Ellie Programming Lang.

Drawing Shapes Made Easier

Çelik Köseoğlu, Berk Evren Abbasoğlu, Fatih Sabri Aktepe - 28 November 2016

Documentation

1. Overview

The Ellie Programming Lang. is an efficient and simple shape drawing solution. The language is designed to be highly capable for drawing simple shapes, as well as user defined complex shapes.

The main focus while defining this new programming language was the artist. The language has to be simple, but not simpler. This means, the language should support complex operations while still having an appealing syntax. Ellie manages to incorporate both of these features by being a Dynamically Typed Language. The interpreter does most of the work while letting the artist work freely and independently.

A. Why Use Ellie?

Ellie offers simple mechanics to draw highly advanced and custom shapes. While maintaining its easy to use properties, it falls no short of supporting complex operations. The users can define custom shapes in addition to those already provided. With simple parameters like the location and size of the shape, it is possible to immediately draw them. These parameters are what Ellie calls “optionals,” meaning the user should use these parameters only if they want to draw customised shapes. Ellie is easy to use, highly functional, and is sure to address any demands regarding drawing shapes.

B. Where Does the Name Come From?

The name Ellie, comes from one of the best video games recorded in recent history: The Last of Us. Ellie is the name of one of the protagonists. You can find her picture on our cover page.

2. Introduction

A. Ellie, The Mother of All Classes

In Ellie programming language, all Shapes are kind of an Ellie. When trying to create a new shape, the user instantiates an Ellie object, giving it a name, and then calls the constructor of the desired shape for this Ellie object. Following is an example of this.

```
Ellie line = Line();
Ellie oval = Oval();
Ellie rect = Rect();
```

While shapes like Line, Oval and Rectangle are defined by Ellie, users can create their own custom shape classes and inherit the features from the parent Ellie class. The relation between the parent Ellie class and the built-in shapes are shown below.

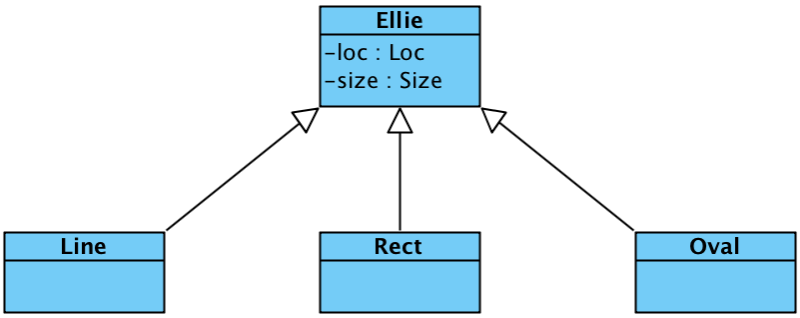

Figure 1: Ellie and the classes that inherit from it.

B. Comments

Ellie incorporates C style comments. This way, developers coming from other languages will grasp the comment syntax easily. Examples are shown below:

```
//this is a single line comment

/* this is a multi
   line comment */
```

C. Entry Point

Like most programming languages, Ellie provides an entry point for the application. When the application is invoked, the **init()** function instantiates the variables in the background. This instantiation process and the identifiers of the variables are all isolated from the user to provide a simple and clean interface. The shortest Ellie program is written as follows:

```
init() {
     //do nothing
}
```

Here, braces are used to capsulate statements into a “scope”, just like the Swift Programming Lang.

D. Concept of Functions

Ellie Programming Lang. supports modularity at the level of functions. The example below shows defining a function called giveReport(). This function gets called from the init() function.

```
init() {
     giveReport();
}

giveReport() {
     // do something
}
```

E. Primitive Types and Operations

Ellie language integrates the common primitive types such as Integer, Float, String and Boolean. These primitive types are objects, just like the Python Programming Lang. All types support the addition (+) operator. However, String and Boolean does not support the rest. Here are some examples of the primitive types and supported operations:

```
Int examGrade = 100;
Double pi = 3.14;
Bool passed = True;
String ellieSays = “Scorpions are pretty creepy.”;
```

In Strings, addition operator (+) is used for concatenation purposes. However, if you want to add an Int type to your String for example, then you have to cast it into a String before concatenation. Examples are as follows:

```
//String Concatenation
String success = “This report got full credit”;
String successTail = “from Buğra Gedik.”;
String bottomLine = success + successTail;
//reads “This report got full credit from Buğra Gedik.”

//Casting
String studentSays = “This report deserves “;
Int highestGrade = 100;
String bottomLine = studentSays + Stringify(highestGrade);
//reads “This report deserves 100”
```

The rest of the operations are subtraction (-), division (/), multiplication (*) and modulus (%). Examples for these operations are as follows:

```
//Addition, Division and Multiplication on numbers
Int midterm = 50;
Int final = midterm * 2;
Double average = (midterm + final) / 2;
//average will be 75

//Addition on booleans
Bool passed = False;
Bool playedVideoGames = True;
Bool bottomLine = passed + playedVideoGames;
//bottomLine is True, because at least one of them is True

//Modulus
Int remainder = 10 % 3; //remainder is 1
```

F. Bitwise Operations

Bitwise operations in the Ellie Programming Lang. are the similar to other programming languages. Examples are shown below:

```
Bool passed = True;
Bool bestReport = True;
Bool goodAtMath = False;

Bool successful = passed && bestReport; //True
Bool stillSuccessful = bestReport || goodAtMath; //True
```

G. Casting

In Ellie, there are built in methods for casting one type to another type. This is particularly useful when the developer wants to concatenate numbers with strings or wants to have higher precision on division operations. All casting methods and their usages are shown below.

```
String piString = Stringfy(3.14); // holds “3.14”
Double pi = Doublify(piString); // holds 3.14
Int simplePi = Intify(pi); // holds 3
```

H. Built-in Types

Ellie comes with three predefined attribute classes and four predefined shape classes. These classes are shown below.

```
Colour ovalColour = Colour(r: 70, g: 90, b: 155);
Loc ovalLoc = Loc(x: 5, y: 3);
Size ovalSize = Size(height: 10, width: 7);

Ellie oval = Oval();
Ellie rect = Rectangle();
Ellie line = Line(direction: “SW”);
Ellie string = EllieString(“Good job with the report!”);
```

This section is only to showcase the different built-in types of the Ellie Programming Lang. These shapes are the core elements for drawing any other custom shapes because you could craft basically any shape using lines and ovals.

Usage of these types will be shown in Section 3: Drawing Shapes.

I. Built-in Constants

Ellie comes with built in constants for Colour. Moreover, Ellie has special selectors for providing optional parameters to use in functions (see optional parameter definition in Section 4.A). Some examples are as follows:

```
Colour ovalColour = Colour.Turquoise;
Size ovalSize = Ellie.DefaultSize; //which is the size of the
Bounding Box.
Loc ovalLoc = Ellie.DefaultLoc; //Loc(x: 0, y:0)
Int defaultStroke = Ellie.DefaultStroke; //5
```

Moreover, you could modify these constants in your init() function before drawing any shapes.

```
init() {
     Ellie.DefaultStroke = 6;
     Ellie.DefaultLoc = Loc(x: 5, y: 5);
}
```

J. Decision Statements

Every programming language has one unique keyword. In Ellie, decision statements are unique in this sense. Instead of using **if** and **else**, like most of the other

programming languages, we picked more sophisticated keywords like **whether** and **proviso**. This way, we believe, we will be creating sophisticated thoughts on the user's mind, enabling them to think creatively.

```
whether ( examGrade == 100 ) {
     //congratulate
}
proviso whether ( examgrade <= 99 ) {
     //punish
}
proviso {
     //congratulate even more
}
```

K. Loops

Most programming languages have a similar for loop. This type of loop was actually derived from the C programming language and it takes a lot of effort and syntactical knowledge to create a simple iteration loop. In Ellie, for loops are as follows:

```
for index in 1...5 {
     //do something
}
```

This simple loop iterates from 0 to 4. The index is accessible by the **index** variable specified in the loop body. It is actually the same as writing (for int i = 0; i < 5 ; i++){} from the C Programming Lang.

While loops are similar with other programming languages. One example is as follows:

```
while ( examGrade < 100 ) {
     //work
}
```

3. Drawing Shapes

A. The Concept of a Bounding Box

In Ellie Programming Lang, the concept of a bounding box implies that; if a shape does not have a size, than it should be drawn inside the box which bounds it. In our case, this is the application window. If the shape does not have a specified Size, then it will be drawn to fit the bounding box (See section 4.A for concept of optional parameters).

B. Drawing an Oval

In the following example, we will be drawing 10 ovals next to each other, fitting all in the bounding box. First, we create an Oval, than we call its drawElli() method with the specified parameters.

```
    Ellie oval = Oval();
    /* draw 10 ovals next to each other horizontally, without
specifying absolute values for width and height */
    oval.drawEllie(count: 10);
```

```
    The next example shows drawing a single Oval inside the
bounding box using no parameters at all.
```

```
    Ellie oval = Oval();
    oval.drawEllie();
```

This is as simple it could get when drawing shapes with the Ellie Programming Lang. The power of optional parameters really show here. Drawing shapes with more parameters will be shown in the part of this section called Drawing a Rectangle.

C. Drawing a Rectangle

In the following example, we will be drawing a rectangle using all available parameters of the drawEllie() method.

```
    Colour rectColour = Colour(r: 70, g: 90, b: 155);
    Loc rectLoc = Loc(x: 10, y: 25);
    Size rectSize = Size(height: 30, width: 10);
    Ellie rect = Rect();
    rect.drawEllie(filledState: True, fillColour: rectColour,
    loc: rectLoc, size: rectSize, count: 2);
```

This example draws two rectangles horizontally with colour **rectColour**, location **rectLoc** and size **rectSize**. And the rectangles are filled with the **rectColour** as well since **filledState** is True.

D. Drawing a Line

Being different than other shapes, line takes one arbitrary parameter called direction. The accepted forms of this parameter are limited to “S”, “N”, “E”, “W”, “SE”, “SW”, “NE”, “NW”. An example for drawing a line is shown below.

```
    Loc lineLoc = Loc(x: 22, y: 90);
    Size lineSize = Size(height: 100); //width is an optional
parameter for the Size class
    Ellie line = Line(direction: "SW");
    line.drawEllie(loc: lineLoc, size: lineSize);
```

This example draws a line starting from location lineLoc and continues for a height of lineSize. The direction is specified as “SW” as seen in the Line constructor.

E. Drawing a String

Ellie language has different types called String and EllieString. String is for primitive operations on characters and EllieString is for drawing Strings. An example is as follows:

```
    Loc stringLoc = Loc(x:23, y: 88);
    Size stringSize = Size(height: 20); //the height of the text
is 20 pixels. Width depends on the height of the string
    Ellie stringDrawing = EllieString(“Hello, Ellie.”);
    stringDrawing.drawEllie(loc: stringLoc, size: stringSize); //
colour parameter is not provided here since it is an optional.
Defaults to black.
```

This example draws a String saying “**Hello, Ellie.**” with size **stringSize** and location **stringLoc.**

F. Drawing Custom Shapes

While supporting all the built-in shapes, Ellie Programming Lang. also supports defining custom shapes. However, definition of custom shapes requires an understanding of how the Core Features of the language work. Therefore, everything about drawing custom shapes is left for the next section.

In the next section, you will find the example of drawing a custom shape called a Snowman. Moreover, the next section covers some powerful language capabilities.

4. Core Features

A. Optional Parameters

Ellie Programming Lang. supports optional parameters like the Python Programming Language. The user could call a function without supplying all necessary parameters.

```
drawBugra(filledState = False, fillColour = Colour.Black, loc =
Ellie.DefaultLoc) {
     //do something
}
```

B. Intelligent Class Constructor

In many programming languages, developers need to write code to copy constructor arguments into the private variables in the language. In Ellie however, the interpreter automatically copies the constructor arguments into automatically created class variables. Here is a comparison with the Java Programming Language:

```
//assume Size and Loc classes are defined
public class Snowman extends Ellie {
     String message;
     Size snowmanSize;
     Size headSize;

     public Snowman(String message, Size snowmanSize) {
          this.message = message;
          this.snowmanSize = snowmanSize;
          this.headSize = new Size(snowmanSize.getHeight() *
          0.4);
     }
}
```

In Ellie, however,

```
Snowman -> Ellie {
     Snowman(message, snowmanSize); //Ellie class constructor
     Size headSize = Size(height: snowmanSize.Height * 0.4);
}
```

is all that is needed. The constructor creates the private variables in the background, without requiring the user to type the code. If needed, additional private variables can still be defined after the constructor. In this case, headSize is a private variable in the Snowman class.

C. Defining Classes

In Ellie, classes could be defined as follows. All classes related to drawing shapes must inherit from the Ellie class, which contains the drawEllie() method. After overriding the drawEllie() method, the user could call the method with optional parameters. The following code example may look messy because it does not fit on an A4 sized paper. However, it showcases the power of intelligent Ellie constructors and simple custom shape drawing syntax. Please take time to read it and understand fully before proceeding.

```
     Snowman -> Ellie { //We will be creating a class called
Snowman, which extends Ellie. We say “->” instead of "extends"

          Snowman(message);

          Ellie head = Oval();
          Ellie body = Oval();
          Ellie message = EllieString(message); //message will
have the default optional parameters

          //this is the perfect spot to show the optional
parameters
          drawEllie(filledState = False, fillColour =
Colour.Black, loc = Ellie.DefaultLoc, size = Ellie.DefaultSize)
{ //Ellie.DefaultLoc is provided as Loc(x: 0, y:0)
               Size headSize = Size(height: size.Height * 0.4)//
this is the ratio of the head over the body in case the user draws
this inside a bounding box

               drawOval(fs: filledState, fc: fillColour, l: loc,
s: headSize);
               drawOval(fs: filledState, fc: fillColour, l: loc,
s: snowmanSize);
               drawString(message);
          }

          //Oval's width is the same as the height unless
specified. It is an optional parameter as well.
     }
```

```
    //this will draw a snowman with head size 4, body size 10 and
the following message
    Size mrSnowmansSize = Size(height: 10);
    Ellie mrSnowman = Snowman(message: "Frozen 2 is coming!");
    mrSnowman.drawEllie(filledState: False, fillColour =
Colour.Black, size: mrSnowmansSize);
```

Here, we defined a class called Snowman which inherits from Ellie. The constructor takes two arguments. The types of the constructors are not know until the constructor runs. These arguments are automatically copied into hidden private variables after the contractor runs. Then the program continues with creating two Oval objects called head and body. Then, the an EllieString object gets created. If the user has supplied anything other than a String for the message argument in the constructor, Ellie will throw an exception and terminate. This way, we do type checking and at the same time, utilise the intelligent class constructors while still keeping the code clean.

D. Shapes Have No Attributes

Ellie Programming Lang.'s mother class (see section 2.A) defines a class called Ellie which contains an abstract drawEllie method. Since all shapes inherit from Ellie, they must implement their own drawEllie method. The drawEllie() method takes a size: Size, a location: Loc, a colour: Colour and a fillState: Bool.

Therefore, Ellie allows a very flexible way of drawing any shape. An example with an Oval is shown below:

```
    //Example for drawing an oval with specified fillState,
    location, colour and dimensions
    Colour ovalColour = Colour(r: 70, g: 90, b: 155);
    Loc ovalLoc = Loc(x: 5, y: 3);
    Size ovalSize = Size(height: 10, width: 8);
    Ellie oval = Oval();
    oval.drawEllie(filledState: False, fillColour: ovalColour,
    loc: ovalLoc, size: ovalSize);

    //the simplest way to draw an Oval inside the bounding box
    Ellie oval = Oval();
    oval.drawEllie();
```

As the example shows, the Oval object's constructor takes no parameters. Everything is controlled by the drawEllie() method. Of course, custom shapes like the Snowman from section 4.C supports arbitrary parameters (like the message parameter).

E. Elegant Getters and Setters

The Ellie programming language takes a new approach for getter and setter methods present in other programming languages. Private variables can be accessed as shown in the following example:

```
//Create a location and size for an Oval
Loc ovalLoc = Loc(x: 5, y: 3);
Size ovalSize = Size(height: 10, width: 8);

//get accessors
Int ovalLocX = ovalLoc.X;
Int ovalSize = ovalSize.Height;

//set accessor
ovalSize.Height = 7;
```

Notice these accessors do not have get or set prefixes like Java Programming Lang.

5. Conclusion

In conclusion, Ellie Programming Lang. Incorporates a Dynamically Typed language's advantages with a very focused syntax design. Since the language is centred solely on drawing shapes, we were able to create a highly specialised design pattern. With the Core Features provided, we believe this is a competitive candidate over complex programming languages. Combining simplicity, regularity, ease of use and high functionality, Ellie is an amazing tool for any person willing to work with drawing shapes. Also with the possibility of creating custom shapes and using them like the predefined ones, Ellie is sure to administer the needs of all users.