CS 535 Object-Oriented Programming & Design
Fall Semester, 2013
Doc 14 Some Building Blocks
Oct 22 2013

Copyright ©, All rights reserved. 2013 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/openpub/) license defines the copyright on this document.
References

Example - Turtle Graphics

Turtle Graphics - used help teach programming

Program Turtle to
Move across screen
Draw patterns

Operations
move
turn
penUp
penDown

Sample Program
penDown
move 5
turn 90 left
move 10
turn 90 left
move 5
turn 90 left
move 10
Assume We Have Turtle class

Displays small turtle on screen

Responds to basic command
  Move
  Turn
  penUp
  penDown

Draws line when pen is down and Turtle moves
How to parse Turtle Program

As String

lines := turtleProgram tokensBasedOn: Character cr.
lines do: [:aLine | | command amount direction |
    parts := aLine tokensBasedOn: Character space.
    command := parts first.
    command = 'move' | 'turn'
        ifTrue: [ifTrue: [
            amount := (parts at: 2) asNumber.
            command = 'turn' ifTrue: [
                direction := parts last.]].
        command = 'turn' ifTrue: [turtle turn: amount direction: direction].
    command = 'move' ifTrue: [turtle move: amount].
    command = 'penDown' ifTrue: [turtle penDown].
    command = 'penUp' ifTrue: [turtle penUp].

turtleProgram := 'penDown
move 5
turn 90 left
move 10
turn 90 left
move 5
turn 90 left
move 10'.
New Commands

color
   One argument - a color

circle
   One argument - radius
Building Block - TurtleStream

Possible Operations

nextToken
nextCommand
nextArgument
Executing Turtle Program/Command

TurtleInterpreter class
Responsibilities
   Analyze and execute turtle programs

Collaborations
   Turtle
   TurtleStream

Turtle class
Responsibilities
   Draw on screen
   Perform operations
TurtleInterpreter

Instance variables
  turtle - instance of Turtle
  source - instance of TurtleStream

TurtleInterpreter>>on: aProgramString
  Initializes turtle and source
  
  turtle := Turtle new.
  source := TurtleStream on: aProgramString

TurtleInterpreter>>evaluate
  [source atEnd]
  whileFalse: [self evaluateCommand]
Simple Solution

TurtleInterpreter>>evaluateCommand
| command |
command := source nextCommand.
command asLowercase = 'penUp'
  ifTrue: [^self penUp].
command asLowercase = 'move'
  ifTrue: [^self move].
command asLowercase = 'turn'
  ifTrue: [^self turn].
etc.

TurtleInterpreter>>move
| distance |
distance := source nextArgument.
turtle move: distance

TurtleInterpreter>>turn
| amount direction |
amount := source nextArgument.
direction := source nextArgument.
turtle
  turn: amount
direction: direction

TurtleInterpreter>>penUp
turtle penUp
What Have We Gained?
Bigger Building Blocks - TurtleCommands

Read line of program

Give line of program to TurtleCommand class

TurtleCommand parses line

Some methods

  isMove
  isTurn
  amount
  direction
Command Solution

```smalltalk
TurtleInterpreter>>evaluateCommand
    | command |
    line := source nextLine.
    command := TurtleCommand on: line
    command isPenUp
        ifTrue:[^self penUp].
    command isMove
        ifTrue:[^self move: command].
    command isTurn
        ifTrue:[^self turn: command].
    etc.
```

```smalltalk
TurtleInterpreter>>move: command
turtle move: command amount.
```

```smalltalk
TurtleInterpreter>>turn: command
turtle
    turn: command amount
direction: command direction
```

```smalltalk
TurtleInterpreter>>penUp
turtle penUp
```
What Have We Gained?

Who knows the syntax for command?

Who has to change if syntax changes
Command Solution - Improved

TurtleInterpreter>>evaluateCommand
    | command |
    command := TurtleCommand fromStream: source
    command isPenUp
        ifTrue: [^self penUp].
    command isMove
        ifTrue: [^self move: command].
    command isTurn
        ifTrue: [^self turn: command].
    etc.

Only TurtleCommand knows program syntax
A class should hide a design decision

Turtle Command now hides all of the syntax of program

Syntax change change - rest of program does not have to know
Smarter Commands

Let the commands tell the turtle what to do
TurtleInterpreter

TurtleInterpreter class>>on: aProgramString
^super new on: aProgramString

TurtleInterpreter>>on: aProgramString
turtle := Turtle new.
source := ReadStream on: aProgramString

TurtleInterpreter>>evaluate
[source atEnd]
whileFalse: [self evaluateCommand]

TurtleInterpreter>>evaluateCommand
| command |
command execute.
## TurtleCommand

<table>
<thead>
<tr>
<th>TurtleCommand&gt;&gt;execute</th>
<th>TuttleCommand instance variables</th>
</tr>
</thead>
</table>
| self isPenUp
  ifTrue: [^ turtle penUp]. | turtle |
| self isMove
  ifTrue: [^ turtle move: amount]. | command |
| self isTurn
  ifTrue: [^ turtle
    turn: amount
    direction: direction]. | amount |
| etc. | direction |
| | programSource |
What Have We Gained?
Undo

Since command know what it did

It knows enough to undo it
   Need eraser to undo drawing

Can save commands in stack for multiple undo
Macros

Can group commands into compound command to make new commands

Square
  move 100
  turn 90 left
  move 100
  turn 90
  move 100
  turn 90
  move 100
Changing Program Syntax

Some environments provide GUI elements to create Turtle program

GUI element for move can produce Move commend

GUI creates list of command object to run
Command Objects

Create a Command Class for each command in language

Command knows how to
  Execute the command
  Undo the command

Allows stepping through the program and undoing operations
MoveCommand

Smalltalk defineClass: #MoveCommand
   superclass: #{Core.TurtleCommand}
   instanceVariableNames: 'turtle amount '

MoveCommand>>execute
   turtle move: amount

MoveCommand>>undo
   turtle
   left: 180;
   move: amount;
   left: 180
Back to Turtle

Sample Program
penDown
move 5
turn 90 left
move 10
turn 90 left
move 5
turn 90 left
move 10

New Syntax
penDown
move: 5
turnLeft: 90
move: 10
turnLeft: 90
move: 5
turnLeft: 90
move: 10

| turtle |
turtle := Turtle new.
turtle
  penDown;
  move: 5;
  turnLeft: 90;
  move: 10;
  turnLeft: 90;
  move: 5;
  turnLeft: 90;
  move: 10

If we have control over
syntax create so we can
use compiler evaluate (Do it)

Read the program, transform the
string into complete Smalltalk code
and use compiler evaluate: (do it)

Of course we could just require the user to enter the text on the right, which would make our job easier.
Domain-Specific language (DSL)

Language dedicated to a particular problem domain

Examples

UNIX shell scripts
ColdFusion Markup Language
FilterMeister
  For writing Photoshop plugins
Some Advantages

Program written in words from the domain
  Domain experts can understand, validate, modify, and write programs

Self-documenting code

Enhance quality, productivity, reliability, maintainability, portability and reusability

Domain-specific languages allow validation at the domain level