CO882 Assignment 1 (Version 2016.11.15.01)

Introduction

This assignment is designed to assess your understanding of code quality issues, such as refactoring, coupling and cohesion, as well as providing further practical experience working with a substantial piece of program code.

Date set: 15th November 2016

Date due: 23:55 7th December 2016

Weighting: 35% of the coursework mark.

The task has been broken down into two stages to guide you in its development.

The Task

The task is to adapt the scribble project from chapter 6 with the addition of a text-based user interface to create drawings:

https://www.cs.kent.ac.uk/~djb/co882/LOCAL-ONLY/assign1.zip

You are provided with the source code of three classes – Pen, Canvas and InputReader – but the BlueJ editor for these classes is currently showing the *Documentation* view of them rather than the *Source Code* view. The *Documentation* view is simply the javadoc documentation for the classes that has been generated from the comments and method headers that are in the source code. The idea is that it should be easy to use these classes without needing to see details of the source code. You can flip the views by using the drop-down menu in the top-right corner of the editor window. There should be no need to change the source code of any of these classes. However, if you feel that you need to make changes to them then ask me whether this is acceptable or appropriate before doing so.

Your task is to complete the code currently in the DrawingTool class so that users can interactively create and manipulate shapes to create pictures. You may add any further classes that you feel are necessary to fulfil the requirements and you may modify the existing code of the DrawingTool class in any way that you wish.

Assessment criteria

The following sections give details of the tasks that are required. In completing these, please bear in mind that your work will be assessed primarily against the following criteria:

- Accurate implementation of the drawing commands.
- Adherence to appropriate principles of cohesion and coupling.
- Appropriate use of Java and its library classes.
- Appropriate detection and reporting of errors when a user uses the command language incorrectly or inappropriately.
- Good coding style: well laid out with the fields, methods and constructors properly documented. In particular, use of javadoc-style notation is a requirement and adherence to these will be included in the marking scheme.

If in doubt about good style, a style guide is available: http://www.bluej.org/objects-first/styleguide.html.

Background

The idea is that users should be able to interactively create drawings manipulating a Pen object on a Canvas via typing commands into the terminal window. If you create a DrawingTool object and then call its `draw` method, the following will appear in the terminal window:

```
Welcome to the drawing tool.
Pen position: 0,0
The pen is up.

Type bye to exit.
>
```

If you type `bye` then the `draw` method will terminate. That is pretty-much all of the functionality that is currently available in the DrawingTool class. Your task will be to add more as described below.

How it currently works

A DrawingTool object uses an InputReader object to prompt for and read input from the user in the Terminal Window. Whatever the user types at the command prompt is read by the InputReader and separated into individual words – just as is done in the *tech-support* project in chapter 6. However, in this project the words are not stored in a HashSet but returned from the InputReader's getInput method as a LinkedList of individual String objects. For instance, if the user typed:

```
moveto 20 30
```

these would be returned as an ordered list containing three String objects: "moveto", "20" and "30".

What to add

Your task is to enhance the current functionality so that users can create drawings on the canvas using a Pen object. Commands typed by the user will need to be turned into calls to methods defined in the Pen class, such as `move`, `penUp`, etc. The task has been broken down into two stages but these are only for guidance.

In Stage One of the assignment you will implement a command language that will work with just a single Pen. In later stages the language will be extended to allow the creation and manipulation of multiple pens as well as user-defined command sequences.

The drawing language – Stage One [60%]

Implement the following commands to control the Pen:

- up
  Raise the pen off the canvas so that movements do not cause drawing.
- down
  Lower the pen onto the canvas so that movements cause drawing.

- move *distance*
  Move the pen the specified distance in its direction of movement. distance must be an integer value.
- moveto *x y*
  Move to co-ordinate position *x*,*y* on the canvas. Both *x* and *y* must be integers.
- turn *angle*
  Change the direction of movement by *angle* degrees. *angle* must be an integer.
- turnto *angle*
  Change the direction of movement to *angle* degrees. *angle* must be an integer.
- colour *colour-name*
  Change the drawing colour to be *colour-name*. *colour-name* must be one of the following: “red”, “blue”, “yellow”, “magenta”, “green” or “black”.
- help
  Print an explanatory help message.

Valid commands must have exactly the correct number of elements. For instance:

```
move 30
```

is valid, but

```
move 30 40
```

is not valid. Commands that are not valid should produce an error message and there should be no change to the state of the pen or the canvas.

Example for Stage 1

The following example sequence of commands would draw a green square with side length 20 pixels, starting at position 10,10:

```
up
colour green
moveto 10 10
down
turnto 0
move 20
turn 90
move 20
turn 90
move 20
turn 90
move 20
```

Note that the InputReader class includes methods that will help you to check for and convert strings that contain integer values.

The drawing language – Stage Two [40%]

Extend the command language so that multiple pens may be created and controlled. The following additional commands must be added:

- pen *name*
  Create a new Pen that will be referred to with the given *name*. It is an error to attempt to create a pen with a name that is currently in use.

- select *name*
  Select the pen with the given *name* as the one to which future commands will apply. It is an error to attempt to select a pen with a name that is not currently in use.
- delete name
  Delete the pen with the given *name*. The name will no longer be in use. It is an error to attempt to select a pen with a name that is not currently in use. It is possible to create a new pen with a name that has previously been used but deleted.

In this version, no Pen object should be created when the DrawingTool object is created. All Pen objects must be created as the result of a `pen` command.

Immediately following the successful execution of a `pen` or `select` command, all commands of the Stage 1 language apply to that pen. Other pens remain inactive until selected again.

Example for Stage 2

The following example will draw two parallel lines using two separate pens; the upper line is drawn in green and the lower line in magenta:

```
pen first
up
colour green
moveto 10 10
down
moveto 20 10
pen second
up
colour magenta
moveto 20 20
down
moveto 10 20
```

What to submit

You will submit a Java ZIP file of your project via a Moodle hand-in area to be provided shortly. In addition, a transcript of a typical user session and a screenshot of what it draws must be provided to illustrate a representative sample of your program working. Use the “Save to file” option of the BlueJ terminal to save the transcript as a text file.

Plagiarism and Duplication of Material

The work you submit must be your own. We will run checks on all submitted work in an effort to identify possible plagiarism, and take disciplinary action against anyone found to have committed plagiarism.

Some guidelines on avoiding plagiarism:

- One of the most common reasons for programming plagiarism is leaving work until the last minute. Avoid this by making sure that you know what you have to do (that is not necessarily the same as how to do it) as soon as an assessment is set. Then decide what you will need to do in order to complete the assignment. This will typically involve doing some background reading and programming practice. If in doubt about what is required, ask a member of the course team.

- Another common reason is working too closely with one or more other students on the course. *Do not* program together with someone else, by which I mean do not work together at a single PC, or side by side, typing in more or less the same code. By all means *discuss* parts of an assignment, but do not thereby end up submitting the same code.

- It is not acceptable to submit code that differs only in the comments and variable names, for instance. It is very easy for us to detect when this has been done and we will check for it.

- **Never** let someone else have a copy of your code, no matter how desperate they are. Always advise someone in this position to seek help from their class supervisor or lecturer. Otherwise they will never properly learn for themselves.

- It is not acceptable to post assignments on sites such as Freelancer and we treat such actions as evidence of attempted plagiarism, regardless of whether or not work is payed for. In addition, posting anywhere other than kent.ac.uk would be an infringement of the author's copyright.

Further advice on plagiarism and collaboration is also available at:

http://www.cs.kent.ac.uk/teaching/student/assessment/plagiarism.local

You are reminded of the rules about plagiarism that can be found in the course Handbook. These rules apply to programming assignments. We reserve the right to apply checks to programs submitted for assignment in order to guard against plagiarism and to use programs submitted to test and refine our plagiarism detection methods both during the course and in the future.

David Barnes