

KIT101 Programming Fundamentals

PP Task 3.3 Stamp Method

Overview




Purpose: Learn how to create a parameterised method.

Task: Create a 'stamp' drawing method that allows you to draw your initials on the screen at different locations.

Learning Outcomes: 1  2  3 

Time: Start during your Week 3 tutorial and submit before Week 4.

Resources:

- Introductory Programming Notes:
 - 05 Using Objects
 - 07 Methods in Self-contained Programs
- Online Mini-lectures:
 - Solving Problems with Computers:  Some commonly used objects
 - Making Your Own Methods:
 -  ...in self-contained programs
 -  Flow of Control When Calling Methods

Submission Details

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

- Your **Stamps.java** source file
- A **screenshot** showing the execution of your Stamps program

Assessment Criteria

A  Completed submission will:

- Draw your initials at three different locations on the Turtle window, each in a different colour (if your drawing is large it's OK if two of them are partly off the screen)
- Be based on your 2.1PP Turtle Graphics initials drawing code

- Have an additional method called `drawStamp` that draws the initials
- Declare `drawStamp` with parameters for the Turtle object to use, and for the stamp's colour and location of its bottom-left corner (if you were to imagine a rectangle around the entire shape)
- Follow the Java coding conventions used in the unit (layout, use of case in variable names, including author name at the top)
- Compile, with the screenshot showing that it works

Instructions

In this task you will adapt your previous initials drawing code (the Initials program from 2.1PP Turtle Graphics) to make it a reusable method. This will allow you to use it like a [rubber stamp](#), placing your initials at any location within the Turtle Graphics window as many times as you like.

The (very) high-level algorithm for the final version of your method is as follows:

Method: `drawStamp(Turtle stamper, double x, double y, Color ink)`

1. Record current colour, direction and x and y locations of *stamper*
2. Set drawing colour to *ink*
3. Without drawing, move *stamper* to starting position (calculated so that the bottom-left corner of the final image is at (x, y))
4. Draw initials (adapted from your 2.1PP Initials code)
5. Without drawing, move *stamper* back to original location (recorded in step 1)
6. Set drawing colour and direction back to original values (recorded in step 1)

Tip: The reason that the (x, y) coordinates are type `double`, even though the drawing window's pixels are at integer-valued positions, is because the Turtle 'sees' its drawing area as a smoothly continuous space. When it draws a line it works out the nearest actual pixels to the 'true' smooth line it wishes to draw to be assigned a colour.

The instructions below will guide you through three stages in implementing this method:

- [Stage 1](#) wraps your drawing code in a method, the equivalent of Step 4 alone
- [Stage 2](#) adds parameters to control the stamp's colour and position within the window, equivalent to Steps 2–4
- [Stage 3](#) adds some additional housekeeping code to save part of the Turtle's state when the method starts and to restore it when it ends

Note: This is a challenging (but rewarding) task. Aim to fully complete the steps in each stage before moving onto the next one, otherwise you may be unable to progress.

Stage 1: Wrap up your original initials drawing code in a method

1. Download and extract the **3.3PP Stamp Method Starter Code** from MyLO. The folder should contain `Stamps.java`, which you will edit, and the `kit101` package (a subfolder containing all the Turtle code).
2. Open `Stamps.java` as well as your `Initials.java` from task 2.1PP in DrJava.
3. Define a new method in `Stamps.java` at the position indicated in the code:

```
/**
 * Draws the author's initials using the given Turtle.
 */
public static void drawStamp(Turtle stamper) {

}
```

4. Copy your initials drawing code from the `Initials` program into `drawStamp`, making the following changes:
 - you only need the lines of code from `Initials`'s `main()` method that give the Turtle instructions; so
 - you do not need to copy the code for declaring or creating the Turtle object; this will be given by the caller (`main()`) later; and
 - find-replace `painter` with `stamper` (the Turtle object's name in your new method).
5. Compile the code and correct any mistakes. Ask your tutor for assistance if you need it.

Tip: Refactoring code like this (copying from one location and moving it into a method) will often lead to small problems that will prevent your code from compiling, so don't worry if it didn't go completely smoothly.

6. Run the program to see what it does (**hint:** you may already be thinking about the *next* step at this point).
7. Well that was (probably) disappointing. All that effort and nothing on the screen. The reason is that merely defining a method does not mean that it will actually be executed at runtime. So add a call to your new `drawStamp` method at the indicated point in `main()`:

```
drawStamp(t);
```

8. Recompile and rerun. Your program *should* now behave the same as your Initials program did before. Also try adding a second call to `drawStamp` and then running the program again. *Is there any change in visible behaviour?*

Your implementation of `drawStamp` at this point is likely the equivalent of just [Step 4](#) from above. Now it's time to make it more flexible.

Stage 2: Add parameters to make the method more flexible

9. Add parameters to the header for `drawStamp` so that it looks like the following:

```
public static void drawStamp(Turtle stamper, double x, double y, Color ink)
```

and modify the method *call* in `main()` so that you give values for these (try 0, 1 and `Color.BLUE` to begin with). Your method won't use these values yet, but this will allow the code to compile.

10. **Modify the header comment** to mention the new parameters (e.g., "...with the bottom-left corner of the drawing at (x, y)"). When you write the header comment you should manually wrap the text to be no more than 80 characters wide.
11. Add a line at the start of `drawStamp` to set the colour to the value of `ink`. Recompile and rerun to see if it had the desired effect.

Tip: If you already used `setColor` in your initials drawing code, remove all other calls to it so that the stamp is a single colour.

12. Work out, on paper or on the computer, where the bottom-left of your drawing is *relative to the starting point*. Your drawing code may not actually draw anything at that point, but if you were to imagine a rectangle surrounding your drawing perfectly then this point would be the bottom-left of that rectangle.
For example, consider the figure below, which could be created by a Turtle by turning left 60°, moving 100, turning left 120°, moving 100, turning left 120°, and moving 100.

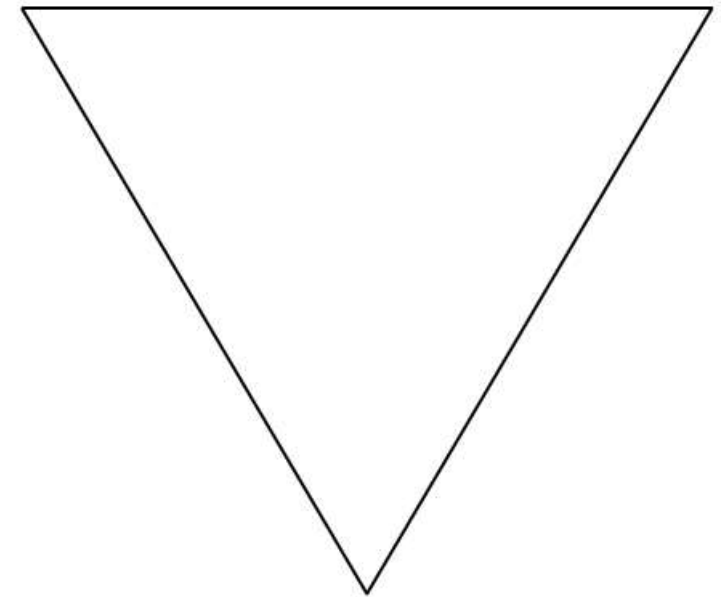
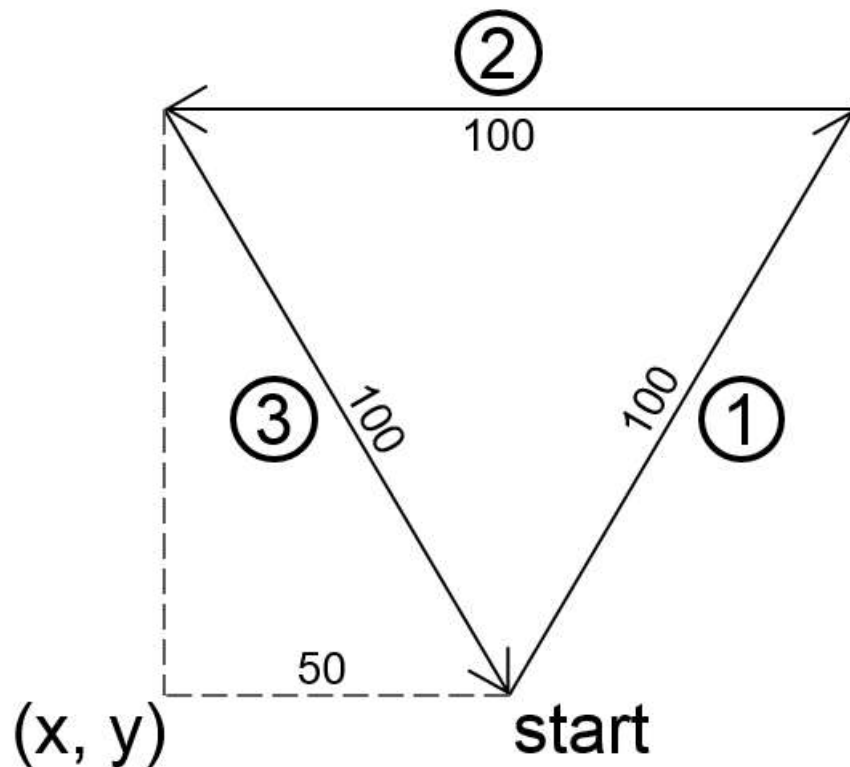


Figure 1: An image's bottom-left corner is the furthest left it goes (x) and the furthest down it goes (y)

You may be able to determine this point mathematically or by carrying out your initial drawing algorithm and observing where you end up compared to where you began. For example, in the triangle drawing above, assuming each side is 100 units long, x is -50 (half a side length to the left of the starting point) and y is 0 relative to the starting point (since the starting point is the lowest point of the drawing).

Tip: Remember that in Turtle graphics the bottom-left of the drawing area is (0, 0), so x -values increase to the right and y -values increase going up the screen.

13. We want to modify the starting point for your drawing code in `drawStamp` (using Turtle's `moveTo()` method) so that the bottom-left of your drawing is the same as the values of the parameters x and y . For example, if you determined the relative position of the bottom-left to be (-50, 0), like in the triangle diagram above, then you need to move to $(x + 50, y)$. Insert and modify the following code (replace *some offset value*) at the top of your method to achieve this:

```
stamper.penUp();
stamper.moveTo(x + some offset value, y + some offset value);
```

```
stamper.penDown();
```

14. To test that your modifications work, change the x and y arguments to your method to be 0 and 1, respectively (setting y to 1 may only be needed on Windows 8.1, where y = 0 is *just* below the bottom of the window).

Stage 3: Making the stamp code a little more independent

The final stage is to make the method have fewer 'side-effects' on the Turtle's state, so that it could be used in combination with other Turtle commands. You will do this by storing the values of some Turtle properties when the method begins and then restoring (setting) these again just before it ends.

15. Declare additional local variables at the top of drawStamp to save the Turtle's initial state:
- its pen colour: oldInk, a Color object
 - its angle: oldDirection, a **double**
 - its (x, y) position: oldX and oldY, both **doubles**
16. After those declarations add assignment statements to get these values from stamper and store them in the variables. You will need four assignment statements, one for each piece of information.

Tip: Consult the Turtle's API documentation (see *Introductory Programming Notes/KIT101 Support Library API Documentation*) to see what methods it has that start with the name *get*.

17. At the end of drawStamp follow the following algorithm to restore the Turtle's state:

```
lift pen up  
move to (oldX, oldY)  
set colour to oldInk  
turn to oldDirection
```

As above, you'll find suitable methods in Turtle for each of these actions.

You can (temporarily) test that this works by calling drawStamp in main() then, still within main(), placing the pen down and moving some distance. If everything is working as it should you will see your coloured stamp at one location and a black line starting at the centre of the drawing space (the Turtle's original condition).

18. In main(), **add two more calls to drawStamp**, at different locations and in different colours.

19. **Take a screenshot of the Turtle window** when running your program. Submit the screenshot and Stamps.java source file to the task's submission folder on MyLO.

Note: Your drawStamp method will be very long. While usual good practice is to have methods of no more than 20 lines of code, given the Turtle is so tedious to draw with it is inevitable that your method will be longer than that.