayan consumido todos los eventos de la lista sin llegar a un estado final.

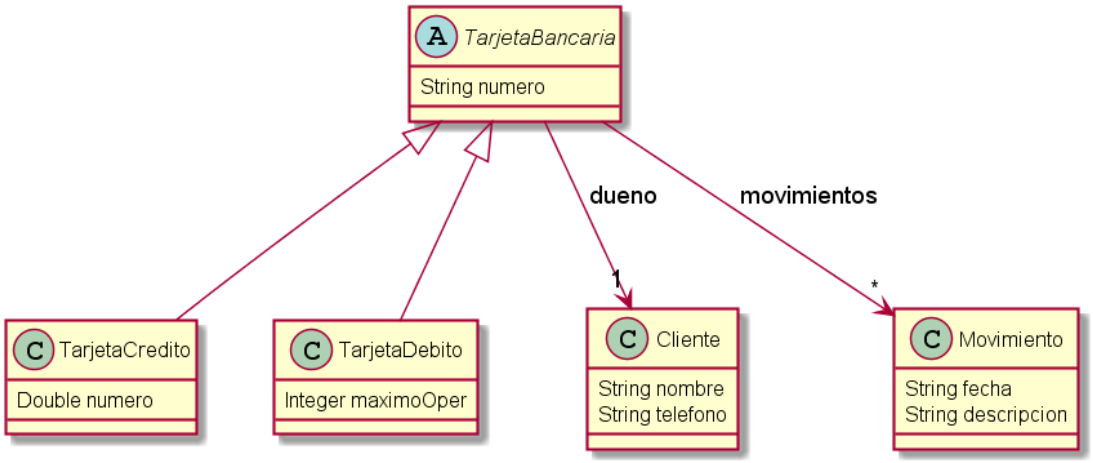
Apartado 2: Diagramas de clases (5 puntos)

2.a) 3.5 puntos.
Se pide crear un lenguaje de dominio específico para definir diagramas de clases. El lenguaje se construirá usando el segundo enfoque visto en clase, basado en el uso de `instance_eval`.

Un diagrama de clases tiene un nombre y está formado por clases. Cada clase tiene un nombre, puede ser abstracta o concreta, puede ser subclase de otra, y define un número arbitrario de atributos y referencias (cero o más). Cada atributo tiene un nombre, un tipo, y una cardinalidad (1 o *, siendo 1 por defecto). Cada referencia tiene un nombre, un tipo (que debe ser otra clase del mismo diagrama), y una cardinalidad (1 o *, siendo 1 por defecto).

La sintaxis concreta del lenguaje pedido queda a la elección del alumno. Se valorará que dicha sintaxis sea intuitiva y fácil de utilizar.

Utiliza tu lenguaje para definir el siguiente diagrama de clases, y entrégalo junto con la definición de tu lenguaje.



2.b) 1.5 puntos.
En el apartado anterior has construido un lenguaje para describir diagramas de clases de manera textual. En este apartado se pide generar una visualización gráfica a partir de esta descripción textual. Para ello usaremos `PlantUML`, un software de libre distribución que permite visualizar diagramas de clases descritos mediante ficheros de texto cuyo formato puedes consultar en http://plantuml.sourceforge.net/classes.html. Por tanto, lo que debes hacer es construir un generador que, dado un diagrama de clases descrito con tu lenguaje, genere un fichero de texto adecuado para su visualización con `PlantUML`. A continuación puedes ver el fichero `PlantUML` que debería generarse para un diagrama como el anterior:

```
@startuml
scale 900 width
abstract class TarjetaBancaria {
  String numero
}
class TarjetaCredito {
  Double numero
}
class TarjetaDebito {
  Integer maximoOper
}
class Cliente {
  String nombre
  String telefono
}
class Movimiento {
  String fecha
```

```
  String descripcion
}
TarjetaBancaria <|-- TarjetaCredito
TarjetaBancaria <|-- TarjetaDebito
TarjetaBancaria --> "1" Cliente : dueno
TarjetaBancaria --> "*" Movimiento : movimientos
@enduml
```

Estos son los pasos para instalarte `PlantUML` y probar que los ficheros que generas son correctos:

1. Instala `GraphViz`: http://www.graphviz.org/Download..php
2. Descarga `PlantUML` (es un fichero `jar`): http://plantuml.sourceforge.net/download.html
3. Copia el jar en el directorio donde tengas el fichero `PlantUML` a visualizar
4. En la consola de comandos, teclea: `java –jar plantuml.jar <nombre-fichero>`
5. Se generará un fichero `png` con la visualización correspondiente

Normas de Entrega

El código se comprimirá en un único fichero zip/rar, que se entregará a través de moodle antes de la fecha de entrega indicada en la cabecera del enunciado.

El nombre del fichero deberá incluir el nombre y apellido del estudiante.