CS 635 Advanced OO Design and Programming
Spring Semester, 2015
Assignment 3

San Diego State University -- This page last updated 3/12/15

Assignment 3
Turtle Programs

We are going to implement part of a program to implement turtle graphics. See the wikipedia article on turtle graphics (http://en.wikipedia.org/wiki/Turtle_graphics) if you are unfamiliar with turtle graphics. We will support the turtle moving, turning, and raising/lowering a pen to draw. We will not support changing the color of the pen. The goal is to create a small language that kids can use to program the turtle. We will not implement the graphics part of the program. We are just interested in the section of the program that interprets the language. The language is given below.

SDSU Mini Turtle Language

The turtle language consists of the following commands.

penUp
: When the pen is up and the turtle moves nothing is drawn.

penDown
: When the pen is down and the turtle moves a black line is drawn

move X
: Move the turtle X units in the current direction. X is a positive integer.

turn X
: Turn the direction of the turtle X degrees.

repeat n
:
```
statement1
statement2
...
statementJ
```
end
: n is an integer. The statements inside the repeat block are any legal statement in the language. the repeat statement has the obvious semantics.

$K = Y
: Y is an integer. The variable $K can be used in move, turn and repeat commands.

When a program starts the turtle is in the middle of the screen headed horizontally right (zero degrees) and the pen is up. Here are two sample turtle programs which draw a square.

```
$length = 10
penDown
move $length
turn 90
move $length
turn 90
move $length
turn 90
move $length
```

Nearly the same program using the repeat statement.

```
$length = 10
penDown
repeat 4
        move $length
        turn 90
end
```

Turtle Class

The full program will have a Turtle class that draws on the screen. The Turtle class has fields to hold the current direction (in degrees) of the turtle and the current location (X & Y coordinates). The class has the following methods.

void move(int distance)
Move the turtle distance units in the current direction and draw on the screen if the pen is down.

void turn(int degrees)
Add “degrees” to the current heading of the turtle.

void penUp()
Lift the pen up.

void penDown()
Put the pen down.

boolean isPenUp()
Return true if pen is up, false if the pen is down.

int direction()
Returns the current direction of the turtle.

Point location()
Returns the current location of the turtle.

Our goal to is to be able to read and execute a turtle program. The focus is not on the graphics part. So we will create a mock Turtle class that has the same fields and methods as the turtle class but will not draw on the screen. All the methods perform the same operation as the real Turtle class except for the move method. This will allow us to test our code without having to draw on the screen.

void move(int distance)
Move the turtle distance units in the current direction and updates the field(s) indicating the current location of the turtle. The method does not draw on the screen.

Geometry & Computing New Location

The turtle starts at the location (0, 0), which if we were going to draw on the screen would be in the center of the screen. The turtle also starts with a direction of zero degrees with the pen down. A direction of 0 degrees is to the right parallel to the X-axis. A direction of 90 degrees is straight up parallel to the Y-axis. A direction of -270 degrees is also the same as a direction of 90 degrees.

Now if the turtle is at (10,20), heading in a direction of 30 degrees and travels 15 units how do we compute where it ends up? First we have to convert the degrees to radians via the formula:

radians = degrees * pi / 180

Then we can get the deltaX and deltaY using:

deltaX = cosine(radians) * distance

deltaY = sine(radians) distance

So in our example we get

x = 10
y = 20
degrees = 30
distance = 15
radians = pi*degrees/180 => 0.5236
deltaY = sin(radians) * distance => 7.5
deltaX = cos(radians) * distance => 12.9904
newX = x + deltaX => 22.9904
newY = y + deltaY => 27.5

So the turtle will end up at (22.99, 27.5)

The Assignment

1. Using the interpreter pattern we want to be able to read a text file containing a turtle program and represent the program as abstract syntax trees which we can execute. So if our file contained the following program after Reading and executing the program the turtle would end up at (22.99, 27.5).

   move 10

```
turn 90
move 20
turn -60
move 15
```

2. The problem with using the abstract syntax trees to execute the program is that the program cannot be stopped or reversed. It is often useful to step through a program. So use the visitor pattern so we can send a visitor to the parts of the program. After which the visitor can return a list of commands. Each command will perform one operation in the program. Using the commands we can then execute the program step by step. The commands also need to allow us to undo an operation. This will allow us to undo and redo parts of our program.

3. Create a second visitor that will compute the total distance traveled by the turtle. So for example given the program below the visitor would compute the distance of 20 even though the turtle would end up where it started.

```
move 10
turn 180
move 10
```

Grading

Item	Points
Working Code	10
Unit Tests	10
Proper implementation of Patterns	20 per Pattern (60 points total)
Quality of Code	10