KIT101 Programming Fundamentals

PP Task 2.1 Turtle Graphics

Overview

Purpose: Use the methods of an existing object to customise one graphical program and then develop your own.

Task: Complete the three tasks of: customising an existing Turtle Graphics program; writing the algorithm for a Turtle Graphics program that draws your initials; and then implementing that program

Learning Outcomes: 1 👁 2 🔀 3 ✏

Time: Start during your Week 2 tutorial and submit before Week 3.

Resources:

- Introductory Programming Notes:
  - 03 Problem Solving with Computers
  - 05 Using Objects
  - KIT101 Support Library API Documentation
- Online Mini-lectures:
  - Solving Problems with Computers:
    - 📹 Algorithms
    - 📹 Importing & using objects
    - 📹 Some commonly used objects

Submission Details

Upload the following to the MyLO submission folder for this task:

- Your **modified RainbowTurtle.java** source file
- A **screenshot of the RainbowTurtle window** when running
- A Word document or PDF of your **algorithm for writing your initials** using a Turtle Graphics object
- Your **completed Initials.java** source file
- A **screenshot of your Initials program** when running

Assessment Criteria

A 🏁 Completed submission will:

- Exhibit good code style by matching the provided code's layout and indentation
- Have your name in both source files in the comment at the top, following the text `@author`
- Compile and run, with the screenshots showing both programs working
- Have each shape in RainbowTurtle appear in a different colour
- Include an algorithm using the Turtle Graphics commands of penUp, penDown, turn and move (and optionally origin and setColor) that will draw your initials
- Show that the Initials program implements that algorithm and draws your initials

Instructions

In this task you'll use a turtle graphics system to draw some simple images. Turtle graphics is based on the idea of a drawing device (the turtle) that sits on top of a flat drawing surface. In a typical turtle graphics system, the device can be instructed to raise or lower its drawing pen, rotate counter-clockwise by a specified amount and to move forward by a specified number of units.

Tip: Any link I provide to Wikipedia is optional and included for your interest only. Any essential information will be given here or in the accompanying notes and lectures (or in online documentation for the Java Standard Library).

Before you complain about turtle graphics being only suitable for primary school children, consider the common approach to manufacturing known as Computer Numerical Control (CNC) routing. CNC is used to instruct a cutting or shaping tool to move in two or three dimensions around a material (wood, metal, polymer, etc.) to manufacture a product. Of course, the reason we have you using turtle graphics is because, unlike most program code, you can *see* what your program has achieved.

The turtle graphics implementation we provide in this unit is defined in the `Turtle` class, and is included in the starter code for this task, available on MyLO. You can read its documentation on MyLO, under *Introductory Programming Notes* | *KIT101 Support Library* | *API Documentation*. The key methods you'll be interacting with are:

Turtle method	Description
`penUp()`	Lifts the virtual pen up. Any movement will not draw a line
`penDown()`	Places the virtual pen down. Any movement will draw a line
`turn(double delta)`	Rotates the turtle `delta` degrees counter-clockwise
`move(double dist)`	Moves the turtle `dist` units (pixels) in its current direction

Turtle method	Description
`origin()`	Moves the turtle to the bottom-left of the drawing area without drawing a line or modifying whether the pen is up or down
`setColor(Color c)`	Changes the pen colour to `c`. Requires `import java.awt.Color;` at the top of the program

Tip: Unlike ‘real’ computer graphics, where the origin (0, 0) at the top-right, turtle graphics typically places it at the bottom-left, with increasing values of its y-axis position moving the turtle *up*.

Work through the following steps to complete the task, which has three parts: modifying an existing program; creating an algorithm for a new program to draw your initials; and then implementing that program.

Part 1: Add some colour

1. Download the **2.1PP Turtle Graphics Starter Code** from MyLO. Extract the folder `2.1PP Turtle Graphics` from the zip file. The unzipped folder should contain `RainbowTurtle.java`, `Initials.java` and a `kit101` subfolder.

   Tip: Be careful on Windows, which allows you to navigate a zip file as if it were a normal folder. It will even let you ‘open’ the files it contains, except what it actually does it extract just that file to the temporary files directory before DrJava or another suitable program opens it. Since much of the time you’ll need other files for your program to work (the `Turtle` in this task), this will not work. So, always be sure to extract all the contents to your work area (which should not be Downloads).

2. The first task uses the file **RainbowTurtle.java**. Open it in DrJava and read the program.

   Activity: Read and understand the code in the program `RainbowTurtle.java`

   For the lines between the “********” comments, document each method call in a table like the one below. *Write your own answer before revealing the solution.*

Method name	Identifier of object containing the method	What arguments are passed?	How is the return value used? (n/a if there is no return value)

Method name	Identifier of object containing the method	What arguments are passed?	How is the return value used? (n/a if there is no return value)

3. Compile and run `RainbowTurtle.java` to see what it does. Then close the Turtle Graphics window.

> **Tip:** Don't worry if DrJava pops up a warning dialog box when you close the window it opens: GUI applications *must* call the command `System.exit()` in order to stop when their window has been closed, so nothing has gone wrong.

4. See at the top of the `RainbowTurtle.java` file that it imports `java.awt.Color`, but it doesn't yet actually use it. Modify the program by inserting calls to the turtle's `setColor()` method so that each of the four shapes—square, triangle, rectangle and cross—is a different colour (and no longer the default black).

> **Tip:** The Color class predefines the following colours as class-level data: black, darkGray, gray, lightGray, white, pink, red, orange, yellow, green, cyan, blue. For example, `Color.RED` is a `Color` object representing the colour red.

5. **Take a screenshot** of your running program and save it to submit once you've completed the next two parts of the task.

Part 2: Write a high-level algorithm

6. In Word or another document editor of your choice **create an algorithm** for using a turtle to draw your initials, using the commands (method names) defined above. You may wish to include a sketch of the desired result with some coordinates labelled as well as the

steps of your algorithm. Because this is just an algorithm you do *not* need to precede each command with an object name. When defining your algorithm keep in mind:

- that the turtle will initially be in the centre (250, 250) of a 500×500 drawing area, facing east, with its pen down and using black 'ink'; and
- that in this task you should *only* use the commands penUp, penDown, turn, move and, optionally, setColor (do *not* use origin in your algorithm or implementation as it will make Task 3.3PP very difficult).

Note: Your drawing does not need to look 'good'. Letters with curves are really hard to create in turtle graphics, so squarer shapes are fine. Imagine a digital clock display and how a 5 resembles an S.

Tip: Remember to include your name at the top of the document.

Part 3: Implement the algorithm

Stop! Have you written the algorithm for this yet? If not then do that now before you write any code.

8. Open the file **Initials.java**. You'll see that it partially implements your algorithm from Part 2. (Well, it declares and creates the Turtle, leaving the rest to you, but that's still 'partially'.)

9. Translate your algorithm into Java code. Compile and test it as you go. If you find algorithmic mistakes (if it doesn't draw what you intend) then also update your algorithm from Part 2.

   Tip: It's likely that all you'll need to do to 'translate' your algorithm into code is to precede each command (method call) with `painter.`, but if you get compilation errors like `cannot find symbol` then it may mean that one of your algorithm commands has a slightly different name to the equivalent Turtle method you need to use. Check the table above to be sure.

10. **Take a screenshot** of your running Initials program.

11. Finally, **submit** the five required files: the two screenshots (of RainbowTurtle and Initials), two completed programs (RainbowTurtle.java and Initials.java) and one algorithm document to the 2.1PP submission folder on MyLO.