Lindenmayer System Lab 6

- All requirements of Lab 5 plus,
- User also inputs an angle from 0 to 360 degrees.
- All '+' and '-' commands turn right or turn left by that angle.
- The rule input may contain up to 25 rules.
- The user is asked if he or she wants to change one of the color commands (K, R, G, B, C, and O) to a custom color. If the user selects "yes", first ask the user for the letter who's color will change. Then, use JColorChooser.showDialog() to display a color chooser dialog with the current letter's color as the default. If the user clicks cancel, then leave the letter's color unchanged. Otherwise, replace it with the selected color. Continue asking the user if he or she wants to change a color until user selects "No".
- New interface for buildString and will be tested by robot.
Example: Using Color Chooser

```java
import java.awt.Color;
import javax.swing.JColorChooser;

public class Test {
    public static void main(String[] args) {
        Color userColor = JColorChooser.showDialog(null, "Pick Color for B", Color.BLUE);
        System.out.println(userColor);
    }
}
```

New Signature for `buildString`

- LSystem2 class must implement the method:
  ```java
  public static String buildString(
      String axiom, String[] ruleList,
      int generation)
  ```

- Your program will be tested by a grading robot that calls this method. If you do not implement the interface `exactly` as shown, you **WILL GET ZERO** points for `buildString`.

- `buildString` must return a string containing no spaces that results from the given number of generations.

  For the robot to work, `buildString, must not` use any class variables.
Moving the Turtle when Heading = $\theta$

Graphical interpretations of $f$, $g$, and $h$:

$$x' = x + d \cos \theta$$
$$y' = y + d \sin \theta$$

$d = 10$ pixels

Special case: $\theta = 90^\circ$, $\cos \theta = 0.0$ and $\sin \theta = 1.0$

Example: Heading = $30^\circ$

$$\sin 30^\circ = 0.5$$
$$\cos 30^\circ \approx 0.866$$

$$x' = x + d \cos \theta$$
$$y' = y + d \sin \theta$$

Turtle locations must be maintained as double.

When drawing, truncate 8.66 to the int 8.
However, after two steps at $30^\circ$, 8+8 would be 16, but 8.66+8.66 = 17.32 (truncating to int 17).
Trigonometry in Java

The equations:
\[ x' = x + d \cos \theta \]
\[ y' = y + d \sin \theta \]

Can be written in Java as:
```java
double d = 10.0; //pixels
double heading = 45.0; //degrees
double x2 = x + d*Math.cos(Math.toRadians(heading));
double y2 = y + d*Math.sin(Math.toRadians(heading));
```

Koch Snowflake

The Koch snowflake appeared in a 1904 paper by the Swedish mathematician Helge von Koch, and is one of the earliest fractional dimension (fractal) curves to have been described.

Koch's method of construction:
1. Start with an equilateral triangle.
2. Divide each segment into three segments of equal length.
3. Draw a new equilateral triangle that has the middle segment from step 2 as its base and points outward.
4. Remove the line segment that is the base of the triangle from step 2.

```
String axiom = "f++f++f";
String[] rules = {
    "f=af-cf++f-af", "a=", "c="};
double turnAngle = 60.0;
int generation = 2;
```
Koch Snowflake with Mitsubishi Diamonds

```java
String axiom = "Rf++f++f----Bh++h++h";
String[] rules =
{ "f=f-f++f-f",
  "h=h+h--h+h"
};
double turnAngle = 60.0;
int generation = 3;
```

Sierpiński Triangle

```java
String axiom = "f";
String[] rules =
{ "f = h - f - h",
  "h = f + h + f"
};
double turnAngle = 60.0;
int generation = 9;
```

The Sierpiński triangle is a fractal named after Wacław Sierpiński who described it in 1915. It was originally constructed as a curve (as it is with this L-system). It can also be constructed using the "Chaos Game", or by using an Iterated Function System.
Space-Filling Peano Curve (gen 1-3)

```java
String[] rules =
{ "f = G f + B h + + h - G f - - f f - B h +",
  "h = - G f + B h h + + h + G f - - f - B h",
  "G =",
  "B ="
};

double turnAngle = 60.0;
```

Space-Filling Peano Curve (gen 4-5)

Space-filling curves or Peano curves are curves, first described by Giuseppe Peano (1858 – 1932), whose ranges contain the entire 2-dimensional unit square (or the 3-dimensional unit cube).

The idea of a 1-dimensional object being space filling was found to be highly counterintuitive.

In their limit, the Koch Snowflake, the Sierpiński Triangle, and the Peano Curve all are everywhere continuous with infinite length, and are nowhere differentiable.
L-System: Grass

Axiom: \( x \)
Rules: \( x = B f - [ [ x ] + x ] + f [ + G f x ] - x \)
\( f = ff \)

Initial Angle: 65.0°;
Turn Angle: 25.0°;
Generations: 6

Extra Credit (carry over from lab 5):

Any extra credit option of lab 5 that you did not do in lab 5 can be done in lab 6:
- Fit to Screen
- Resizing Screen
- Metallic Coloring
- Do it in 3D
Extra Credit [+10] : Stack commands

Implement the stack commands ' [' and ' ]'.

- The ' [' command saves the current state of the turtle.
- The ' ]' command restores the turtle to the state saved by the matching ' [ '.
- Both of these commands are terminals.
- These commands can be nested to any level. For example:

  ```
  +Bf [+Gff [+Rf]] f
  ```

Extra Credit [+10] : Age Lines

Implement the '!' command.

- The '!' is a terminal.
- The '!' gets older. A newly born '!' draws a line at the current heading of the same length as an 'f' or 'h'.
- The length of each '!' increases by a factor of 1.5 with each generation it is old.
- The color of each '!' must get darker with each generation it is old (of course, black cannot get darker).
L-System Output: Fern (Generation 1, 2 & 3)

Axiom: \( f \)

Rules:
\[
\]

Initial Angle: \( 90° \)
Turn Angle: \( 8° \)
Growth: \( 2.5 \)

Note: The main stems drawn with ! symbols and need to show color aging. The tips are drawn with \( f \) and can be any color (except the background color). All of the tips must be the same color.

L-System: Fern – generation 5

Axiom: \( f \)

Rules:
\[
\]

Initial Angle: \( 90° \)
Turn Angle: \( 8° \)
Growth: \( 2.5 \)
Generations: \( 5 \)
Extra Credit: [+10] Speed (part 1 of 2)

In Java, Strings are immutable: Data in a String cannot be changed.

Consider the code:

1) `String str = "";`
2) `for (int i=0; i<n; i++)`
3) `{ str += "*";`
4) `}`

Each time line (3) is reached:

a) New memory is allocated that is 2 bytes larger than the memory currently used for `str`.

b) All data in the current `str` is copied to this new memory.

C) The last two bytes of the new memory for `str` are filled with "*"

d) The old `str` is marked for garbage collection.

When `n` is a few hundred, this all happens very quickly.

When `n` is in the thousands or millions, this is very slow.

---

Extra Credit: [+10] Speed (part 2 of 2)

Unlike in a Java String, the data in a `char[]` can be changed.

1) `int n = 100000000;`
2) `String str = "";`
3) `for (int i=0; i<n; i++)`
4) `{ str += "*";`
5) `}`
6) `char[] array = new char[n];`
7) `for (int i=0; i<n; i++)`
8) `{ array[i] = ' '*;`
9) `}`

---

20
String[] split(String delimiter)

Very useful method of the String class for breaking the user input rules into an String array Where each element of the array is a different rule:

```java
String input = "Hello;There;My;Friend";
String[] list = input.split(";");

for (int i=0; i<list.length; i++)
{ System.out.println("0) " + list[i]);
 }
```

Output:
0) Hello
1) There
2) My
3) Friend