

COMP 1771 – Fall 2008

HW#1: Graphics Command Interpreter

Instructor:

Prof. Matt Rutherford – mjr@cs.du.edu

Due by:

10:00am on September 23, 2008

Overview

The goal of this homework is to create a Java applet that is capable of reading a file containing drawing commands and to render the picture described by the commands. This is an example of a common activity in programming and computer science, in which you come up with a simple language to describe some behaviors of interest, and then implement an interpreter for the language to execute the behaviors.

Your applet will use an instance of `java.util.Scanner` to read input that contains a list of drawing commands. Each command has different syntax that you will use the scanner to parse and interpret appropriately.

In addition to processing the basic drawing commands, your interpreter will have to translate between “user coordinates” and “screen coordinates”. *Screen coordinates* are simply the natural coordinates of the `Graphics` object, with the origin in the top-left corner, *x* increasing to the right, and *y* increasing downward. *User coordinates* are more natural for the user with the origin in the bottom-left corner, and *x* and *y* increasing in whichever direction makes sense for them. The final wrinkle is that user coordinates must be scaled to fit in the visible canvas.

User coordinates are set with the `coords` command described below. By default (i.e., without any `coords` commands), *user coordinates* are identical to *screen coordinates*.

Policies

As stated in the class syllabus:

Homework assignments and labs may be discussed in a general fashion with other students. You should not discuss specific solutions or code. You should NEVER copy assignments that have been written by another student or allow another student to copy your assignments. If any of your work includes ideas or quotes from a book, paper, or web site, you should clearly cite the original source.

Your code (a single Java source file) should be emailed as an attachment to me at mjr@cs.du.edu before 10am on September 23, 2008. Please use the subject: “COMP 1771 Homework 1”.

Code Skeleton

In order to parse a variable number of commands, we will use a `while` loop controlled by the method `Scanner.hasNext()`. So, your code will have an overall structure like:

```

// Your name here
public class GraphicsInterpreter extends JApplet
{
    public void paint(Graphics canvas)
    {
        Scanner scanner = ...; // instantiate Scanner appropriately

        while(scanner.hasNext()){ // this loops over all tokens
            // read the command
            // parse its parameters
            // call corresponding canvas methods
        }
    }
}

```

Command Language

The input data will consist of a list of commands and their arguments separated by whitespace. Each command consists of a command name, parameters, and is terminated by a semi-colon (separated from the last argument by white space). For example, the following commands would draw the basic happy face:

```

oval 100.0 50.0 200.0 200 false ;
oval 155.0 100.0 10.0 20 true ;
oval 230 100.0 10.0 20.0 true ;
arc 150.0 160 100.0 50.0 180.0 180.0 false ;

```

The command language is case insensitive. In most cases, the commands correspond to one or more methods of the `Graphics` class and have arguments with the same name as in the Javadoc. Coordinates and widths in the commands are all in *user coordinates*.

In order to parse the command language properly, you should become familiar with the `Scanner.hasNext()` methods which let you peek ahead a single token without actually reading the token.

The full set of commands and parameters that your code should support is:

- **arc** – draws an arc. See `Graphics.drawArc()` and `Graphics.fillArc()`. The parameters are:
 - **x** – required double;
 - **y** – required double;
 - **width** – required double;
 - **height** – required double;
 - **startAngle** – required double;
 - **arcAngle** – required double; and
 - **filled** – optional Boolean, defaults to **false** if not present.

Examples:

```

arc 150.0 160.0 100.0 50.0 180.0 180.0 ;
ARC 150.0 160.0 100.0 50.0 180.0 180.0 true ;

```

- **line** - draws a line. See `Graphics.drawLine()`. The parameters are:

- **x1** – required double;
- **y1** – required double;
- **x2** – required double; and
- **y2** – required double.

Example:

```
line 10 10.5 400 323.4 ;
```

- **oval** – draws an oval. See `Graphics.drawOval()` and `Graphics.fillOval()`. The parameters are:

- **x** – required double;
- **y** – required double;
- **width** – required double;
- **height** – required double; and
- **filled** – optional Boolean, defaults to **false** if not present.

Examples:

```
oval 40 40.25 101.1 111 ;
```

- **rect** – draws a rectangle. See `Graphics.drawRect()`, `Graphics.drawRoundRect()`, `Graphics.fillRect()` and `Graphics.fillRoundRect()`. The parameters are:

- **x** – required double;
- **y** – required double;
- **width** – required double;
- **height** – required double;
- **arcWidth** – optional double, if this is set, `arcHeight` must also be set;
- **arcHeight** – optional double, if this is set, `arcWidth` must also be set;
- **filled** – optional boolean, defaults to **false** if not present.

Examples:

```
rect 40 40.25 101.1 111 ;
RECT 40 40.25 101.1 111 10.0 10.0 TRUE ;
```

- **text** – places a text string at a location. See `Graphics.drawString()`. The parameters are:

- **x** – required double;
- **y** – required double; and
- **str** – String, at least one token. All tokens between the `y` parameter and the semi-colon should be concatenated together (with spaces between, preserving the original case) into a single String. Parsing this properly will require a loop. **HINT:** first get this command working for a single token and deal with multiple tokens after the rest is working.

Examples:

```
text 100 23.24 Hello ;
TEXT 100 23.24 Hello, World ;
```

- **color** – sets the current color. See `Graphics.setColor()` and `java.awt.Color`. Color can be called in two ways (different parameters). The first:

- **name** – one of (case insensitive): black, blue, cyan, darkgray, gray, green, lightgray, magenta, orange, pink, red, white, yellow

or:

- **r** – required double;
- **g** – required double; and
- **b** – required double.

Examples:

```
color WHITE ;
COLOR 200 200 100 ;
```

- **coords** – set the user coordinates. See `Applet.getHeight()` and `Applet.getWidth()` (both of these are defined on `java.awt.Component` which is a superclass of `Applet`). The parameters are:

- **x1** – the x coordinate of the bottom-left corner of the bounding box (required double);
- **y1** – the y coordinate of the bottom-left corner of the bounding box (required double);
- **x2** – the x coordinate of the upper-right corner of the bounding box (required double);
- **y2** – the y coordinate of the upper-right corner of the bounding box (required double);

Examples:

```
coords 0 0 1000 1000 ;
COLOR 0 400 400 0 ;
```

Reading Input

Your applet should support two methods of reading input: from a file, and from the standard input (i.e., the keyboard). The `Scanner` class is used in both cases.

The general API is as follows:

- The system property “input.file” is used to pass the file name containing data (see `System.getProperty()` and the slides from class 3).
- If this property is not set, or it contains the string “-” (a dash), then the data should be read from `System.in`.

Grading

Module	Percentage
Command language processing of all parameter patterns for 6 non-coords commands.	60%
Complete handling of user coordinates.	15%
Correct handling of text command.	10%
Support reading from both files and <code>System.in</code> .	10%
Usage of "helper methods", assertions in parsing, variable naming conventions, style	5%