CSC-240 Lesson 5: Arrays and ArrayLists

Assignment 5

This assignment is based on Exercise 7.21 (Turtle Graphics) from the textbook (but is not identical). Please follow the instructions below.

This is a more challenging and mainly a longer exercise than the previous exercises. The steps involved are not particualry complex, but there are a number of separate steps involved in getting to a working solution.

Take an early look at what this exercise requires, and be prepared to ask questions while there is still time to get answers. *Note that this program is a command-line style program, not a GUI program.*

Please submit at least the following file for the assignment (together with any additional Java source files you find necessary):

- `TurtleGraphics.java`, which implements the **TurtleGraphics** class with just the two methods, *enterCommands*, and *executeCommands*, as described below. Note that the supplied test program (**TurtleGraphicsTest**) expects to find the **TurtleGraphics** class in the same package as the test program.)

Note: that this assignment requires only that this one **TurtleGraphics** class (plus any other classes you might create to support it) to be submitted. There is a provided test program and a data package (`edu.frontrange.csc240.a5.data`) that supplies all the needed inputs to test and verify the implementation of this class. The same test program will be used to evaluate the submitted work. The test program is described below. Please do not submit the test program or its data package together with the classes you have written to the Assignments Submission Folder.

A "skeleton" (*i.e.*, initial but incomplete) version of the class **TurtleGraphics**, and the test program, **TurtleGraphicsTest**, are supplied in this zipfile. There are, also, in this file the test data used by the **TurtleGraphicsTest** program, a **Debug** class that you may want to use, as well as all the accompanying Javadoc. The content of this file is discussed in more detail in the topic "About the Assignment."

The Scenario

The concept of "turtle graphics" is based on an invention made in 1967, in a language called "Logo" (https://en.wikipedia.org/wiki/Logo_%28programming_language%29). The "turtle" is a mechanical robot that walks around the floor of a room. It carries a "pen." If the pen is down, it marks a track on the floor when it moves. If the pen is up, it just moves, leaving no mark. The robot is controlled by a "turtle program," which is a sequence of commands, each controlling an action of the turtle. In this exercise, the commands are represented by integer values. For this program, the floor is assumed to be 20 by 20 units.

There are two phases to the processing of a turtle program, just as there is with any programming language.

- The first phase is reading the program, checking it for correct form and syntax, and storing it in some defined but convenient "compiled" form for later execution.
- The second phase is the execution, where the "compiled" form is read and the actions indicated are actually taken.

The class to be written, **TurtleGraphics**, implements both of these phases, by means of its defined two methods.

The two methods of the TurtleGraphics *class*

The definition of the commands is reproduced here:

Table of Commands

Command	Meaning
1	Pen up
2	Pen down
3	Turn right
4	Turn left
5,d	Move in the current direction "d" spaces
6	Display the current state of the floor
9	Indicates the end of the program

	(nothing after this command is part of the program, and is to be ignored)

There are two **public** methods of the **TurtleGraphics** class,

- *enterCommands*, which is the "compiler," and reads the commands of the program and decides whether they are validly formed or not, and the
- *executeCommands* method, that actually carries out the steps described under ***Meaning*** in the above table.

enterCommands *method*

The *enterCommands* method acts as the "compiler" for a turtle program. Each call to this method enters a new program, overwriting any previous program. The signature and return value of the method is:

```
public List<String> enterCommands(Scanner source)
```

The **Scanner** parameter passed to this method is the source of the "turtle program." The integer values representing the commands may be obtained from this **Scanner** instance as has been done for any **Scanner** seen in previous assignments.

When passed to the *enterCommands* method, the method should analyze the values delivered by the **Scanner**.

The tasks to be carried out by the *enterCommands* method are:

- to see that the input consists only of the recognizable and valid commands (see the table above);
- that a "move" command is followed by another integer value (which represent a number of steps of movement); after a "move" command, there must be another integer or the program has an error;
- to check the program is terminated properly. The program must end with the proper end-of-program command. (Any input beyond this is "junk" and must be ignored)
- the method should create informative messages; in particular, there should be error messages for any problem recognized in the program (see the examples in the "About the Assignment" topic). The messages (which are Strings) must be added to a **List**, and this **List** is the value to be returned by the method. (The test program will take care of actually printing the messages).
- The method may also, if you wish, echo the values input as a "listing" of the program, by adding a representation of those value to the same **List** (as is done in the samples shown in the *About the Assignment* topic.
- the *enterCommands* method should transform the program into a more convenient representation, *i.e.*, more convenient than having to re-analyze the commands read from the **Scanner**, and at the same time store this more convenient representation in an internal structure ready for execution (see the *executeCommands* method below).

Note that the **TurtleGraphics** class might not be instantiated again for each new program (it does not have a constructor with any defined requirements). This requires that the *enterCommands* method must be "serially reusable," meaning that it should reset all the state of the instance (the variables) that need to be in an initial state in order to process a program.

executeCommands *method*

The program that was entered may be "executed." The program is executed by calling the *executeCommands* method, which is also done by the test program (do not call the *executeCommands* method from the *enterCommands* method). The program created in one instance of the **TurtleGraphics** class may be executed as often as desired. The signature and return value of the method is:

```
public boolean executeCommands()
```

This method does not need any parameters, as there is no further input required and it will execute the program previously processed and stored by the *enterCommands* method. Executing the program actually does the work to move the turtle over the "floor," and as it moves, if its pen is down, it makes a mark, otherwise it just continues moving and leaves whatever is already on the floor undisturbed and unchanged. The *executeCommands* returns a value of **false** if there is no valid program to execute.

The floor and how the turtle moves

Assume that the turtle drawing is in a box that has a floor 20 units by 20 units in size (as mentioned above). You yourself may specify what you want the behavior to be when the turtle gets to the edge of the box (since this is not mentioned in the problem description—your choice—but you ***must*** write some commentary to say what you have decided the turtle will do when reaching a boundary, so it may be determined whether you have done it as you intend).

When a "display" (that is, a "6") command is found in the program, the current state of the floor is rendered to the standard output. (A single "*" is suggested in the textbook as a character to represent a mark on the floor, but two characters such as "* "—an asterisk followed by a space—gives a much better appearance.)

The turtle marks the floor only when it is asked to move (not when the pen is put down). When the turtle starts a move, it marks the current position where it is, and then moves one position in the required direction (but does not mark the floor again). This is repeated for every position it moves. That means, if the turtle is required, say, to move 3 units, it will mark the current position at the start of the move, and also will mark the next two positions, but will be left over the position after that without marking it. (So, with a move of 3 units, only three positions are marked, not four.)

Format of the Turtle Program

Spaces, end-of-line, blank lines have no significance in a turtle program (just like in a Java program) other than to separate the numbers that make up the commands and distances. A single comma immediately after a number is like a single space. A comma may be replaced by a single space, and the program will be unchanged, and will produce exactly the same drawing. The single comma is allowed in a turtle program just for visual effect, intended to make it easier to read the program when a command has a related move value. It does not have to be there, so need not be checked. All of the five following one-line programs (that draw the box shown in the textbook) are equivalent.

```
2 5,12 3 5,12 3 5,12 3 5,12 1 6 9
2   5 12 3 5 12    3 5 12 3 5,12  1  6 9
2  5  12  3  5  12  3  5  12  3  5  12  1  6  9
2, 5,12 3, 5,12 3, 5,12 3, 5,12 1, 6, 9,
2, 5  12 3,  5, 12,  3 5, 12,  3 5,12 1 6     9
```

Initial conditions

When the turtle starts executing a program, it is to be at the point 0,0 (top left-hand corner) of the floor. When the turtle moves, it moves in the direction it is currently facing. The initial direction is in the positive *x* direction, that is, to the east. Also, the pen must be initially up, so that it makes no mark on the floor.

Test Program

The supplied test program tests drawing samples additional to just the textbook example. These test programs are shown in the *About the Assignment* topic.

For more information about test program supplied, also see the topic "About the Assignment." If you wish, you may write additional test programs and add them to the list of programs. You will note that the test program uses some inputs that are intended to be incorrect, in order to assess that the **TurtleGraphics** class deals correctly with such problems.

> The following *About the Assignment* topic contains further explanation, directions, and hints about this assignment. Please read that topic before starting this assignment.

Important

Please follow the directions of Programming Assignment Identification in submitting your programming solutions.

If you have any questions or concerns about the Assignment or the *About the Assignment* topic, please use the Lesson Question discussion topic, or send a message by the D2L Internal Messaging system if you prefer.

Note: you may submit to the Assignment Submission Folders as many times and as often as you wish, up to the deadline time. Each submission is tagged with the date/time, and so each submission remains separate and distinct. Unless you leave instructions to the contrary, only the most recent of each file with the same name will be viewed for the purposes of grading. Details may be found in the topic *How To Submit and Get Feedback on Assignments* in the *How To* module.

Messages that accompany Assignment Submissions are read, and responded to, ***only*** when assignment submissions are graded (which is after the Assignment Submission Folder closing date/time). If you have a comment or question about an assignment, or a request for assistance, that needs an earlier response, then that comment, question or request should be made or asked *via* an Internal Message or the Discussion board, as these are usually read and answered every day.