COMS 4115
Programming Languages and Translators
IpsoFracto: Fractal Generation Language

Leo Gertsenshteyn (lpg2006 [AT] columbia [DOT] edu)
Sarah Gilman (srg2104 [AT] columbia [DOT] edu)
Rob Notwicz (rcn15 [AT] columbia [DOT] edu)
Anya Robertson (alr33 [AT] columbia [DOT] edu)

Part I

IpsoFracto Whitepaper

Introduction

IpsoFracto is a language for defining fractals from basic lines, polygons and kinks (bends in a line). It produces bitmap images of a fractal from the code. This is a handy tool allowing a user to alter the definition of their fractal, by changing the initial shape or the kinks applied to the lines, and quickly see the results.

Usability

IpsoFracto is a simple language with very few syntactic restraints. The language is easy to use due to the visual nature of the results; if a user is unsure of what the attributes of a kink do, they can always change them and see immediately what they control. In this way, IpsoFracto is well suited to the beginner, even one relatively unfamiliar with programming or fractal math.

Portability

IpsoFracto uses ANLTR to parse the code initially, and a Java interpreter to produce bitmap images from the AST. This allows IpsoFracto programs to be compiled on any platform that supports ANTLR and Java.

Error Handling

IpsoFracto is a fairly simple language, with very few possible errors. The parser catches all syntax errors, and the interpreter is responsible for catching errors such as invalid points. It will obviously be possible for a user to get unexpected images, but the nature of the language makes it easy for the user to return to the program and alter lines or kinks.

Library

Kinks are defined in a library. They take parameters indicating ratios of the entire line on which they are placed to the height and width of the kink itself. For instance, `trikink(5,5)` produces a triangle shaped kink with a height and width each roughly equal to one fifth of the line on which it is placed. All kinks can be applied to lines using the `apply` function, with the key-word parameters for positions (`LEFT`, `RIGHT`, and `CENTER`) and line orientation (`POS` and `NEG`) as well as a parameter for specifying how many kinks should appear next to each other.

Data Types

IpsoFracto has three main data types: `Line`, `MultiLine`, and `Polygon`. These define the basic starting points for the fractal. All three are based on an implicit type, the `Point`, which is a 2-D position in the bitmap. `Line`s, `MultiLine`s, and `Polygon`s are all sets of `Point`s, linked with the operators `->` and `<-`. These operators indicate the orientation of a line. For example,

```
Line myline (0, 10) -> (0, 30)!
```

defines a line from (0, 10) to (0, 30), while

```
Line myline (0, 10) <- (0, 30)!
```

defines a line from (0, 30) to (0, 10). These operators were chosen because it is intuitively clear which point is the start and which is the end of the line.

`Multiline`s and `Polygon`s are collections of points, with the `Polygon` having an added implicit line from the last point to the first. For example,

```
Polygon mysquare = (0,10)->(10,10)->(10,20)->(0,20)!
```

defines a square.

Scope

All `Line`s defined in a program will be valid and available throughout the program. `Line`s, `MultiLine`s and `Polygon`s should be defined at the top of the file since they are the starting point for the fractal.

Control flow

IpsoFracto supports `iter` loops, a variant of the for loop, in order to allow the user to chose how many times they wish to apply a kink to a line. These can be controlled by the number of kink iterations, by the time to add kinks, or by the kink size.

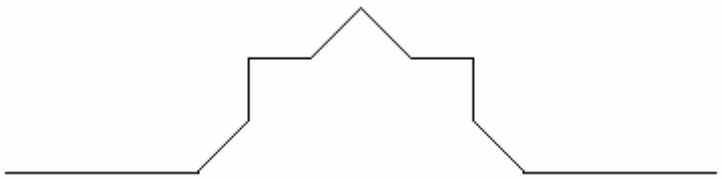
IpsoFracto also supports a variation of the standard `if` statement, in order to allow the user to alter the kink applied based on variables such as the current iteration count.

Sample syntax

```
line lineA (20, 0) -> (50, 0)!
iter 2 {
```

```
        apply ( lineA, CENTER, POS, trikink(3,3), 1 )!
    }
```

This syntax defines a single line, and then iteratively places a single triangular kink in the line at the center position and positive orientation. This syntax produces a fractal that looks like this:



Extensions

It is easy to extend IpsoFracto by defining libraries with new kinks.

Part II

IpsoFracto Language Tutorial

Introduction

IpsoFracto is a simple language for fractal generation. The graphical fractal output is written to a bitmap file. IpsoFracto is fairly easy to use, and a great deal can be accomplished with a small amount of code. This tutorial will walk you through writing your first IpsoFracto program and highlight the basic commands you will need to develop more complex programs.

The most basic block required for an IpsoFracto program is a line, in a loose sense, as all graphical manipulation is ultimately done on a line. You may declare and assign `Point`s, `Line`s, `Polygon`s, `MultiLine`s, and/or `Group`s in IpsoFracto programs. `Polygon`s, `MultiLine`s, and `Group`s, can be seen as collections of `Line`s, and `Point`s as the building blocks for `Line`s.

The lines within a `Polygon` or within a `MultiLine` are connected to one another. A polygon has the distinction of being an implicitly closed shape. A `Group`, on the other hand is simply a collection of possibly unrelated lines. Every object declared in an IpsoFracto program is automatically part of a group called `Thegroup`. This is useful to allow the application of effects on all lines in the program. Lines are manipulated by applying kinks to them. A kink is simply a bend in the line.

Installing IpsoFracto

First unzip the IpsoFracto zip archive. In the root directory are two batch files. Run build.bat to compile all code, by typing `build` on the command line from the root directory of IpsoFracto.

Getting Started

Every IpsoFracto program is in a separate text file, and each begins with an export line. This line is of the following format:

`#export "<`*`path/filename`*`>"`,

where <*`path/filename`*> is replaced by the path and filename of the output bitmap file. Choose a location for your output file, create a text file for your IpsoFracto program, and insert the export line at the top of the file.

You are now ready to add statements to your program.

Statements

There are several types of statements valid in IpsoFracto programs: declarations, assignments, postfix increment and decrement, control flow statements, and function

calls. All statements except the initial `export` statement and control flow statements end with a bang (`!`) or question mark (`?`). The choice between these is at the programmer's discretion.

Declarations and Assignments

Points are declared with the `Point` keyword, followed by a label for the point, the assignment operator (=), then either the label of an existing point, or the definition of a new one. A new point definition consists of the x- and y-coordinates, separated by a comma, and enclosed in square brackets (`[]`). The following is a point declaration with a new point definition:

```
Point p = [20, 0]!
```

Let this be the next line in your IpsoFracto program. Note that previously declared `Point`s may be reassigned later in the program, using the same syntax rules as the declaration, only without the keyword `Point`.

Lines can be declared with the `Line` keyword, followed by a label for the line, the assignment operator (=), then either the label of an existing line, or the definition of a new one. A new line definition consists of a point label or point definition, followed by an orientation operator (`->` or `<-`), followed by a point label or point definition. The orientation operator should "point" from the start point of the line to the end point of the line and is significant only in determining the "positive" or "negative" side of the line. We will discuss orientation further in the section on applying kinks.

Insert the following line of code in your program:

```
Line myline = p -> [50, 0]!
```

Polygons, multi-lines, and groups may also be declared with the `Polygon`, `MultiLine`, and `Group` keywords, respectively, but we will not be using them for this simple program. Please see the Language Reference Manual for how to declare these types.

Now we will discuss control flow statements and function calls before adding anything else to your first IpsoFracto program. At this point, your program should look something like this:

```
#export "<path/filename>",

Point p = [20, 0]!
Line myline = p -> [50, 0]!
```

Control Flow Statements

There are two types of control flow statements in IpsoFracto: iter and if-else.

The iter control statement is a variation on the standard for loop. It is used to repeatedly run a block of code. An iter statement begins with the `iter` keyword, followed by

control values enclosed in parentheses (`()`). Inside the parentheses should be a start value and end value, separated by the yadda operator (`..`), followed by a comma (`,`) and an increment or decrement step. Next should be a control variable label and an opening curly brace (`{`). The curly brace is then closed (`}`) after the block of code that is to be repeated. The braces may optionally be omitted if the block is only one statement long.

The iter statement will likely make much more sense when we add an example to your program below. First, however, we will discuss if-else statements and function calls.

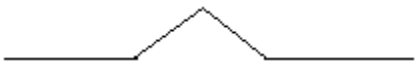
The if-else control statement in IpsoFracto is essentially the standard if-else found in most programming languages. The if-else statement begins with the `if` keyword, followed by an expression that evaluates to a boolean value, enclosed in parentheses (`()`). Next comes an opening curly brace (`{`), then the block of code to be conditionally executed, then a closing curly brace (`}`). At each closing curly brace, one may place the keywords `else if`, followed by another boolean expression in parentheses, and another block of code in curly braces. The final section may simply be an `else`, with no boolean condition. The opening and closing curly braces (`{}`) may be optionally omitted if the block of code is only one statement long.

Function Calls
There are two built-in functions in the IpsoFracto language: `add` and `apply`.

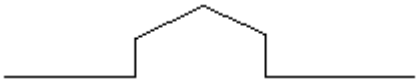
The `add` function is used to add an object to a group. Since we are not declaring any groups in our program, we will not go into detail about the add function. Please see the Language Reference Manual.

The apply function is used to apply kinks to the lines of an object, or to a single line. This is the fundamental operation supported by IpsoFracto. A call to apply starts with the `apply` keyword, followed by an open parenthesis (`(`), the object to whose lines to apply the kink, a position constant, an orientation constant, the kink to be applied, and the number of "parallel" kinks to apply, then a closing parenthesis (`)`). The position constant is one of `CENTER`, `LEFT`, or `RIGHT` and indicates the position along the line to place the kink. The orientation constant is either `POS` or `NEG`, indicating the side of the line on which to place the kink. Specifying `POS` will place the kink on the left side of the line, when facing along the orientation of the line from start point to end point, while specifying `NEG` will place the kink on the right side. The kink to be applied should be one of those defined in the libraries. The kinks included in the standard library are as follows:

trikink:



squarekink:

housekink: 

The syntax for a kink is <kinkname>(<width ratio>, <height ratio>). The width and height ratios are the ratio of the original line to the width and height of the kink applied. Finally, the number of kinks to be applied in parallel indicated the number of kinks to apply next to one another in the position specified.

You are now ready to insert the main portion of your program. Insert the following into your program:

```
iter (2..0, -1) x {
     if (x % 2 == 0 ) {
          apply(Thegroup, CENTER, POS,
                                trikink(3,3), 2 )!
     } else {
          apply(Thegroup, LEFT, POS,
                                squarekink(5,5), 1 )!
     }
}
```

The first line opens an iter block, which will be controlled by the variable `x`. This variable will start with a value of 2, and will be decremented by 1 in each iteration. The iter block will be exited when `x` equals 0. Within the iter block, there is a if-else block. It states that if *x mod 2* is equal to 0, then two triangular kinks, whose width and height are 1/3 of the size of the original line should be placed in the center of every line in `Thegroup` (every line in the program so far), and they should lie to the left of the line. Otherwise, one square kink should be placed on the left end of each line in the program, the width and height of the square should be 1/5 the size of the original line, and the square should lie to the left of the line.

Your program should now look like this:

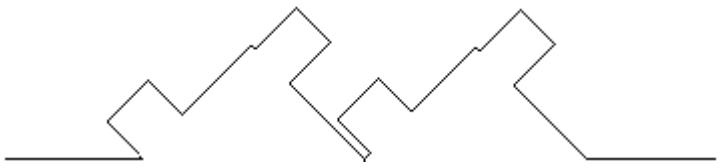
```
#export "<path/filename>",

Point p = [20, 0]!
Line myline = p -> [50, 0]!

iter (2..0, -1) x {
     if (x % 2 == 0 ) {
          apply(Thegroup, CENTER, POS,
                                trikink(3,3), 2 )!
     } else {
          apply(Thegroup, LEFT, POS,
                                squarekink(5,5), 1 )!
```

```
        }
    }
```

Running Your Program

Run run.bat in the root directory of IpsoFracto to run any IpsoFracto program. This batch takes a single argument: the name of the IpsoFracto program to be run. This will automatically open a Swing GUI with the generated fractal and saves the fractal image in the bitmap file specified in the export line of the program. Run the batch file by typing `run <path/program.txt>` on the command line in the root directory of IpsoFracto, where <*path/program.txt*> is replaced by the path and file name of your program.

This program should produce the following fractal:



Congratulations! You have written and run your first IpsoFracto program! Please refer to the Language Reference Manual for more details on using IpsoFracto.

Part III

IpsoFracto Language Reference Manual

Introduction

IpsoFracto is a language for defining and displaying fractals. IpsoFracto code is interpreted and produces bitmap images, so a programmer can easily see and alter the fractal. Tokens in the language include identifiers, keywords, operators and separators. Whitespace is not considered meaningful. A token is recognized as the longest matching sequence possible in the code.

In the beginning there was Thegroup.

- IpsoFracto revolves around groups of lines, multi-lines, and polygons
- Importance of the group is its use in iteration. Particularly the ability to apply kinks to a group (thus applying them to every line in it)
- There is an implied group of everything created called Thegroup
- Programmers can also create sub-groups

Installing and Running IpsoFracto

First unzip the IpsoFracto zip archive. In the root directory are two batch files. Run build.bat to compile all code, by typing `build` on the command line from the root directory of IpsoFracto.

Run run.bat in the root directory of IpsoFracto to run any IpsoFracto program. This batch takes a single argument: the name of the IpsoFracto program to be run. This will automatically open a Swing GUI with the generated fractal and saves the fractal image in the bitmap file specified in the export line of the program. Run the batch file by typing `run <path/program.txt>` on the command line in the root directory of IpsoFracto, where <*path/program.txt*> is replaced by the path and file name of your program.

Conventions

All IpsoFracto programs begin with a line of code of the following format:

```
#export "<path/filename>",
```

where <path/filename> is the path and filename of the target output bitmap file.

Java's standard comments are supported in IpsoFracto. Multi-line comments are enclosed in `/*` and `*/`, while single line comments start with `//` and end with a new line. Single line comments may appear on the same line after code.

Code blocks, such as control statements or loops, are separated by the { } symbols. Single statements such as declarations and function calls to apply kinks to lines end either with an exclamation point (`!`) or with a question mark (`?`). Function arguments

immediately follow a function name and are enclosed in parenthesis and separated by commas.

Keywords

IpsoFracto's reserved words are:

Word	*Type*	*Purpose*
`add`	function	Adds a line to a multi-line or an object to a group
`apply`	function	Applies kinks to a line
`bool`	data type	Data type for Boolean declarations
`else`	control flow	Optional portion of the if conditional statement
`export`	start rule	Used to begin the program and specify bitmap filename
`float`	data type	Data type for floating point number declarations
`Group`	data type	Data type for encapsulating other data types
`if`	control flow	Conditional statement
`int`	data type	Data type for integer declarations
`iter`	control flow	Loop statement
`Line`	data type	Data type for line declarations
`MultiLine`	data type	Data type for multi-line declarations
`Point`	data type	Data type for point declarations
`Polygon`	data type	Data type for polygon declarations

Keywords in IpsoFracto are case sensitive.

Additional keywords are specified by the libraries. Specifically, any library will have a keyword for each type of kink defined in that library. The standard libraries include routines for: `trikink`, `squarekink`, and `housekink`. (See **Kink Library** below)

Predefined constants

Word	*Type*	*Description*
`CENTER`	imperative constant	Imperative that kink should be applied to the center of the line
`LEFT`	imperative constant	Imperative that kink should be applied to the left of the line
`NEG`	imperative constant	Imperative that kink should be applied to the negative orientation of the line
`FALSE`	constant	Literal for the Boolean false

`POS`	imperative constant	Imperative that kink should be applied to the positive orientation of the line
`RIGHT`	imperative constant	Imperative that kink should be applied to the right of the line
`TRUE`	constant	Literal for the Boolean true

Operators

The following operators are defined, and are in order of decreasing precedence:

Operator	*Purpose*
`{ }`	Separators for loop code blocks (`iter` blocks)
`( )`	Separators for grouping function arguments and loop controls
`[ ]`	Separators for point declarations
`,`	Separates items in dynamic lists
`.`	Used to associate a function to an object
`%`	Mod – valid on integers
`*` `/`	Multiply and Divide – valid on integers, floating point numbers and points
`+` `-`	Plus and Minus – valid on integers, floating point numbers and points

`&&`
`||`
`~`
	And, Or, and Not – valid on Booleans

`==`
`<=`
`>=`
`<`
`>`
	Equals, Less than or equal to, Greater than or equal to, Less than, Greater than – valid on Booleans

`..`	The yadda operator, produces items of a dynamic list
`->` `<-`	The orientation operators
`=`	Assignment operator
`++` `--`	Post-fix increment, Post-fix decrement – valid on integers and floating point numbers

Data Types

Atomics

IpsoFracto has three types of literals: integers, floating point numbers, and booleans. Integers are any sequence of numbers 0-9 repeated. Floating point numbers consist of any integer followed by a decimal point, followed by another integer. The exponent functionality available for floating point numbers in many languages is not included in

IpsoFracto. (Any factors of ten that take more characters to type out than to write in e-notation would either be indiscernible or would dominate the screen.) Booleans consist of the predefined constants TRUE and FALSE. When operators are applied to different types, all literals are promoted to the more general type. Literals may be defined with the keywords `int`, `float`, and `bool`.

Moleculars

There are five types that are composed of atomics in IpsoFracto: points, lines, multi-lines, polygons, and groups. All of these types can be associated with variables, whose names follow the identifier rules as shown below. In addition each of these types may be defined dynamically, using variable names in their declarations.

Points are defined by pair of `int`s or `float`s, such as [10, 20]. A point variable may be defined using the `Point` keyword. The following two examples are equivalent:

```
Point p = [10, 20]!

int i = 10!
int j = 20!
Point p = [i, j]!
```

A line is a pair of points with one point tagged as the start point and the other as the end point. The direction of the line is from the start to end, and the positive orientation of the line is the left side when facing the direction of the line. A line variable may be defined with the keyword `Line`. The operators `->` and `<-` define the relationship between the points and therefore the direction of the line, in the visually expected manner. The following three examples are equivalent:

```
Line l = [0, 4] -> [10, 20]!

Point p = [10, 20]!
Line l = [0, 4] -> p!

Point p = [10, 20]!
Point q = [0, 4]!
Line l = q -> p!
```

Polygons are, in essence, lists of points, with the implication of lines in a continuous, single orientation to connect them. Polygon variables are specified with the keyword `Polygon`, and must consist of at least three points. There is always an implicit final connection from the last point specified to the first point specified. The following examples are equivalent:

```
Polygon poly = ([30,30]->[26,40]<-[4,50]<-[60,40])?
```

```
Point p = [30, 30]!
Polygon poly = (p->[26,40]<-[4,50]<-[60,40])?
```

Multi-lines are much like polygons, except that there is on implicit final connection between the last point and first point specified. Multi-line variables are specified with the keyword `Multiline`, as follows:

```
MultiLine m = ([30,30]->[26,40]<-[4,50]<-[60,40])?
```

Groups are a way of organizing lines. Groups do not honor sub-hierarchies. That is, even if a group has polygon or multi-line members, an operation on a group will treat the polygons and multi-lines as simply the lines of which they consist, with the appropriate endpoints and orientations. A group may be defined with the keyword `Group`, followed by a list of its members:

```
Line l = [30,40] -> [50,60]!
Polygon p = ([20,70] -> [30,10] <- [90,90])!
Group g = (l, p)!
```

Identifiers and Declarations
Specific instances of all data types can be declared using the keywords mentioned, followed by an identifier, =, and a dynamic version of the appropriate data type. Identifiers are any combinations of upper and lower case letters, digits and underbars starting with a letter.

Additional Declaration Examples

```
Point startPoint = [10,20]!
Line singleLine = startPoint -> [20, 30]!
Polygon Polly3 = startPoint -> [10, 50] -> [30, 30]!
Group mygroup = (myLine, myLine2)!
```

Scope
All variable declarations are valid with the { } separators in which they are defined. Declarations appearing at the top of a program (outside a {}) are valid throughout the entire program.

Control Flow

Conditional Statement

IpsoFracto supports the basic if statement. This includes a single conditional, an optional else and one or more statements for both conditions enclosed in `{}`. If using only single statements, the `{}` are optional.

Loops

IpsoFracto defines a looping function, `iter`. The keyword `iter` should be followed by a parenthesized block, followed by the name of a variable, followed by an open brace (`{`, required), an arbitrary number of statements, and a close brace (`}`, required). The parenthesized block should be of the form (`start .. end, step`), where `start`, `end`, and `step` are numeric values. Iter will repeat all statements inside the `iter` scope for values of the variable set in turn from `start` to `end` inclusive, (evaluated once at the beginning of the loop and not dynamically updated,) in steps of `step` size. The loop variable may be first instantiated in the `iter` command and it will be of valid scope throughout the `iter` execution. The values `end` and `step` may be left out and assumed to be 0 and 1, respectively.

Functions

There are two predefined functions in IpsoFracto: `apply` and `add`.

The `apply` function is used to apply a kink to a line and is the pivotal function of the language. The function requires that the user specify the following arguments, comma separated, in parentheses: position in the line of the kink (`CENTER`, `LEFT` or `RIGHT`), the orientation of the kink (`POS` or `NEG`), the kink to be applied, and the number of kinks applied "in parallel".

The `add` function is used simply to add members to a group. A group calls `add`, using the dot (`.`) operator, and the argument to the function is the object to be added to the group. This argument may be a line, polygon, multi-line, or another group.

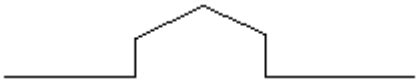
Kink Library

The initial library for IpsoFracto includes three kinks: `trikink`, `squarekink`, and `housekink`. These can be declared using the `Kink` keyword. Declarations for these kinks require two numbers, which indicate the ratios of the whole line to the height and width of the kink. The syntax is: `trikink (5, 5).`

The kinks look like:

trikink:

squarekink:

housekink: 

Syntax summary

See lp.g in the Appendix for more details.

<start rule> =>
‘#’ ‘`export`’ ‘″’ <string literal> ’″’ <statement>+ EOF

<statement> =>
(<int declaration>
| <float declaration>
| <boolean declaration>
| <point declaration >
| <line declaration >
| <polygon declaration >
| <multiline declaration >
| <group declaration >
| <add statement>
| <apply statement>
| <assignment>
| <postfix statement>)(‘!’ | ’?’)
| <if statement>
| <iter statement>

<int declaration> =>
‘`int`’ <int label> ‘=’
(<any number of numerals>
| <int label>
| <mathematical expression that evaluates to an int>)

<float declaration> =>
‘`float`’ <float label> ‘=’
(<any number of numerals> ‘.’ <any number of numerals>
| <float label>
| <mathematical expression that evaluates to a float>)

<boolean declaration> =>
‘`bool`’ <boolean label> ‘=’
(<boolean value>
| <boolean label>
| <expression that evaluates to a boolean>)

<label> =>
<alphanumeric string starting with a letter, but not reserved keyword>

<point declaration > =>
‘`Point`’ <point label> ‘=’ <point predicate>

<point predicate> =>
‘`[`‘ <number> ‘`,`’ <number> ‘`]`’

<number> =>
<value>|<number label>|<mathematical expression>

<value> =>
Any `int` or `float` value

<point> =>
<point label> | <point predicate>

<line declaration > =>
‘`Line`’ <label> ‘=’ <line predicate>

<line predicate> =>
<point> (‘`->`‘ | ‘`<-`‘) <point>

<line> =>
<line label> | <line predicate>

<polygon declaration > =>
‘`Polygon`’ <polygon label> ‘=’
<polygon predicate>

<polygon predicate> =>
<point> (‘`->`‘ | ‘`<-`‘) <point> ((‘`->`‘ | ‘`<-`‘) <point>)+ | <polygon label>

<multiline declaration > =>
‘`MultiLine`’ <multiline label> ‘=’ <multiline predicate>

<multiline predicate> =>
<point> (‘`->`‘ | ‘`<-`‘) <point> ((‘`->`‘ | ‘`<-`‘) <point>)+ | <multiline label>

<group declaration > =>
‘`Group`’ <group label> ‘=’ ‘`(`‘ <non-point label> (‘`,`’ <non-point label>)* ‘`)`’

<assignment> =>
<label> ‘=’ <appropriate predicate for label type>

<postfix statement> =>
<label> (‘++’ | ‘`--`')

<if statement> =>
‘if’ ‘(‘ <boolean expression> ‘)’ ‘{‘ <statement>+ ‘}’ (<else if>)* (<else>)?
|‘if’ ‘(‘ <boolean expression> ‘)’ <statement> (<else if>)* (<else>)?

<else> =>
‘else’ ‘{‘ <statement> +’}’|‘else’ <statement>

<else if> =>
‘else if’ ‘(‘<boolean expression> ’)’ ‘{‘ <statement> +’}’
|‘else if’ ‘(‘<boolean expression> ’)’ <statement>

<iter> =>
‘iter’ ‘(‘<number> ’..’ <number> ‘,’ <number>’)’ <number label> ‘{‘
<statement> + ‘}’
|‘iter’ ‘(‘<number> ’..’ <number> ‘,’ <number>’)’ <number label>
<statement>

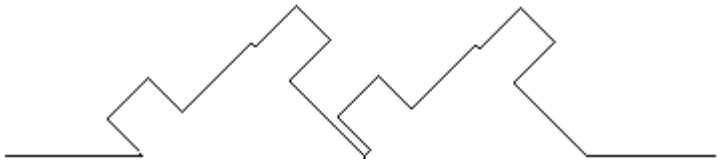
<apply statement> =>
‘apply’ ‘(‘ <line, poly, multi, or group label> ‘,’ <position constant> ‘,’
<pos or neg constant> ‘,’ <kink defined in libraries> ‘,’ <number> ‘)’

<add statement> =>
<group label> ‘.’ ‘add’ ‘(‘ <non-point label> ‘)’

Sample Program

```
#export “C:\Ipso\image.bmp”

Line myline = [20, 0] -> [50, 0]!
iter (2..0, -1) x {
     if ( x % 2 == 0 ) {
          apply( Thegroup, CENTER, POS,
                                   trikink(3,3), 2 )!
     } else {
          apply (Thegroup, LEFT, POS,
                                   squarekink(5,5), 1 )!
     }
}
```

Output:



Part IV

IpsoFracto Project Plan

Process

Our process for planning, specification, development, and testing was a mixture of democracy and meritocracy. We usually met at lunch on Tuesdays and Thursdays to discuss our progress and next steps. Ideas were brought to the table by a democratic process, and decisions were made based on the merit of the ideas.

When it came to development, we worked very closely together, often employing Extreme Programming techniques. At every milestone, we zipped and emailed the changed portions of the source out to the team. Since our code was well compartmentalized into packages and classes and each assignment was designed not to interfere with the other current work, we never had conflicts of change. We opted not to use CVS, as it was not necessary, and we had trouble setting it up.

Programming Style Guide

1. Keep all code as object-oriented and compartmentalized as possible.

2. Use standard indentation rules.

3. Where methods are complex, explain the intent in comments at the start of the code in question. It is unnecessary to follow the javadoc standard of listing parameters and outputs.

4. Begin all classes with comments covering the purpose of the class.

5. No standard is necessary for spacing or brackets.

6. Use simple, clear, and descriptive names for variables and methods.

Project Timeline

Date	*Task*	*Assigned To*
Mon., Sep. 27	Language Idea	All team members
Mon., Oct. 18	Full language specification	All team members
Mon., Nov. 1	Lexer	Rob, Leo
Thurs., Nov. 18	Support packages	Sarah, Anya
Thurs., Nov. 18	Parser	Rob, Leo
Thurs. Dec. 2	Tree walker	Rob, Leo
Thurs. Dec. 16	Test suite, testing, refining	Rob, Leo, Sarah
Tues., Dec. 21	Documentation	Anya

Team Member Roles and Responsibilities

Leo Gertsenshteyn: Lexer/Parser development, Tree Walker development, language specification

Sarah Gilman: Package design, support package development, language specification, project planning

Rob Notwicz: Lexer/Parser development, Tree Walker development, language specification

Anya Robertson: Documentation, some support package development, language specification, project planning

Development Environment

We developed IpsoFracto using Java v. 1.4.x and Antlr v. 2.7.4.

Project Log

Since we did not use CVS, we do not have a project log. All team members were always aware of what needed to be done, and what had been completed, because we kept strong verbal communication with each other.

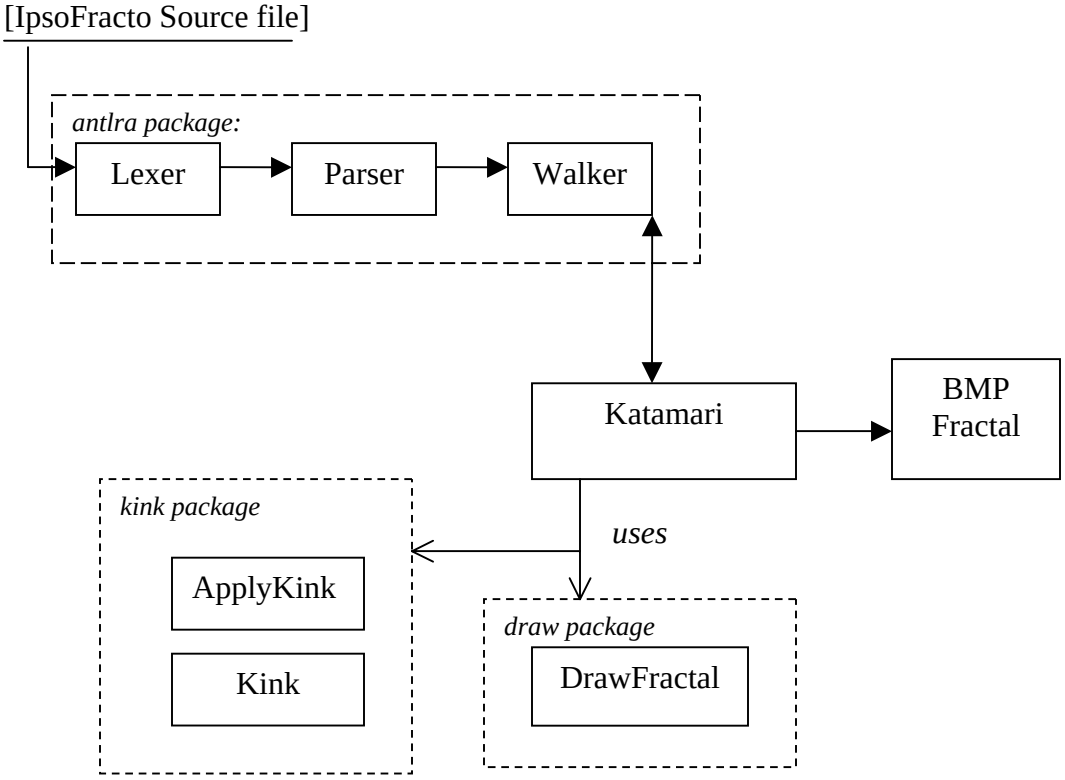
Part V

IpsoFracto Architectural Design

Overview

IpsoFracto consists of four source packages: basic, antlra, draw and kinky. An IpsoFracto source file is tokenized by the Lexer, then passed to to the Parser. The Parser transforms the program into an AST. The AST is then passed to the Walker, which checks for the proper program starting point. If the starting point is correct, the AST is passed to the Katamari class. Katamari walks the tree and performs most operations, relying on the kinky package for many transformations. Katamari calls on the TreeWalker for evaluating mathematical and boolean expressions. After the entire AST has been walked, Katamari calls the classes in the draw package to produce the final fractal design, which is displayed in a Swing Frame and stored in a bitmap file.

System Diagram



Detailed Package Descriptions

basic
The basic package is used by all other packages, but we did not include it here in the system diagram for simplicity's sake. The basic package contains Java classes for IpsoFracto's datatypes: Point, Line, MultiLine, and Polygon. For convenience, all objects except Point extend the LineGroup class, which can contain multiple lines. Only

LineGroups can be kinked or drawn, allowing for the proper separation of Points from the other types. Sarah Gilman was the primary developer of the basic package.

antlra
The antlra package contains the Lexer, Parser, and TreeWalker that the antlr tool generates from lp.g. It also contains our class Katamari, which is called from the TreeWalker to do most of the actual tree walking. As mentioned above, Katamari is called once the TreeWalker confirms the proper program format, by the code `Katamari.doEverything()`. Katamari then begins walking the AST and calls on functions of the TreeWalker only for evaluating mathematical and boolean expressions. Rob Notwicz and Leo Gerstenshteyn were the primary developers of the antlr file lp.g and Katamari.java.

draw
The draw package contains the code to display a LineGroup. The fractal image is shown in a Swing Frame and optionally stored in a bitmap file. The class for writing a Java Image object to a bitmap file was borrowed from http://www.javaworld.com/javaworld/javatips/jw-javatip60.html and written by Jean-Pierre Dubé. Anya Robertson was the primary developer for the remainder of the draw package.

kinky
The kinky package contains the code to apply a kink to a LineGroup as well as the code defining the kink types discussed in the LRM: trikink, squarekink, and housekink. Sarah Gilman was the primary developer of the kinky package.

Part VI

IpsoFracto Test Plan

The IpsoFracto fractal generation language was tested continually through the course of it's creation.

I. The first phase consisted largely of printing and inspecting ASTs built by the ANTLR-generated parser code, to verify proper parsing. Once everything was parsing properly and in the desired tree form, testing of the tree walker could begin.

II. The tree walker was verified by running test programs. Since IpsoFracto is an interpreted language, and not a compiled one, there was no target language code to inspect. We simply had to verify that a program would produce the expected result. There is no lower bound on the complexity of a valid IpsoFracto program, excepting the inclusion of an `export` directive. This allowed the team to run single lines of code at a time to check that:

- all variable declarations work when they should and don't when they shouldn't
- mathematical expressions produce the expected output
- boolean expressions similarly produce the expected output
- mathematical and boolean statements work together as designed
- assignments executed properly
- calls to `apply` and `add` worked in the expected manner
- conditional and iterative blocks executed as expected

For a short example, the following code was used to test mathematical and boolean operations and to ensure they play nicely together:

```
#export "C:\\ANTLR\\11thTEST\\duh.bmp"

int i = 1+2!
int j = i+1!
int k = 5*j!
int l = k || false!
int m = l && false!
int n = l && (2-3)!
int o = (k == 20) && (i <= 7)!
```

Refer to testA.txt below for our test program for more assignments and apply statements. Refer to testB.txt below for the program we used to test iter statements.

III. Full programs were created as examples of what sort of data structures and algorithms could be combined in IpsoFracto to produce interesting results. In this process, deeper bugs came to light and were resolved.

```
/For starters, this should give us and error, and does.

bonk


================================================================================
//     this program runs through a lot of declaration basics

#export "C:\\testdecs.bmp"

// '?' means that it should cause an error
int i=9!
int i2=8.76?
int i3=true?

float f=5.4!
float f2= 70?
float f3=true?

bool b= true!
bool b2= false!
bool b3= 7893.57!


Point PpA = [500, 650]?

Line LA = ([150,80]->[190,500])!

MultiLine MLA = ([160, 30]->[220, 60]->[185, 70]<-[210,5]->[240,45]-
>[750, 40])?

Polygon PgA = ([370, 620]->[870, 620]->[870, 220]->[370,220])?

Polygon PgB = (PpA -> PpA -> PpA)!


Group GA = ()?

Group GB = (MLA)!

Group GC = (LA, MLA, PgA, GA)?

Group GD = (GB, GA, GC)!

================================================================================
// this program extends the last one to run through a lot of
//application basics
// just to show that they work in the general case so that testing does
// not need to be as rigorous in the next phases
#export "C:\\testapply.bmp"

Point PpA = [500, 650]?
Line LA = ([150,80]->[190,500])!
MultiLine MLA = ([160, 30]->[220, 60]->[185, 70]<-[210,5]->[240,45]-
>[750, 40])?
Polygon PgA = ([370, 620]->[870, 620]->[870, 220]->[370,220])?
Polygon PgB = (PpA -> PpA -> PpA)!
```

```
Group GA = ()?
Group GB = (MLA)!
Group GC = (LA, MLA, PgA, GA)?
Group GD = (GB, GA, GC)!

//apply(PgA, CENTER, POS, trikink(2,2), 4)?
//apply(PgA, CENTER, POS, trikink(2,2), 2)?
//apply(PgA, CENTER, POS, trikink(2,2), 20)!
//apply(PgA, CENTER, POS, trikink(2,2), 1)!
//apply(PgA, CENTER, POS, trikink(5,2), 1)!
//apply(PgA, CENTER, POS, trikink(300,6), 2)!
//apply(PgA, CENTER, POS, squarekink(5,1), 3)!
//apply(PgA, CENTER, POS, squarekink(1,1), 1)!    // this one is kinda
cool if you think about it
//apply(PgA, CENTER, POS, housekink(5,1), 3)!
//apply(PgA, CENTER, POS, squarekink(1,1), 17)!   // this one is
pushing it a bit
//apply(PgA, CENTER, POS, housekink(1,1), 17)!   // just think about it
for a sec
//apply(PgA, CENTER, POS, squarekink(5,1), 300)!  // ha ha ha... makes
a solid block around the rectangle
//apply(PgA, CENTER, POS, trikink(2,2), 2)!
//apply(PgA, CENTER, NEG, trikink(2,2), 2)!
//apply(PgA, CENTER, POS, squarekink(2,2), 2)!
//apply(PgA, CENTER, NEG, squarekink(2,2), 2)!
//apply(PgA, RIGHT, POS, trikink(2,2), 1)!
//apply(PgA, LEFT, POS, trikink(4,4), 1)!
//apply(PgA, RIGHT, NEG, trikink(2,2), 4)!
//apply(PgA, LEFT, NEG, trikink(2,2), 4)!
//apply(PgB, CENTER, POS, trikink(1,1), 1)!

//apply(LA, CENTER, POS, trikink(2,2), 4)?
//apply(LA, CENTER, POS, trikink(2,2), 2)?
//apply(LA, CENTER, POS, trikink(2,2), 20)!
//apply(LA, CENTER, POS, trikink(2,2), 1)!
//apply(LA, CENTER, POS, trikink(5,2), 1)!
//apply(LA, CENTER, POS, trikink(300,6), 2)!
//apply(LA, CENTER, POS, squarekink(5,1), 3)!
//apply(LA, CENTER, NEG, squarekink(1,1), 1)!
//apply(LA, CENTER, POS, housekink(5,1), 3)!
//apply(LA, CENTER, NEG, squarekink(1,1), 17)!
//apply(LA, CENTER, NEG, housekink(1,1), 17)!
//apply(LA, CENTER, POS, squarekink(5,1), 300)!
//apply(LA, CENTER, POS, trikink(2,2), 2)!
//apply(LA, CENTER, NEG, trikink(2,2), 2)!
//apply(LA, CENTER, POS, squarekink(2,2), 2)!
//apply(LA, CENTER, NEG, squarekink(2,2), 2)!
//apply(LA, RIGHT, POS, trikink(2,2), 1)!
//apply(LA, LEFT, POS, trikink(4,4), 1)!
//apply(LA, RIGHT, NEG, trikink(2,2), 1)!
//apply(LA, LEFT, NEG, trikink(4,4), 1)!

//apply(MLA, CENTER, POS, trikink(2,2), 4)?
//apply(MLA, CENTER, POS, trikink(2,2), 2)?
//apply(MLA, CENTER, POS, trikink(2,2), 20)!
//apply(MLA, CENTER, POS, trikink(2,2), 1)!
```

```
//apply(MLA, CENTER, POS, trikink(5,2), 1)!
//apply(MLA, CENTER, POS, trikink(300,6), 2)!
//apply(MLA, CENTER, POS, squarekink(5,1), 3)!
//apply(MLA, CENTER, NEG, squarekink(1,1), 1)!
//apply(MLA, CENTER, POS, housekink(5,1), 3)!
//apply(MLA, CENTER, NEG, squarekink(1,1), 17)!
//apply(MLA, CENTER, NEG, housekink(1,1), 17)!
//apply(MLA, CENTER, POS, squarekink(5,1), 300)!
//apply(MLA, CENTER, POS, trikink(2,2), 2)!
//apply(MLA, CENTER, NEG, trikink(2,2), 2)!
//apply(MLA, CENTER, POS, squarekink(2,2), 2)!
//apply(MLA, CENTER, NEG, squarekink(2,2), 2)!
//apply(MLA, RIGHT, POS, trikink(2,2), 1)!
//apply(MLA, LEFT, POS, trikink(4,4), 1)!
//apply(MLA, RIGHT, NEG, trikink(2,2), 1)!
//apply(MLA, LEFT, NEG, trikink(4,4), 1)!

//apply(GB, CENTER, POS, trikink(300,6), 2)!  //Group GB = (MLA)!
//apply(GB, CENTER, POS, squarekink(5,1), 3)!
//apply(GB, CENTER, POS, housekink(5,1), 3)!
//apply(GB, CENTER, POS, squarekink(5,1), 300)!
//apply(GB, RIGHT, POS, trikink(2,2), 1)!
//apply(GB, LEFT, NEG, trikink(4,4), 1)!

//apply(GC, CENTER, POS, trikink(300,6), 2)!  //Group GC = (LA, MLA,
PgA, GA)?
//apply(GC, CENTER, POS, squarekink(5,1), 3)!
//apply(GC, CENTER, POS, housekink(5,1), 3)!  // this one's a keeper!
//apply(GC, CENTER, POS, squarekink(5,1), 300)!
//apply(GC, RIGHT, POS, trikink(2,2), 1)!
//apply(GC, LEFT, NEG, trikink(4,4), 1)!

//apply(GD, CENTER, POS, trikink(300,6), 2)!  Group GD = (GB, GA, GC)!
//apply(GD, CENTER, POS, squarekink(5,1), 3)!
//apply(GD, CENTER, POS, housekink(5,1), 3)!
//apply(GD, CENTER, POS, squarekink(5,1), 300)!
//apply(GD, RIGHT, POS, trikink(2,2), 1)!
//apply(GD, LEFT, NEG, trikink(4,4), 1)!

========================================================================
//This code runs through ifs and iters to see that they work.
#export "C:\\testif.bmp"

bool b= true!
bool b2= false!
bool b3= 7893.57!

//if(){
//}

if(b){
//    print "b is true!"!
}

if(~b){
    print "b is false"!
}
```

```
iter(1 .. i2, 1) m{
    print "again!"!
}

//iter(){
//}

//iter(1 ..){
//}

//iter(1 .. 4){
//}

//iter(1 .. 1, 1){
//}

iter(1 .. f, 1)m{
//     print "floating"!
}

iter(1 .. i, 2) m{
//    print "finally"!
}

iter(1 .. i3, 1) m{
//      print "and yet again!"!
}

iter(1 .. i2, 1) m{
//    print "again!"!
}

iter(1 ..i, 1) m{
//     print "here we go!"!
}

iter(1 .. 2, 1) m{
//    apply(GA, CENTER, NEG, trikink(3,3), 1)!
}

iter(1 .. 3, 1) m{
    apply(GA, CENTER, NEG, trikink(3,3), m)!
}

iter(1 .. 3, 1) m{
    if(b && b2){
//    apply(GA, CENTER, NEG, trikink(3,3), m)!
    }
}

iter(0 .. b, 1) m{
    if(m){
//    apply(GA, CENTER, NEG, trikink(3,3), b+1)!
    }
```

```
}

iter(-2 .. 3, 1) m{
    if(3+4==7 && m~=0){
//      print "3+4 equals 7"!
//      apply(GA, LEFT, POS, squarekink(m,m), 1)!
    }
}

iter(1 .. 3, 1) m{
    iter(1 .. 4, 1) n{
//        apply(GA, LEFT, POS, squarekink(m,m), 1)!
    }
//    apply(GA, LEFT, POS, squarekink(n,n), 1)!
}

=========================================================================
//This test code checks to make sure kinks apply through entire groups
correctly:

#export "C:\\test.bmp"

Line LineA = [140,140]->[385,140]!
Line LineB = ([385, 140]->[385, 385])!
Line LineC = [385, 385]->[140, 385]?
Line LineD = [140, 385]->[140, 140]?
Group GroupA = (LineA)!
GroupA.add(LineB)!
GroupA.add(LineC)!
GroupA.add(LineD)!

int k= 7!

Point PointA = [0,0]!
Point PointB = [70, 100]!
Line LineE = PointB -> [3,55] !
Line LineF = LineE!


iter(1 .. 4, 1) i{
    iter(1 .. 3, 1) j{
        if(i*j<=4){
            apply(theGroup, CENTER, NEG, squarekink (2*i,i+3), j/2+1)!
        }
        else{
            apply(GroupA, CENTER, POS, trikink (i+1,1), i)!
        }
    }
}

=========================================================================
//The following code examines how housekinks end up composing

#export "C:\\ANTLR\\nameforit.bmp"

int k=3?
```

```
Line LineA = [k, 100]-> [100, k]!

Line LineB = [1,1] <- [1,1]!

LineB = LineA?

Polygon PolyA = ([300, 300]->[260, 400]<-[40, 500]<-[600, 400])?

Polygon PolyB = PolyA!

Group GroupA = (LineA, LineB, PolyA)!

iter(1 .. k, 1) i{
    apply(GroupA, RIGHT, POS, housekink(2,2), 1)!
}
```

```
==========================================================================
//This code explores the adorable side of fractals.
#export "C:\\face.bmp"

Polygon faceBox= ([20,20]->[20,370]->[370,370]->[370,20])?
Line eye1= ([70,120]->[170,120])!
Line eye2= ([220,120]->[320,120])!
Line nose= ([170,220]->[220,220])!
Line smile= ([70,270]<-[320,270])!

apply(eye1, CENTER, POS, trikink(5,5), 1)!
apply(eye2, CENTER, POS, trikink(5,5), 1)?
apply(nose, CENTER, POS, trikink(5,5), 1)!
apply(smile, CENTER, NEG, trikink(8,1), 1)!
```

Part VII

IpsoFracto Lessons Learned

Our Advice to Future Teams

As a team, we stress to future teams to start early. Designing and implementing a programming language is a complicated undertaking. It may seem to be a simple project upon first look, but it is much larger than it seems. We suggest that future teams fully understand the project early and make realistic deadlines and stick to them. We also found it quite useful to make the project very modular. That is, having clear divisions between classes and packages is extraordinarily helpful in maintaining clean code and a clear project.

Team Member Quotes

Leo Gertsenshteyn
“It's always better to be clever than to cross your fingers and hope a kludge works when it comes to parsing.”

Sarah Gilman
“Having clear specifications for the interfaces between modules is more important than it seems at first.”

Rob Notwicz
“Coding is much easier when you can pass off work to other classes at will. Object-Oriented plus Encapsulated.”
"fractal am yummy fun"

Anya Robertson
“Pick your project partners well, work closely with them, and be flexible. Communication is key.”

Part VIII

IpsoFracto Appendix: Code Listing

Basic Package:

lp.g

```
header { package antlra;
}
class IFLexer extends Lexer;

options {
        k=3;
       charVocabulary='\u0000'..'\u007F';
}

protected ALPHA : ( 'a'..'z' | 'A'..'Z' | '_') ;
protected DIGIT  : '0'..'9' ;

AR :                "->";
AL :                "<-";
COMMA:                ',';
YADA:                "..";
PERIOD:               '.';
MODULUS:            '%';
STAR:               '*';
SOLIDUS:           '/';
PLUS:               '+';
MINUS:                '-';
AND:               "&&";
OR:               "||";
NOT:               '~';
EQ:              "==";
NE:              "~=";
LE:              "<=";
GE:              ">=";
LT:              '<';
GT:              '>';
ASS:               '=';
INC:               "++";
DEC:                "--";
LPAREN:                   '(';
RPAREN:                   ')';
LBRACKET:  '{';
RBRACKET:  '}';
LSQBRACKET:          '[';
RSQBRACKET:          ']';
BANG:                     '!' | '?';
HASH:          '#';
DQUOT:                    '"';

WS:
         (' '|'\t')+ {$setType(Token.SKIP);}
        ;

NL:
         ('\n' | ('\r' '\n') => '\r' '\n' | '\r') {$setType(Token.SKIP); newline();}
        ;

STRINGLIT:
```

```
            DQUOT! (~(""))* DQUOT!  //"
          ;

LABEL:
        ALPHA (ALPHA|DIGIT)*
      ;

INTEGER:
   (DIGIT)+
   (
   ( PERIOD (DIGIT)+ { $setType(FLOAT); }
   )
   | /* empty */ //{ $setType(Token.Integer); }
   )
   ;

FLOAT:
   PERIOD (DIGIT)+ //{ $setType(DoubleConst); }
   ;


SLCOMMENT:
        (SOLIDUS SOLIDUS (
                (~('\n'|'\r'|'\3'|'\4'))* {$setType(Token.SKIP);}
                ) ((NL)=>NL | /*Nothing, here comes the EOF*/))
        ;

MULTICOMM:
( "/*"
( { LA(2)!='/' }? '*'   //this requires a lookahead of 3
| ('\n'|("\r" "\n")=>"\r" "\n" |'\r') { newline(); }
| ('\3'..'\11'|'\16'..'\51'|'+'..'\177')  //ALSO, no Form Feed or VERT TAB!
)*
  "*/"
){ $setType(Token.SKIP); }
;


////////////////////////////////////////////////////////////////////////
//            PARSER
////////////////////////////////////////////////////////////////////////

class IFParser extends Parser;
options {
   buildAST = true; // Enable AST building
   k = 2;
}

startRule:
        HASH^ "export"! STRINGLIT program EOF!
;
          exception
          catch [RecognitionException ex] {
                System.err.println("flagrant error: 2 free throws and a turnover");
                reportError(ex.toString());
                System.exit(0);
```

```
        }

program:
        (stmt)*
      ;
        exception
        catch [RecognitionException ex] {
                System.err.println("Error reading program start");
                reportError(ex.toString());
                Katamari.parseProblems=true;
        }

schroe:
        LABEL
      | INTEGER
      | FLOAT
      ;
        exception
        catch [RecognitionException ex] {
                System.err.println("Error reading in a value");
                reportError(ex.toString());
                Katamari.parseProblems=true;
        }

stmt:
        declaration BANG!
        | action BANG!
        | question
        ;
        exception
        catch [RecognitionException ex] {
                System.err.println("Error reading this statement");
                reportError(ex.toString());
                Katamari.parseProblems=true;
        }

affirm:
        declaration BANG!
        | action BANG!
        | iter
        ;
        exception
        catch [RecognitionException ex] {
                System.err.println("Error reading this statement");
                reportError(ex.toString());
                Katamari.parseProblems=true;
        }

block:
        LBRACKET!
        (stmt)*
        RBRACKET!
      |
        stmt
      ;
        exception
```

```
            catch [RecognitionException ex] {
                  System.err.println("Error reading this block");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

    ablock:
          LBRACKET!
          (stmt)*
          RBRACKET!
        |
          affirm
        ;
            exception
            catch [RecognitionException ex] {
                  System.err.println("Error reading this block");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

    declaration:
           "Line" (LABEL ASS^ (linep | LPAREN! linep RPAREN! | LABEL))
         | "Point" LABEL ASS^ (pointp | LABEL)
         | "Group" LABEL ASS^ (LPAREN! (( LABEL COMMA! )* LABEL | /* nothing */) RPAREN! |
LABEL)
         | "MultiLine" LABEL ASS^ (LPAREN! mlp RPAREN! | LABEL)
         | "Polygon" LABEL ASS^ (LPAREN! mlp RPAREN! | LABEL)
         | "int" LABEL ASS^ mstmt
         | "float" LABEL ASS^ mstmt
         | "bool" LABEL ASS^ mstmt
         ;
            exception // for rule
            catch [RecognitionException ex] {
                  System.err.println("Error reading this declaration");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

    assignment:
          LABEL ASS^ atail
        ;
            exception
            catch [RecognitionException ex] {
                  System.err.println("Error reading this assignment");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

    atail:
          (LPAREN! at2) => (LPAREN! at2)
         | pointp ((AR|AL) (pointp | LABEL))?
         | LABEL (AR|AL) (pointp | LABEL)
         | mstmt
        ;
            exception
            catch [RecognitionException ex] {
```

```
                System.err.println("Error reading this assignment");
                reportError(ex.toString());
                Katamari.parseProblems=true;
            }

//at2 is good?
    at2:
         pointp ((AR|AL) (pointp | LABEL)) ( (AR|AL) (pointp | LABEL))+ RPAREN!
        | LABEL (AR|AL) (pointp | LABEL) ( (AR|AL) (pointp | LABEL))+ RPAREN!
        |(( LABEL COMMA! )* LABEL | /* nothing*/)RPAREN!
        ;
            exception
            catch [RecognitionException ex] {
                System.err.println("Error reading this assignment");
                reportError(ex.toString());
                Katamari.parseProblems=true;
            }

    booleannj: "true" | "false"
          ;
            exception
            catch [RecognitionException ex] {
                System.err.println("Error reading this declaration");
                reportError(ex.toString());
                Katamari.parseProblems=true;
            }

    pointp:
          LSQBRACKET! mstmt COMMA^ mstmt RSQBRACKET!
          ;
            exception
            catch [RecognitionException ex] {
                System.err.println("Error reading this declaration");
                reportError(ex.toString());
                Katamari.parseProblems=true;
            }

    linep:
           (pointp | LABEL) (AR|AL) (pointp | LABEL)
          ;
            exception
            catch [RecognitionException ex] {
                System.err.println("Error reading this declaration");
                reportError(ex.toString());
                Katamari.parseProblems=true;
            }

    mlp:
           linep ( (AR|AL) (pointp | LABEL))+
          ;
            exception // for rule
            catch [RecognitionException ex] {
                System.err.println("Error reading this declaration");
                reportError(ex.toString());
                Katamari.parseProblems=true;
            }
```

```
// the schroes above can be anything

action:
        add
      | apply
      | post
      | assignment
      | "print"^ STRINGLIT
      ;
         exception
         catch [RecognitionException ex] {
               System.err.println("Error reading this action");
               reportError(ex.toString());
               Katamari.parseProblems=true;
         }

question:
      iter
      | ifelse
      ;
         exception
         catch [RecognitionException ex] {
               System.err.println("Error reading this question");
               reportError(ex.toString());
               Katamari.parseProblems=true;
         }

ifelse:
      (
       "if"^ LPAREN! mstmt RPAREN!
            block
      (options {greedy=true;}:
         ("else" ifelse )
      )*
      ( options {greedy=true;}:
      "else" ablock )?
      )
      ;
         exception
         catch [RecognitionException ex] {
               System.err.println("Error reading if/else");
               reportError(ex.toString());
               Katamari.parseProblems=true;
         }

/*
 or whatever other constants we wish to define here which shouldn't be taken as labels
*/

iter:
        "iter"^ LPAREN! mstmt YADA! mstmt COMMA! mstmt RPAREN! LABEL
            block
      ;
         exception
```

```
        catch [RecognitionException ex] {
            System.err.println("Error reading iteration");
            reportError(ex.toString());
            Katamari.parseProblems=true;
        }

add:
        LABEL PERIOD! "add"^ LPAREN! LABEL RPAREN!
      ;
        exception
        catch [RecognitionException ex] {
            System.err.println("Error reading add statement");
            reportError(ex.toString());
            Katamari.parseProblems=true;
        }

apply:
        "apply"^ LPAREN! LABEL COMMA! position COMMA! orientation COMMA! kinkp
COMMA! mstmt RPAREN!
      ;
        exception
        catch [RecognitionException ex] {
            System.err.println("Error reading apply statement");
            reportError(ex.toString());
            Katamari.parseProblems=true;
        }

position:
            "RIGHT"
          | "LEFT"
          | "CENTER"
          ;
        exception
        catch [RecognitionException ex] {
            System.err.println("Error reading position value");
            reportError(ex.toString());
            Katamari.parseProblems=true;
        }

orientation:
            "POS"
          | "NEG"
          ;
        exception
        catch [RecognitionException ex] {
            System.err.println("Error reading orientation value");
            reportError(ex.toString());
            Katamari.parseProblems=true;
        }

// the schroe above need not be an integer

kinkp:
        kinkname LPAREN! mstmt COMMA! mstmt RPAREN!
      ;
        exception
```

```
        catch [RecognitionException ex] {
              System.err.println("Error reading kink definition");
              reportError(ex.toString());
              Katamari.parseProblems=true;
        }

// the mstmts above need to evaluate to integers

kinkname:
              "trikink"
            | "squarekink"
            | "housekink"
            ;
        exception
        catch [RecognitionException ex] {
              System.err.println("Error reading kink type");
              reportError(ex.toString());
              Katamari.parseProblems=true;
        }

//---------------------------------
//DEFINING MATHEMATICAL EXPRESSIONS
//---------------------------------


mstmt:(
   options { greedy=true;}:
     b2 (AND^ mstmt | OR^ mstmt)?
   )
   ;
        exception
        catch [RecognitionException ex] {
              System.err.println("Error reading math/boolean statment");
              reportError(ex.toString());
              Katamari.parseProblems=true;
        }

b2:(
   options { greedy=true;}:
     b3 (  LE^ b3
    | GE^ b3
    | LT^ b3
    | GT^ b3
    | NE^ b3
    | EQ^ b3)?
   )
   ;
        exception
        catch [RecognitionException ex] {
              System.err.println("Error reading math/boolean statment");
              reportError(ex.toString());
              Katamari.parseProblems=true;
        }

b3:(
   options { greedy=true;}:
```

```
        b4 (PLUS^ b3 | MINUS^ b3)?
        )
      ;
            exception
            catch [RecognitionException ex] {
                  System.err.println("Error reading math/boolean statment");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

   b4:(
      options { greedy=true;}:
        b5 (STAR^ b5  | SOLIDUS^ b5 | MODULUS^ b5)*
        )
      ;
            exception
            catch [RecognitionException ex] {
                  System.err.println("Error reading math/boolean statment");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

   b5:
         booleannj  // "true" or "false"
      | post
      | schroe
      | LPAREN! mstmt RPAREN!
      | NOT^ b5
      | MINUS^ b5
      ;
            exception
            catch [RecognitionException ex] {
                  System.err.println("Error reading math/boolean statment");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

   post:
         LABEL (INC^ | DEC^)
      ;
            exception
            catch [RecognitionException ex] {
                  System.err.println("Error reading math/boolean statment");
                  reportError(ex.toString());
                  Katamari.parseProblems=true;
            }

   ////////////////////////////////////////////////////////////////////
   //            TREE PARSER
   ////////////////////////////////////////////////////////////////////

class IFTreeWalker extends TreeParser;

startRule:
      #(T:HASH .) {
            if(Katamari.parseProblems){
```

```
            System.err.println("There have been parsing problems, quitting");
            System.exit(0);
          }
          Katamari.doEverything(T);}
    ;

mstmt returns [double e]
  {double x,y; e=0.0;}:
      #(AND x=mstmt y=mstmt){e = (x!=0 && y!=0)? 1 : 0; }
    | #(OR x=mstmt y=mstmt) {e = (x!=0 || y!=0)? 1 : 0; }
    | #(NOT x=mstmt) {e = (x!=0) ? 0 : 1 ;}
    | #(m:MINUS .) {e = Katamari.doMinus(m);}
    | #(LE x=mstmt y=mstmt) {e= (x<=y)?(1):(0);}
    | #(LT x=mstmt y=mstmt) {e= (x<y)?(1):(0);}
    | #(GE x=mstmt y=mstmt) {e= (x>=y)?(1):(0);}
    | #(GT x=mstmt y=mstmt) {e= (x>y)?(1):(0);}
    | #(EQ x=mstmt y=mstmt) {e= (x==y)?(1):(0);}
    | #(NE x=mstmt y=mstmt) {e= (x!=y)?(1): (0);}
    | #(PLUS x=mstmt y=mstmt){e=x+y; System.out.println("in mstmt plus: "+ e);}
   // | #(MINUS x=mstmt y=mstmt){e=x-y; System.out.println("in mstmt minus: "+ e);}
    | #(STAR x=mstmt y=mstmt){e=x*y; System.out.println("in mstmt star: "+ e);}
    | #(SOLIDUS x=mstmt y=mstmt){e=x/y; System.out.println("in mstmt solidus: "+ e);}
    | #(MODULUS x=mstmt y=mstmt){e=x%y; System.out.println("in mstmt modulus: "+ e);}
    | g:INTEGER {e= Integer.parseInt(g.getText());}
    | h:FLOAT {e= Double.parseDouble(h.getText());}
    | i:"true" {e=1;}
    | ii:"false" {e=0;}
    | j:LABEL {Object o = Katamari.symbol_table.get(j.getText());
       if(o instanceof Integer) {e=((Integer)o).doubleValue();}
       else if(o instanceof Float) {e=((Float)o).doubleValue();}
       else if(o instanceof Double) {e=((Double)o).doubleValue();}
       else{
          System.err.println("Halting execution: "
                +j.getText()+" is not defined as a numerical or boolean value.");
          System.exit(1);
          }
       }
      | #(INC a:LABEL){Object o = Katamari.symbol_table.get(a.getText());
       //POST INCREMENT
       if(o instanceof Integer) {e=((Integer)o).doubleValue();
            Katamari.symbol_table.put( a.getText(), new Integer((int)e+1) );}
       else if(o instanceof Float) {e=((Float)o).doubleValue();
            Katamari.symbol_table.put( a.getText(), new Float(e+1) );}
       else if(o instanceof Double) {e=((Double)o).doubleValue();
            Katamari.symbol_table.put( a.getText(), new Double(e+1) );}
       else{
          System.err.println("Halting execution: "
                +a.getText()+" is not defined as a numerical or boolean value.");
          System.exit(1);}
      }
      | #(DEC b:LABEL){Object o = Katamari.symbol_table.get(b.getText());
       //POST DECREMENT
       if(o instanceof Integer) {e=((Integer)o).doubleValue();
            Katamari.symbol_table.put( b.getText(), new Integer((int)e-1) );}
       else if(o instanceof Float) {e=((Float)o).doubleValue();
            Katamari.symbol_table.put( b.getText(), new Float(e-1) );}
```

```
  else if(o instanceof Double) {e=((Double)o).doubleValue();
        Katamari.symbol_table.put( b.getText(), new Double(e-1) );}
  else{
     System.err.println("Halting execution: "
          +b.getText()+" is not defined as a numerical or boolean value.");
     System.exit(1);}
}
```

```
/*
 * Katamari.java
 *
 * This is the main method of IpsoFracto. It contains static methods needed for
 * walking the AST, keeping a symbol table of variables and making all calls
 * to the kink and draw packages.
 */
package antlra;

import antlr.CommonAST;
import antlr.collections.AST;
import java.io.*;
import java.util.Vector;
import java.util.HashMap;
import basic.*;
import kinky.*;
import draw.*;

public class Katamari implements IFLexerTokenTypes{
   static boolean DEBUG = true;
   static boolean parseProblems=false;
   static HashMap symbol_table = new HashMap();
   static IFTreeWalker walker = new IFTreeWalker();
   static String theGroup_symbol = "theGroup";

   /**
    * This method handles a declaration or assignment node.  Based on the
    * keyword typs of the assignment, calls submethod or just builds object.
    * New object is then stored in the symbol table.
    */
   static void declare(AST root) throws antlr.RecognitionException, KatamariException {
      int num=0;
      Object newobject = null;
      String currentText;
      if(root.getFirstChild().getNextSibling()==null){
         if(symbol_table.containsKey(root.getFirstChild().getText())){
            currentText= root.getFirstChild().getText();
            newobject = readGroup(root.getFirstChild());
         }
         else
            throw new KatamariException(root.getFirstChild().getText()+" invalid assignment");
      }
      else{
         currentText=root.getFirstChild().getNextSibling().getText();
         AST current=root.getFirstChild().getNextSibling().getNextSibling();
         println("# Declaring "+currentText+" as (type) "+ root.getFirstChild().getText());
         switch(root.getFirstChild().getType()){
            case LITERAL_int:
               newobject = new Integer( (int) walker.mstmt(current) );
               break;
            case LITERAL_float:
               newobject = new Float( (float) walker.mstmt(current) );
               break;
            case LITERAL_bool:
               newobject = new Double( walker.mstmt(current) );
               break;
```

```
case LITERAL_Line:
   newobject = readLine( current );
   break;
case LITERAL_Polygon:
   num= (root.getNumberOfChildren()-1)/2;
   newobject = readMultiline( current, num, true );
   break;
case LITERAL_MultiLine:
   num= (root.getNumberOfChildren()-1)/2;
   newobject = readMultiline( current, num, false );
   break;
case LITERAL_Point:
   newobject = readPoint(current);
   break;
case LITERAL_Group:
      current = root.getFirstChild().getNextSibling();
      System.out.println("the current group node fed in" +current.getText());
      newobject = readGroup(current);
   break;
default:
   //the first child is not the name of a type
   //must be an assignment
   String gname = root.getFirstChild().getText();
   current=root.getFirstChild().getNextSibling();
   currentText= gname;
   Object o = symbol_table.get(gname);
   if (o == null) {
      throw new KatamariException( "cannot assign value to"
         +gname +" -- cannot resolve." );
   }
   else if (o instanceof Point){
      newobject = readPoint(current);
   }
   else if (o instanceof Integer){
      newobject = new Integer((int) walker.mstmt(current));
   }
   else if (o instanceof Float){
      newobject = new Float((float) walker.mstmt(current));
   }
   else if (o instanceof Double){
      newobject = new Double(((double) walker.mstmt(current)));
   }
   else if (o instanceof SingleLine){
      newobject = readLine( current );
   }
   else if (o instanceof MultiLine){
      num= (root.getNumberOfChildren()-1)/2;
      newobject = readMultiline( current, num, false );
   }
   else if (o instanceof Polygon){
      num= (root.getNumberOfChildren()-1)/2;
      newobject = newobject = readMultiline( current, num, true );
   }
   else if (o instanceof Group){
      current = root.getFirstChild().getNextSibling();
      newobject = readGroup(current);
```

```
            }
        }
    }
    println("  == "+ newobject );
    symbol_table.put(currentText, newobject);
    if( newobject instanceof LineGroup ) {
        ((Group)symbol_table.get(theGroup_symbol)).addObject(currentText);
    }
    else if( newobject instanceof Group ){
        ((Group)symbol_table.get(theGroup_symbol)).addObject(currentText);
    }
}

/**
 * Expects a node equivalent to the "," and returns the Point represented
 * by the children
 */
static Point readPoint( AST node ) throws antlr.RecognitionException, KatamariException{
    Point p;
    if( node.getType() == LABEL ) {
        p = new Point((Point) symbol_table.get(node.getText()));
    }
    else if(node.getNumberOfChildren()<2){
        throw new KatamariException("this is not a point: "+ node.getText());
    }
    else{
        double x = walker.mstmt(node.getFirstChild());
        double y = walker.mstmt(node.getFirstChild().getNextSibling());
        p = new Point(x,y);
    }
    return p;
}

/**
 * Expects a node equivalent to the first Point in a line (or a label which
 * maps to a known point) and returns the Line represented by this node and
 * its siblings.
 */
static SingleLine readLine( AST node ) throws antlr.RecognitionException, KatamariException{
    SingleLine newline;
    if( node.getType() == LABEL && node.getNextSibling()==null) {
        newline = new SingleLine((basic.SingleLine) symbol_table.get(node.getText()));
    } else {
        Point p1 = readPoint(node);
        Point p2 = readPoint(node.getNextSibling().getNextSibling());
        if( node.getNextSibling().getType() == AR ) {
            newline = new SingleLine( p1, p2 );
        } else {
            newline = new SingleLine( p2, p1 );
        }
    }
    return newline;
}

/**
 * Expects a node equivalent to the first Point in a line (or a label which
```

```
 * maps to a known point) and returns the Polygon represented by this node
 * and its siblings.
 */
static LineGroup readMultiline( AST node, int points, boolean isPolygon )
throws antlr.RecognitionException, KatamariException, ClassCastException {

   LineGroup newgroup = null;
   if( node.getType() == LABEL ) {
      Object o = symbol_table.get(node.getText());
      if (o instanceof Point){
         Line[] newlines = new Line[points-1];
         Point start = null, end = readPoint(node);
         for( int l=0; l<(points-1); l++ ) {
            Point p1 = readPoint(node);
            Point p2 =  readPoint(node.getNextSibling().getNextSibling());
            if( node.getNextSibling().getType() == AR ) {
               newlines[l] = new Line( p1, p2 );
            } else {
               newlines[l] = new Line( p2, p1 );
            }
            node = node.getNextSibling().getNextSibling();
            start = p2;
         }
         if( isPolygon ) { newgroup = new Polygon(newlines, start, end); }
         else { newgroup = new MultiLine(newlines); }
      }
      else if( isPolygon){
                  if(o instanceof Polygon) {
            newgroup = new Polygon((basic.Polygon) o);
         }
         else{
            throw new KatamariException(node.getText() + " is not a Polygon");
         }
      }
      else if(!isPolygon){
         if( o instanceof MultiLine){
            newgroup = new MultiLine((basic.MultiLine) o);
         }
         else{
            throw new KatamariException(node.getText() + " is not a MultiLine");
         }
      }
      else{
            throw new KatamariException(node.getText() + " is not a Polygon or a MultiLine");
      }
   }
   else {
      Line[] newlines = new Line[points-1];
      Point start = null, end = readPoint(node);
      for( int l=0; l<(points-1); l++ ) {
         Point p1 = readPoint(node);
         Point p2 =  readPoint(node.getNextSibling().getNextSibling());
         if( node.getNextSibling().getType() == AR ) {
            newlines[l] = new Line( p1, p2 );
         } else {
            newlines[l] = new Line( p2, p1 );
```

```
            }
            node = node.getNextSibling().getNextSibling();
            start = p2;
          }
          if( isPolygon ) { newgroup = new Polygon(newlines, start, end); }
          else { newgroup = new MultiLine(newlines); }
        }
        return newgroup;
    }

    /**
     * Expects a node equivalent to a label which is going into the Group
     * and returns the Group represented by this node and its siblings.
     */
    static Group readGroup( AST node ) throws antlr.RecognitionException, KatamariException{
        Group newgroup = new Group();
        String aname;
        while(node.getNextSibling()!=null){
            node = node.getNextSibling();
            aname=node.getText();
            Object a = symbol_table.get(aname);
            if( a==null || (!(a instanceof LineGroup) && !(a instanceof Group))) {
                throw new KatamariException( "can't add this type of object to group ("+
a.getClass().getName() );
            }
            newgroup.addObject( aname );
        }
        return newgroup;
    }

    /**
     * Expects a node equivalent to the label of a Group and adds items
     * represented by the siblings to the this Group.
     */
    static void add(AST root) throws KatamariException {
        AST current=root.getFirstChild();
        String gname=current.getText();
        current=current.getNextSibling();
        String aname=current.getText();
        Object a = symbol_table.get(aname), g = symbol_table.get(gname);
        if(g != null){
            if(a == null) {
                throw new KatamariException( "unkown symbol " +aname+ " can't be added to group." );
            } else {
                if( !(a instanceof LineGroup) && !(a instanceof Group)) {
                    throw new KatamariException( "A can't add this type of object to group ("+
a.getClass().getName() );
                }
            }

            if( (g instanceof Group) ){
                ((Group)g).addObject( aname );
            }
            else {
                throw new KatamariException( "B can't add to this type of object ("+ g.getClass().getName() );
            }
```

```
    } else{
        throw new KatamariException( "no group "+ gname +" found to add object to." );
    }
    println(" == "+ g);
}

/**
 * Expects a node equivalent to the label of the iteration variable and
 * repreatedly calls the statements inside the loop as long as this
 * variable is within the bounds of the iteration.
 */
static void iter(AST root) throws antlr.RecognitionException, KatamariException {
    AST current=root.getFirstChild();
    double lower= walker.mstmt(current);
    current= current.getNextSibling();
    double upper= walker.mstmt(current);
    current= current.getNextSibling();
    double increment= walker.mstmt(current);
    current= current.getNextSibling();
    String itername=current.getText();
    AST insideIteration = current.getNextSibling();
    AST firstInside = insideIteration;
    current = insideIteration;
    if(increment >0) {
        for(double i=lower; i<=upper ; i+=increment){
            insideIteration = firstInside;
            symbol_table.put( itername, new Double(i) );
            while(insideIteration != null){
                statements(insideIteration);
                insideIteration = insideIteration.getNextSibling();
            }
        }
    } else if(increment <0) {
        for(double i=lower; i>=upper ; i+=increment){
            symbol_table.put( itername, new Double(i) );
            while(insideIteration != null){
                statements(insideIteration);
                insideIteration = insideIteration.getNextSibling();
            }
        }
    } else {
        throw new KatamariException("increment value can't be zero");
    }
}

/**
 * Expects a node equivalent to the label of a Group, creates the kink
 * represented by the siblings and calls it on the Group.
 */
static void apply(AST root) throws antlr.RecognitionException, KatamariException {
    String groupname = root.getFirstChild().getText();
    AST sib = root.getFirstChild().getNextSibling();
    int location = sib.getType();
    sib = sib.getNextSibling();
    int orientation = sib.getType();
    sib = sib.getNextSibling();
```

```
      int kinkType = sib.getType();
      sib = sib.getNextSibling();
      int height = (int) walker.mstmt(sib);
      sib = sib.getNextSibling();
      int width = (int) walker.mstmt(sib);
      kinky.Kink k = null;
      if( kinkType == LITERAL_trikink ) {
         k = new kinky.TriKink(height, width);
      } else if( kinkType == LITERAL_squarekink ) {
         k = new kinky.SquareKink(height, width);
      } else if( kinkType == LITERAL_housekink ) {
         k = new kinky.HouseKink(height, width);
      }
      sib = sib.getNextSibling();
      int kinkCount = (int) walker.mstmt(sib);

      Object applyto = symbol_table.get(groupname);
      if( applyto instanceof Group ) {
         groupApply(applyto, location, orientation, height, width, k, kinkCount);
      } else if( applyto instanceof LineGroup ) {
         kinky.ApplyKink.ApplyKink( ((LineGroup)applyto), location, orientation, k, kinkCount );
      } else {
         throw new KatamariException( "can't apply kink to object "+ applyto.getClass().getName() );
      }
   }

   /**
    * Applies a kink to a LineGroup (using ApplyKink) or Group (using
    * ApplyKink on the individual elements).
    */
   static void groupApply(Object applyto, int location, int orientation, int height, int width, kinky.Kink k,
int kinkCount) throws antlr.RecognitionException, KatamariException{
       String[] symbols = ((Group)applyto).getGroupNames();
         for( int g=0; g<symbols.length; g++ ) {
            Object nextgroup = symbol_table.get(symbols[g]);
            if( nextgroup instanceof LineGroup ) {
               kinky.ApplyKink.ApplyKink( ((LineGroup)nextgroup), location, orientation, k, kinkCount );
            }
            else if(nextgroup instanceof Group){
               System.out.println("there's another group in here!");
               if(true){
                  groupApply(nextgroup, location, orientation, height, width, k, kinkCount);
               }
            }
      }
   }

   /**
    * Expects a node equivalent to the boolean (or mstmt) of an if and calls
    * the statements inside the sibling nodes if this boolean evaluates to
    * true. Otherwise attempts "walkIntoElse".
    */
   static void iF(AST root) throws antlr.RecognitionException, KatamariException {
      AST current=root.getFirstChild();
      //       println("# Iffing on: " + current.getText());
      boolean b;
```

```
    if((b=makeBool(walker.mstmt(current)))){
        System.out.println("value in this if: "+ b + "iffing on: "+current.getText());
        current=current.getNextSibling();
        while(current!=null){
            statements(current);
            current=current.getNextSibling();
        }
        return;
    }
    else{
        walkIntoElse(current);
    }
}

/**
 * turn them nasty numbers into friendly booleans
 */
static boolean makeBool(int i){
    if(i==0) { return false; }
    else     { return true;  }
}

static boolean makeBool(double d){
    if(d==0.0) { return false; }
    else       { return true;  }
}

/**
 * Checks to see if this (the node after an if) is an else, and if it is
 * then we do the statements contained within it.
 */
static void walkIntoElse(AST root) throws antlr.RecognitionException, KatamariException {
    AST current=root;
    while(current!=null && current.getText().compareTo("else")!=0){
        current=current.getNextSibling();
    }
    if(current==null){
        return;
    }
    current=current.getNextSibling();
    while(current!=null){
        statements(current);
        current=current.getNextSibling();
    }
    return;
}

/**
 * Main method: creates theGroup, kinks off walking the tree and draws
 * the resulting fractal at the end.
 */
static void doEverything(AST root) throws antlr.RecognitionException {
    try {
        Group theGroup = new Group();
        symbol_table.put( theGroup_symbol, theGroup );
        if(root.getText().compareTo("#")!=0){
```

```
            throw new KatamariException("file must start with a # <output file> - "+ root.getText());
        }
        root=root.getFirstChild();
        String filename=root.getText();
        try{
            File f = new File(filename);
            if( f.exists() ) {
                throw new KatamariException("export file already exists.");
            }
        } catch(Exception x){ }

        // walk the tree
        root=root.getNextSibling();
        AST current=root;
        while(root!=null){
            statements(current);
            root=root.getNextSibling();
            current=root;
        }

        // print the group!
        LineGroup biggroup = new MultiLine();
        String[] symbols =theGroup.getGroupNames();
        for( int g=0; g<symbols.length; g++ ) {
            biggroup.addObject( symbol_table.get(symbols[g]) );
        }

        draw.DrawFractal.DrawFractalBitmap( biggroup, filename );
    } catch( KatamariException ke ) {
        System.err.println( "KatamariException: "+ ke.getMessage() );
        ke.printStackTrace();
    }
}

/**
 * Reads a node and calls the appropriate method from above based on the
 * node type.
 */
static void statements(AST root) throws antlr.RecognitionException, KatamariException {
    switch(root.getType()){
        case ASS:
            declare(root);
            break;

        case INC:
            inc(root);
            break;

        case DEC:
            dec(root);
            break;

        case LITERAL_add:
            add(root);
            break;
```

```
        case LITERAL_if:
            iF(root);
            break;

        case LITERAL_iter:
            iter(root);
            break;

        case LITERAL_apply:
            apply(root);
            break;

        case LITERAL_print:
            print(root);
            break;

        case LITERAL_else:
            return;
        default:
            throw new KatamariException("incompatible types: "+ root.getText() + "  " +root.getType());
    }
}

static void print(AST root){
    if(root.getFirstChild()!=null){
        System.out.println(root.getFirstChild().getText());
    }
}

static void inc(AST root) throws antlr.RecognitionException {
    System.out.println("Im over here.");
    walker.mstmt(root);
}

static void dec(AST root) throws antlr.RecognitionException {
    walker.mstmt(root);
}

static double doMinus(AST root) throws antlr.RecognitionException{
    try{
        int nch = root.getNumberOfChildren();

        AST current = root.getFirstChild();
        if(nch == 1){
            return (-1)*( (double) walker.mstmt(current));
        }
        else if(nch==2){
            return ( ((double)walker.mstmt(current))
            - ((double)walker.mstmt(current.getNextSibling())));
            //as close to functional programming as possible
        }else{
            throw new KatamariException("Why do you have more than two arguments for '-'??");
        }
    }catch( KatamariException ke ) {
        System.err.println( "KatamariException: "+ ke.getMessage() );
        ke.printStackTrace();
```

```
        }
        return 0;
    }

    private static void println( String toprint ) {
        if( DEBUG ) { System.out.println( toprint ); }
    }
}

/**
 * Subclass for storing a Group - ie. a Vector of symbols for items in which
 * should be located in the symbol table.
 */
class Group {
    private Vector symbols = new Vector();
    public void addObject( String symbol ) {
        symbols.add(symbol);
    }
    public String[] getGroupNames() {
        String[] names = new String[symbols.size()];
        symbols.copyInto(names);
        return names;
    }
    public String toString() {
        String out = "{ "+ (symbols.isEmpty()?"":symbols.elementAt(0));
        for( int l=1; l<symbols.size(); l++ ) {
            out += ", " + symbols.elementAt(l);
        }
        return out +" }";
    }
}

/**
 * Subclass - A generic exception for Katamari to throw.
 */
class KatamariException extends Exception {
    public KatamariException( String msg ) {
        super( msg );
        //System.out.println("error, quitting in katamari");
    }
}
```

Basic Package:

```
/*
 * Line.java
 *
 * Stores a start and end Point that specify this Line. Note, this is a sub-
 * class of LineGroup and should not be used in the "Line" assignment of the
 * tree walker.
 */

package basic;

public class Line extends Object{

   private Point start;
   private Point finish;

   /** Creates a new instance of Line */
   public Line( Point start, Point finish ) {
      this.start = start;
      this.finish = finish;
   }

   public Point getStart() { return start; }
   public Point getFinish() { return finish; }
   public void setStart( Point start ) {
      this.start = start;
   }
   public void setFinish( Point finish ) {
      this.finish = finish;
   }

   public double getLength() {
      double x = start.getX() - finish.getX();
      x = x*x;
      double y = start.getY() - finish.getY();
      y = y*y;
      return( Math.sqrt(x+y) );
   }

   public double getSlope() {
      double s;
      if( start.getX() == finish.getX() ) {
         s = Double.POSITIVE_INFINITY;
      } else {
         s = (start.getY() - finish.getY())
         / (start.getX() - finish.getX());
      }
      return s;
   }

   public String toString() {
      return "("+ start.getX() +", "+ start.getY() +") -> "
         + "("+ finish.getX() +", "+ finish.getY() +")";
   }
```

```
    public Line getAllLines(){
        return this;
    }

    public boolean equals( Object o ) {
        if( o instanceof Line ) {
            return ((Line)o).getStart().equals(start) && ((Line)o).getFinish().equals(finish);
        } else {
            return false;
        }
    }

}
```

```
/*
 * LineGroup.java
 *
 * Stores two Vectors of Lines, allowing user to add and remove from the group,
 * replace a Line with one or more Lines, and retrieve and object array of all
 * the activeLines contained within the group.  The lines are classified as
 * ACTIVE (should be kinked if ApplyKink is called on this group) and COMPLETE
 * (should not be kinked).
 */

package basic;

import java.util.Vector;

public abstract class LineGroup extends Object{

   public static final int COMPLETE = 0;
   public static final int ACTIVE = 1;

   protected Vector activeLines;
   protected Vector completeLines;

   /** Creates a new instance of Group */
   public LineGroup() {
      this.activeLines = new Vector();
      this.completeLines = new Vector();
   }

   public LineGroup(LineGroup lg){
      activeLines= (java.util.Vector)lg.activeLines.clone();
      completeLines= (java.util.Vector)lg.completeLines.clone();
   }

   public void addLine( Line newline, int linetype ) {
      if( linetype == ACTIVE )
         activeLines.add(newline);
      else if( linetype == COMPLETE )
         completeLines.add(newline);
   }

   public void addLines( Line[] newlines, int linetype ) {
      for( int l=0; l<newlines.length; l++ ) {
         addLine( newlines[l], linetype );
      }
   }

   public void addLineGroup( LineGroup group ) {
      Line[] a = group.getLines( ACTIVE );
      for( int l=0; l<a.length; l++ ) {
         addLine( a[l], ACTIVE );
      }
      Line[] c = group.getLines( COMPLETE );
      for( int l=0; l<c.length; l++ ) {
         addLine( c[l], COMPLETE );
      }
   }
```

```
public void addObject( Object object ) {
   if( object instanceof LineGroup ) { addLineGroup((LineGroup)object); }
   else if( object instanceof Line ) { addLine((Line)object, ACTIVE); }
}

public void addLineAt( Line newline, int index, int linetype ) {
   if( linetype == ACTIVE )
      activeLines.insertElementAt( newline, index );
   else if( linetype == COMPLETE )
      completeLines.insertElementAt( newline, index );
}

public void removeLine( Line oldline, int linetype ) {
   if( linetype == ACTIVE )
      activeLines.remove(oldline);
   else if( linetype == COMPLETE )
      completeLines.remove(oldline);
}

public void removeLineAt( int index, int linetype ) {
   if( linetype == ACTIVE )
      activeLines.removeElementAt( index );
   else if( linetype == COMPLETE )
      completeLines.removeElementAt( index );
}

public void replace( Line oldline, Line[] newlines, int linetype ) {
   int index = activeLines.indexOf(oldline);
   if( index > 0 ) {
      activeLines.removeElementAt( index );
   } else {
      index = activeLines.size();
   }
   for( int l=0; l<newlines.length; l++ ) {
      activeLines.insertElementAt( newlines[l], index++ );
   }
}

public Line[] getLines( int linetype ) {
   Line[] out = null;
   if( linetype == ACTIVE ) {
      out = new Line[activeLines.size()];
      activeLines.toArray( out );
   } else if( linetype == COMPLETE ) {
      out = new Line[completeLines.size()];
      completeLines.toArray( out );
   }
   return out;
}

public Line[] getAllLines() {
   Vector v = new Vector( activeLines );
   v.addAll( completeLines );
   Line[] out = new Line[v.size()];
   v.toArray( out );
```

```
        return out;
    }

    public String toString() {
        String group = "Group: Active="+ activeLines.size()+" Complete="+ completeLines.size()+" ";
        Line[] mylines = getAllLines();
        for( int l=0; l<mylines.length; l++ ) {
            group += mylines[l].toString() + "\n";
        }
        return group;
    }

}
```

```
/*
 * MultiLine.java
 *
 * For the convenience, this class defines a generic collection of Lines
 */

package basic;

public class MultiLine extends LineGroup {

   public MultiLine() {
      super();
   }

   public MultiLine(MultiLine ml){
      super();
      activeLines=(java.util.Vector)ml.activeLines.clone();
      completeLines=(java.util.Vector)ml.completeLines.clone();
   }

   public MultiLine(Line[] lines) {
      super();
      for( int l=0; l<lines.length; l++ ) {
         addLine( lines[l], ACTIVE );
      }
   }

}
```

```
/*
 * Point.java
 *
 * Stores two double values for the X & Y coordinates of a point.
 */

package basic;

public class Point extends Object{

   private double x;
   private double y;

   /** Creates a new instance of Point */
   public Point( double x, double y ) {
      this.x = x;
      this .y = y;
   }

   public Point(Point P){
      x=P.getX();
      y=P.getY();
   }

   public double getX() { return x; }
   public double getY() { return y; }
   public void setX(double x) { this.x = x; }
   public void setY(double y) { this.y = y; }

   public String toString() {
      return "("+ x +", "+ y +")";
   }

   public boolean equals( Object o ) {
      if( o instanceof Point ) {
         return ((Point)o).getX() == x && ((Point)o).getY() == y;
      } else {
         return false;
      }
   }

}
```

```
/*
 * Polygon.java
 *
 * A special type of group which stores the lines given and creates a final
 * line from the start and end of the polygon.
 */

package basic;

public class Polygon extends LineGroup {

   public Polygon() {
      super();
   }

   public Polygon(Polygon p){
      super();
      activeLines=(java.util.Vector)p.activeLines.clone();
      completeLines=(java.util.Vector)p.completeLines.clone();
   }

   public Polygon(Line[] lines, Point start, Point end) {
      super();
      for( int l=0; l<lines.length; l++ ) {
         addLine( lines[l], ACTIVE );
      }
      Line l = new Line( start, end );
      addLine( l, ACTIVE );
   }
}
```

```
/*
 * SingleLine.java
 *
 * This class is a single Line but stored in a LineGroup so that it can be
 * smoothly used with ApplyKink.
 */
package basic;

public class SingleLine extends LineGroup {

   public SingleLine() {
      super();
   }

   public SingleLine(SingleLine sl){
      super();
      activeLines=(java.util.Vector)sl.activeLines.clone();
      completeLines=(java.util.Vector)sl.completeLines.clone();
   }

   public SingleLine( Point start, Point finish ) {
      super();
      addLine( new Line(start, finish), ACTIVE );
   }

   public Line getLine() {
      return (Line)activeLines.elementAt(0);
   }

}
```

Draw Package:

```
//---------------------------------------------------------------------
//  Author:     Jean-Pierre Dubé
//  Source:     http://www.javaworld.com/javaworld/javatips/jw-javatip60.html
//---------------------------------------------------------------------


package draw;

import java.awt.*;
import java.io.*;
import java.awt.image.*;

public class BMPFile extends Component {
  //--- Private constants
  private final static int BITMAPFILEHEADER_SIZE = 14;
  private final static int BITMAPINFOHEADER_SIZE = 40;
  //--- Private variable declaration
  //--- Bitmap file header
  private byte bitmapFileHeader [] = new byte [14];
  private byte bfType [] = {'B', 'M'};
  private int bfSize = 0;
  private int bfReserved1 = 0;
  private int bfReserved2 = 0;
  private int bfOffBits = BITMAPFILEHEADER_SIZE + BITMAPINFOHEADER_SIZE;
  //--- Bitmap info header
  private byte bitmapInfoHeader [] = new byte [40];
  private int biSize = BITMAPINFOHEADER_SIZE;
  private int biWidth = 0;
  private int biHeight = 0;
  private int biPlanes = 1;
  private int biBitCount = 24;
  private int biCompression = 0;
  private int biSizeImage = 0x030000;
  private int biXPelsPerMeter = 0x0;
  private int biYPelsPerMeter = 0x0;
  private int biClrUsed = 0;
  private int biClrImportant = 0;
  //--- Bitmap raw data
  private int bitmap [];
  //--- File section
  private FileOutputStream fo;
  //--- Default constructor
  public BMPFile() {
  }
  public void saveBitmap (String parFilename, Image parImage, int
parWidth, int parHeight) {
    try {
      fo = new FileOutputStream (parFilename);
      save (parImage, parWidth, parHeight);
      fo.close ();
    }
    catch (Exception saveEx) {
      saveEx.printStackTrace ();
    }
```

```
  }
  /*
   *  The saveMethod is the main method of the process. This method
   *  will call the convertImage method to convert the memory image to
   *  a byte array; method writeBitmapFileHeader creates and writes
   *  the bitmap file header; writeBitmapInfoHeader creates the
   *  information header; and writeBitmap writes the image.
   *
   */
  private void save (Image parImage, int parWidth, int parHeight) {
    try {
      convertImage (parImage, parWidth, parHeight);
      writeBitmapFileHeader ();
      writeBitmapInfoHeader ();
      writeBitmap ();
    }
    catch (Exception saveEx) {
      saveEx.printStackTrace ();
    }
  }
  /*
   * convertImage converts the memory image to the bitmap format (BRG).
   * It also computes some information for the bitmap info header.
   *
   */
  private boolean convertImage (Image parImage, int parWidth, int parHeight) {
    int pad;
    bitmap = new int [parWidth * parHeight];
    PixelGrabber pg = new PixelGrabber (parImage, 0, 0, parWidth, parHeight,
                                        bitmap, 0, parWidth);
    try {
      pg.grabPixels ();
    }
    catch (InterruptedException e) {
      e.printStackTrace ();
      return (false);
    }
    pad = (4 - ((parWidth * 3) % 4)) * parHeight;
    biSizeImage = ((parWidth * parHeight) * 3) + pad;
    bfSize = biSizeImage + BITMAPFILEHEADER_SIZE +
BITMAPINFOHEADER_SIZE;
    biWidth = parWidth;
    biHeight = parHeight;
    return (true);
  }
  /*
   * writeBitmap converts the image returned from the pixel grabber to
   * the format required. Remember: scan lines are inverted in
   * a bitmap file!
   *
   * Each scan line must be padded to an even 4-byte boundary.
   */
  private void writeBitmap () {
      int size;
      int value;
      int j;
```

```
    int i;
    int rowCount;
    int rowIndex;
    int lastRowIndex;
    int pad;
    int padCount;
    byte rgb [] = new byte [3];
    size = (biWidth * biHeight) - 1;
    pad = 4 - ((biWidth * 3) % 4);
    if (pad == 4)   // <==== Bug correction
      pad = 0;     // <==== Bug correction
    rowCount = 1;
    padCount = 0;
    rowIndex = size - biWidth;
    lastRowIndex = rowIndex;
    try {
      for (j = 0; j < size; j++) {
        value = bitmap [rowIndex];
        rgb [0] = (byte) (value & 0xFF);
        rgb [1] = (byte) ((value >> 8) & 0xFF);
        rgb [2] = (byte) ((value >>  16) & 0xFF);
        fo.write (rgb);
        if (rowCount == biWidth) {
          padCount += pad;
          for (i = 1; i <= pad; i++) {
            fo.write (0x00);
          }
          rowCount = 1;
          rowIndex = lastRowIndex - biWidth;
          lastRowIndex = rowIndex;
        }
        else
          rowCount++;
        rowIndex++;
      }
      //--- Update the size of the file
      bfSize += padCount - pad;
      biSizeImage += padCount - pad;
    }
    catch (Exception wb) {
      wb.printStackTrace ();
    }
  }
/*
 * writeBitmapFileHeader writes the bitmap file header to the file.
 *
 */
private void writeBitmapFileHeader () {
  try {
    fo.write (bfType);
    fo.write (intToDWord (bfSize));
    fo.write (intToWord (bfReserved1));
    fo.write (intToWord (bfReserved2));
    fo.write (intToDWord (bfOffBits));
  }
  catch (Exception wbfh) {
```

```
      wbfh.printStackTrace ();
    }
  }
  /*
   *
   * writeBitmapInfoHeader writes the bitmap information header
   * to the file.
   *
   */
  private void writeBitmapInfoHeader () {
    try {
      fo.write (intToDWord (biSize));
      fo.write (intToDWord (biWidth));
      fo.write (intToDWord (biHeight));
      fo.write (intToWord (biPlanes));
      fo.write (intToWord (biBitCount));
      fo.write (intToDWord (biCompression));
      fo.write (intToDWord (biSizeImage));
      fo.write (intToDWord (biXPelsPerMeter));
      fo.write (intToDWord (biYPelsPerMeter));
      fo.write (intToDWord (biClrUsed));
      fo.write (intToDWord (biClrImportant));
    }
    catch (Exception wbih) {
      wbih.printStackTrace ();
    }
  }
  /*
   *
   * intToWord converts an int to a word, where the return
   * value is stored in a 2-byte array.
   *
   */
  private byte [] intToWord (int parValue) {
    byte retValue [] = new byte [2];
    retValue [0] = (byte) (parValue & 0x00FF);
    retValue [1] = (byte) ((parValue >>  8) & 0x00FF);
    return (retValue);
  }
  /*
   *
   * intToDWord converts an int to a double word, where the return
   * value is stored in a 4-byte array.
   *
   */
  private byte [] intToDWord (int parValue) {
    byte retValue [] = new byte [4];
    retValue [0] = (byte) (parValue & 0x00FF);
    retValue [1] = (byte) ((parValue >>  8) & 0x000000FF);
    retValue [2] = (byte) ((parValue >>  16) & 0x000000FF);
    retValue [3] = (byte) ((parValue >>  24) & 0x000000FF);
    return (retValue);
  }
}
```

```
/*
 * DrawFractal.java
 *
 * This class takes a LineGroup and draws it to a Swing window, optionally
 * saving it to bitmap file as well.
 */

package draw;

import basic.*;
import javax.swing.*;
import java.awt.*;

public class DrawFractal {

   public static final int w = 1000;
   public static final int h = 1000;

   /**
    * Opens a new Swing window and displays the LineGroup in it.
    */
   public static Image DrawFractalSwing( LineGroup group ) {
      JFrame window = new JFrame("You've been Ipso Fracto-ed!");
      window.setSize( w,h );
      window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      Canvas display = new Canvas();
      display.setBackground(Color.white);
      window.getContentPane().add(display);
      window.setVisible(true);

      Image img = display.createImage(w,h);
      Graphics2D g = (Graphics2D) img.getGraphics();

      Line[] lines = group.getAllLines();
      for (int i=0; i<lines.length; i++){
         Line line = lines[i];
         basic.Point start = line.getStart();
         basic.Point finish = line.getFinish();
         g.drawLine((int)start.getX(), (int)start.getY(), (int)finish.getX(), (int)finish.getY());
      }

      Graphics2D g2 = (Graphics2D) display.getGraphics();
      g2.drawImage(img, 0, 0, display);
      window.repaint();

      return img;
   }

   /**
    * Calls DrawFractalSwing to generate fractal and saves the result in a
    * bitmap file.
    */
   public static void DrawFractalBitmap( LineGroup group, String filename ) {
      Image img = DrawFractalSwing( group );
      BMPFile bmp = new BMPFile();
```

```
        bmp.saveBitmap(filename, img, w, h);
    }
}
```

Kink Package:

```
/*
 * ApplyKink.java
 *
 * Provides the static method that applies a kink to all lines in a group.
 * Refers to the Kink class and relies on specific kink instances to implement
 * the methods for building that type of kink.
 */

package kinky;

import basic.*;
import kinky.Kink;

public class ApplyKink {

   public static int LOC_CENTER = antlra.IFLexerTokenTypes.LITERAL_CENTER;
   public static int LOC_LEFT = antlra.IFLexerTokenTypes.LITERAL_LEFT;
   public static int LOC_RIGHT = antlra.IFLexerTokenTypes.LITERAL_RIGHT;
   public static int ORIENT_POS = antlra.IFLexerTokenTypes.LITERAL_POS;
   public static int ORIENT_NEG = antlra.IFLexerTokenTypes.LITERAL_POS;

   /**
    * For every line in the LineGroup, removes it from the group, applies Kink
    * getting the new lines represented which are added back to the group.
    */
   public static LineGroup ApplyKink( LineGroup thegroup, int location,
      int orientation, Kink thekink, int kinkCount ) {

      Line[] active = thegroup.getLines( LineGroup.ACTIVE );
      for( int a=0; a<active.length; a++ ) {
         Line nextline = active[a];
         if( nextline.getLength() >= 1.0 ) {
            // remove the original unkinked line
            thegroup.removeLine( nextline, LineGroup.ACTIVE );
            // get the end corners where the kink starts and ends
            thekink.setKinkPoints( nextline, location );
            Line[] done = thekink.getEndLines( nextline, location );
            thegroup.addLines( done, LineGroup.COMPLETE );
            // get the lines that make up the kinks
            Line[][] kinked = thekink.applykink( nextline, location, kinkCount, orientation );
            thegroup.addLines( kinked[1], LineGroup.COMPLETE );
            thegroup.addLines( kinked[0], LineGroup.ACTIVE );
         } else {
            return thegroup;
         }
      }
      return thegroup;
   }
}
```

```
/*
 * HouseKink.java
 *
 * Extends Kink to implement a house (or pentagon) shaped kink.
 */

package kinky;

import basic.*;

public class HouseKink extends Kink {

   public HouseKink( int height, int width ) {
      super( height, width );
      LINES_PER_KINK = 4;
   }

   public HouseKink( double height, double width ) {
      super( height, width );
      LINES_PER_KINK = 4;
   }

   /**
    * Implements the method kink() from parent class. Calculates a midpoint
    * and two sidepoints and about 2/3 the height of the kink and returns the
    * four lines representing the house shaped part of the kink.
    */
   protected Line[] kink(Point start, Point end, double kinkheight) {
      double sideheight = (2.0/3.0) * kinkheight;
      double[] sidedist = getDistances( new Line(start,end), sideheight, true );
      Point left = new Point( start.getX()+sidedist[0], start.getY()+sidedist[1] );
      Point right = new Point( end.getX()+sidedist[0], end.getY()+sidedist[1] );

      Point midpoint = getMidpoint( start, end );
      double[] centerdist = getDistances( new Line(start,end), kinkheight, true );
      midpoint.setX( midpoint.getX() + centerdist[0] );
      midpoint.setY( midpoint.getY() + centerdist[1] );

      Line[] out = {  new Line( start, left ),
                      new Line( left, midpoint ),
                      new Line( midpoint, right ),
                      new Line( right, end ) };
      return out;
   }

}
```

```
/*
 * Kink.java
 *
 * Abstract definition of a Kink. This provides methods
 */

package kinky;

import basic.*;

public abstract class Kink {

    protected int LINES_PER_KINK = 0;
    protected double height;
    protected double width;
    protected Point startKink;
    protected Point endKink;

    /** Creates a new instance of Kink with its relative hieght and width. */
    public Kink( int height, int width ) {
        this.height = height;
        this.width = width;
    }

    public Kink( double height, double width ) {
        this.height = height;
        this.width = width;
    }

    /**
     * Applies given number of this kink to a line at the given location and
     * orientation. This is done by:  1) Working out the parts of the line not
     * altered by the kink and moving these to the "done" parts of the
     * LineGroup.  2) Calling the kink method (implemented by children) to get
     * the lines which are the altered part of the line.  3) Creates small gap
     * lines between kinks and adds these to "done" lines.
     */
    public Line[][] applykink( Line line, int location, int kinkCount, int orientation ) {
        Line[] newcomplete = new Line[(kinkCount-1)];
        Line[] newactive = new Line[kinkCount * LINES_PER_KINK];
        int aindex = 0; int cindex = 0;
        Line tokink = new Line(startKink, endKink);
        double kinkheight = line.getLength() / height;
        double width = tokink.getLength() / ((2*kinkCount)-1);
        double[] distances = getDistances( tokink, width );
        Point start = startKink;
        Point end = endKink;
        for( int k=0; k<kinkCount; k++ ) {
            end = new Point( start.getX()+distances[0], start.getY()+distances[1] );
            Line[] kink;
            if( orientation == ApplyKink.ORIENT_POS ) {
                kink = kink( start, end, kinkheight );
            } else {
                kink = kink( end, start, kinkheight );
            }
            for( int k2=0; k2<LINES_PER_KINK; k2++ ) {
```

```
            newactive[aindex++] = kink[k2];
        }
        start = end;
        if( k < kinkCount-1 ) {
            end = new Point( start.getX()+distances[0], start.getY()+distances[1] );
            newcomplete[cindex++] = new Line( start, end );
            start = end;
        }
    }
    Line[][] l = { newactive, newcomplete };
    return l;
}

/**
 * Should be implemented by choldren to return the lines involed in the
 * kink - based on orientation, height and type.
 */
protected abstract Line[] kink( Point start, Point end, double kinkheight );

/**
 * Gets the two lines on either side of the kink based on the original
 * straight line.
 */
public Line[] getEndLines( Line line, int location ) {
    Line[] doneLines;
    if( location == ApplyKink.LOC_LEFT ) {
        Line[] d = { new Line(endKink, line.getFinish()) };
        doneLines = d;
    } else if( location == ApplyKink.LOC_RIGHT ) {
        Line[] d = { new Line(line.getStart(), startKink) };
        doneLines = d;
    } else {
        Line[] d = { new Line(line.getStart(), startKink),
            new Line(endKink, line.getFinish()) };
        doneLines = d;
    }
    return doneLines;
}

/**
 * Gets the start and end points along the original line where the kinked
 * portion will start and end.  Done by dividing line length with relative
 * line distance then adding the x & y distances to the start point.
 */
public void setKinkPoints( Line line, int location ) {
    double kinklength = line.getLength() / width;
    double donelength = (line.getLength() - kinklength) / 2;
    double[] distances = getDistances( line, donelength );

    Point b1 = new Point( line.getStart().getX()+distances[0], line.getStart().getY()+distances[1] );
    Point b2 = new Point( line.getFinish().getX()-distances[0], line.getFinish().getY()-distances[1] );
    if( location == ApplyKink.LOC_LEFT ) {
        startKink = line.getStart();
        endKink = b1;
    } else if( location == ApplyKink.LOC_RIGHT ) {
        startKink = b2;
```

```
            endKink = line.getFinish();
        } else {
            startKink = b1;
            endKink = b2;
        }
    }

    /**
     * Gets the X & Y disantances of each line segment based on length and
     * slope of the line.  Has optional boolean to indicate we want the normal
     * of these distances.
     */
    protected double[] getDistances( Line line, double length, boolean normal ) {
        double slope = line.getSlope();
        double xdist, ydist;
        if( slope == Double.POSITIVE_INFINITY || slope == Double.NEGATIVE_INFINITY ) {
            xdist = 0; ydist = length;
            if( line.getStart().getY() > line.getFinish().getY() ) {
                ydist = -length;
            }
        } else if( slope == 0 ) {
            xdist = length; ydist = 0;
            if( line.getStart().getX() > line.getFinish().getX() ) {
                xdist = -xdist;
            }
        } else {
            ydist = Math.sqrt( ((slope*slope)*(length*length)) / (1+(slope*slope)) );
            if( line.getStart().getY() > line.getFinish().getY() ) {
                ydist = -ydist;
            }
            xdist = ydist / slope;
        }
        if( normal ) {
            double tmp = xdist;
            xdist = -ydist;
            ydist = tmp;
        }
        double[] dist = { xdist, ydist };
        return dist;
    }
    protected double[] getDistances( Line line, double length ) {
        return getDistances(line, length, false);
    }

    /**
     * Gets the midpoint between two other points.
     */
    protected Point getMidpoint( Point start, Point end ) {
        double x = start.getX() - ((start.getX() - end.getX())/2);
        double y = start.getY() - ((start.getY() - end.getY())/2);
        return new Point(x,y);
    }

}
```

```
/*
 * KinkException.java
 *
 * Thrown if there is an error kinking.
 */

package kinky;

public class KinkException extends Exception {

    /** Creates a new instance of KinkException */
    public KinkException( String msg ) {
        super( msg );
        System.out.println("error, quitting");
    }

}
```

```
/*
 * SquareKink.java
 *
 * Extends Kink to implement a square shaped kink.
 */

package kinky;

import basic.*;


public class SquareKink extends Kink {

   public SquareKink( int height, int width ) {
      super( height, width );
      LINES_PER_KINK = 3;
   }

    public SquareKink( double height, double width ) {
      super( height, width );
      LINES_PER_KINK = 3;
   }

   /**
    * Implements the method kink() from parent class. Calculates points
    * normal from the original line at the height of the kink and returns the
    * three lines representing the square shaped part of the kink.
    */
   protected Line[] kink( Point start, Point end, double kinkheight ) {
      double[] dist = getDistances( new Line(start,end), kinkheight, true );
      Point left = new Point( start.getX()+dist[0], start.getY()+dist[1] );
      Point right = new Point( end.getX()+dist[0], end.getY()+dist[1] );
      Line[] out = {  new Line( start, left ),
                      new Line( left, right ),
                      new Line( right, end ) };
      return out;
   }

}
```

```
/*
 * TriKink.java
 *
 * Extends Kink to implement a triangular shaped kink.
 */

package kinky;

import basic.*;

public class TriKink extends Kink {

   public TriKink( int height, int width ) {
      super( height, width );
      LINES_PER_KINK = 2;
   }

   public TriKink( double height, double width ) {
      super( height, width );
      LINES_PER_KINK = 2;
   }

   /**
    * Implements the method kink() from parent class. Calculates midpoint
    * normal from the original line at the height of the kink and returns the
    * two lines representing the triangle shaped part of the kink.
    */
   public Line[] kink( Point start, Point end, double kinkheight ) {
      Point midpoint = getMidpoint( start, end );
      double[] dist = getDistances( new Line(start,end), kinkheight, true );
      midpoint.setX( midpoint.getX() + dist[0] );
      midpoint.setY( midpoint.getY() + dist[1] );
      Line[] out = {  new Line( start, midpoint ),
                      new Line( midpoint, end ) };
      return out;
   }
}
```