Shapes: Lab exercise III Part One

Chiel van Griethuysen (4014200) en Inès Duits(3930971)

January 6, 2014

Contents

1 Design **2**
1.1 ParserImport 2
1.1.1 ParserXML 2
1.2 ParserExport 2
1.2.1 ParserSVG 3
1.3 Shape 3
1.3.1 Rectangle 3
1.3.2 Circle 3
1.3.3 Star 3
1.4 ShapeDrawingForm 4

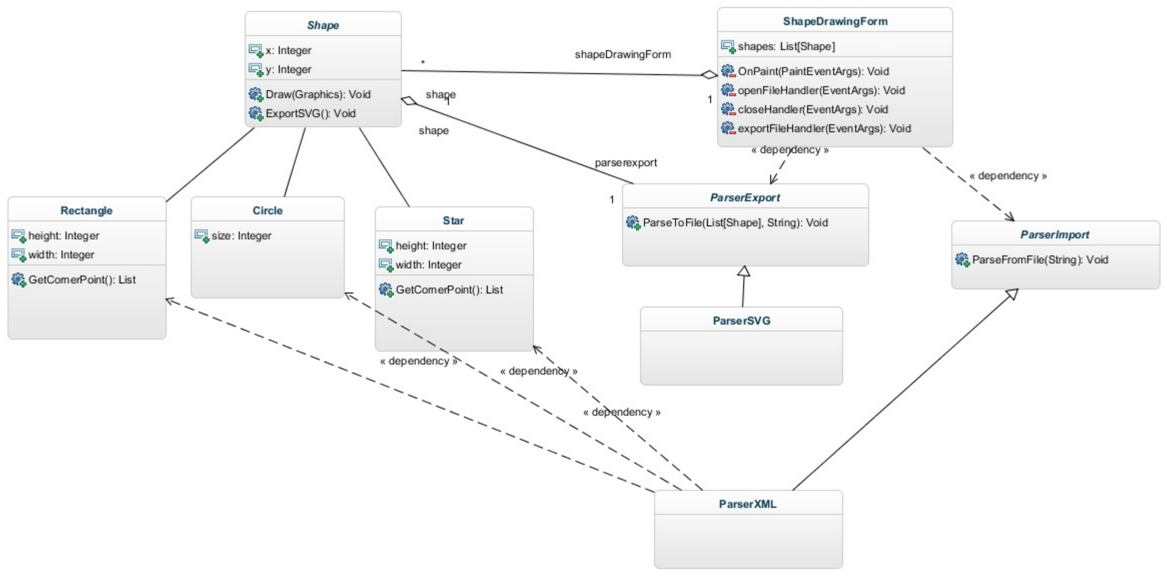
2 Design Pattern **4**
2.1 The Bridge Pattern 4
2.2 Strategy Pattern 4
2.3 Alternative 5
2.4 Requirement Changes 5

3 Evaluation **5**
3.1 Parser 5
3.1.1 ParserImport 5
3.1.2 ParserExport 5
3.2 Shapes 5
3.3 Requirement Changes 6

4 Refactoring play **6**
4.1 Pull Up: Shapes 6
4.2 Add Attribute 7
4.3 Extract Variable: GetCornerPoints 7
4.4 Introducing Abstract Class: ParserImport 7
4.4.1 Rename Method and Push Down Method 7
4.4.2 Change Method in ShapeDrawingForm 8
4.5 Introducing Abstract Class: ParserExport 8
4.6 Add new subclass: ParseSVG 8
4.6.1 Change Method in ShapeDrawingForm 8

1 Design

We created the follow class diagram



1.1 ParserImport

The abstract ParserImport class makes the importing from a file possible. The abstract class has the method ParseFromFile and the subclass ParserXML.

1.1.1 ParserXML

ParseXML inheritance the method called ParseFromFile which was first the ParseShapes method, which will do the same. ParseFromFile gets a file name, reads the figures from this file and create a list with that figures.

1.2 ParserExport

The abstract ParserExport class makes the exporting of a list with Shapes to a file possible. The class has the method ParseToFile and the subclass ParseSVG.

1.2.1 ParserSVG

This subclass inheritance the method called ParseToFile. The ParseToFile gets a list of figures and a name. The method creates a file with the given name and add the each shape from the list by using the ExportSVG method from the Shape class.

1.3 Shape

The abstract shape class contains all the methods and attributes a shape should have. We have the method Draw, which will draw the current shape. And the method ExportSVG, which will give the SVG-code, in the form of a String, needed to export that current shape. The attributes x and y, which represent the x coordinate and the y coordinate of where the shape will spawn. We made all the attributes from the subclasses public, because for exporting the shapes into a file we will need to know the explicit attributes.

1.3.1 Rectangle

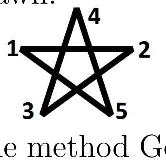
The Rectangle class has four attributes: height and width, and inheritance the attributes x and y. Which represent the height, width, the x coordinate and y coordinate of where the rectangle will be drawn.
We also add the method GetCornerPoint, which calculate the corners of this Rectangle. It will return a List of Points, which represent the corner point from upper left, clockwise, until bottom left.

1.3.2 Circle

The Circle class has three attributes: size, and inheritance the attributes x and y. The size represent the diagram of the circle.

1.3.3 Star

The Star class has four attributes: height and width, and inheritance the attributes x and y. Which represent the maximum height of the star, the maximum width of the star, the x coordinate and y coordinate where the star will be drawn.



We also add the method GetCornerPoint, which calculate the corners of this Star. The star we create has five corner points, which will be given in a

List of Points, starting with the most left (middle) one, then the most right (middle) one, then the left bottom one, then the top one and ending with the bottom right one. This particular order of the corner points will create the star if you draw lines between them. The lines between the corner points will form the star.

1.4 ShapeDrawingForm

This is the class which handles the events. It has an attribute which contains the current shapes. Also we expand the class with one extra method. We already had the methods OnPaint, openFileHandler and closeHandler, which handle representatively the event of painting the Shapes, open and get the shapes from a file and closing the program. The new method exportFileHandler, makes it possible to export the current shapes into a file.

2 Design Pattern

2.1 The Bridge Pattern

The Bridge Pattern decouples an abstract set of implementations from the object using them. Or when you look at our model, we have the abstraction Shapes decoupled from the abstract class ParserExport. The Shape uses the ParserExport to make sure it exports itself to the correct format, when a shape is needed to be exported. A Shape does not have to know to which file type it has to be exported. We implement a way for every Export file type to be correct exported for every Shape. When the Shape has the correct parserExport it uses this in its Export method to return the correct document format information. The only export file type is SVG, so only the ExportSVG is implemented. This method uses a method to create SVG cicles and a polyline for stars and rectangles.
By using the Bridge Pattern we can add new export file types and add new Shapes without changing our design. When we add a new export file type we only have to define for every shape how it should represent itself if it is exported. When adding new shapes, we only have to define how this new shapes should be exported to the available export file types.

2.2 Strategy Pattern

The Strategy Pattern defines a family of algorithms, encapsulate each one, and make them interchangeable. We try to think of a way to use this pattern, but there is not really a algorithm that varies. But you can see the translation from an import file to our Shape class, as a algorithm that varies depending on which file type the imported file has. Although we still have one file type we can import from, XML, we already have an abstract class ParserImport, which can get all kind of subclasses to represent new import file types.

2.3 Alternative

An alternative design we have considered is making one abstract Parser class that can both import and export. We found that for exporting we created an unnecessary switch, to determine of a Shape was a Rectangle, Circle or a Star. Now, used the Bridge Pattern, this switch is not longer necessary. For importing we do not yet have any shapes, not in the way it is implemented in the code, so we can not handle an import file the same way we handle an export file.

2.4 Requirement Changes

In this design of the program the only way to create shapes is by importing them from a XML file. But a program which can only import a XML file and export it as a SVG file seems very strange. So in the future it is very likely that we need to add a way to create shapes, like a paint program with buttons. But for this we need to add a lot of new classes, which will handle the mouse events when a user is drawing. But the new classes can use the already excising shapes in Shapes. And the shapes which are drawn can also easily be exported!

3 Evaluation

3.1 Parser

We created instead of one abstract Parser class, two different Parser classes. Because exporting a list of shapes is a different thing then importing a list of shapes, we made this decision.

3.1.1 ParserImport

We change the class Parser into an abstract class ParserImport and create a subclass, ParserXML, for it. The original Parser class already did what the subclass ParseXML schould do. We could have chosen to only add a new class for exporting shapes, ParserExport, but we decided to also change the

Parser class into the ParserImport class. Now it is easier to understand what the two Parser classes do and it is very easy to add new import file types in the future, by just adding a new subclass.

3.1.2 ParserExport

We create a new abstract class ParserExport. It subclass, ParserSVG, solves the main problem: We needed to find a way to export a list of shapes into a SVG file. This class can in the future handle all kind of ways to export the shapes, by adding a new subclass. But in this state it only can export a list of shapes into a SVG file.

3.2 Shapes

We add the method GetCornerPoints to the subclasses Rectangle and Star. In the draw method of star the corner points are calculated, but we also need that corner points to put a star into a SVG file. But then we have to, or duplicate our code, or create a method to calculate the corner points of the star, which is the GetCornerPoint. The same holds for the rectangle, but not for the circle. A circle has a own SVG function to create it, and otherwise a circle will consist of a infinite list of points.
The method ExportSVG for every Shape, make it possible to ask every shape to become SVG code, without the subclass ParserSVG having to know which kind of shape it is. This ExportSVG method uses for the shapes Star and Rectangle the GetCornerPoints function, which is also used by the Draw method of this shapes. So we do not duplicate the calculation.
With the abstract Shapes class it is also very easy to add new shapes. For every new file type we want to export shapes to, every shape should need a new method. We already need a way to transfer a shape from Shapes in something we can put in a SVG file. When we add another new file type we need a new interpretation of every shape in Shapes, which can be like an other implementation but also can be very different. We already made this abstracter using the GetCornerPoints method. Depending on which file types we want to export to in the future, we can add more of that kind of methods.

4 Refactoring play

Here we will describe our refactoring plan. The following sections, which contains the changes, are in order.

4.1 Pull Up: Shapes

We see that the attributes in Rectangle, Circle and Star for the x and y position are all integers with the name x and y, and represent the appear position for this Shapes. Every new Shape should also need a x and y position. Therefore we can pull this attribute up to the abstract Shape class.
Now we will compile the program to see if we didn't make a mistake.

4.2 Add Attribute

The original ShapeDrawingForm class suggest to have a Shapes. But because it is not shown in the original Design we introduce the attribute containing a List of Shapes. We need a way to store the shapes the ShapeDrawingForm class have to draw, and we think there was maybe another place where it was stored. But now we want to store it in a public attribute called shapes. It is useful because we need to give the shapes to the Parser class.

4.3 Extract Variable: GetCornerPoints

We create the new methods, GetCornerPoints, in Star and Rectangle.
For Star we will get the original code of his Draw function, transform it in a code to calculate the corner points and place it in the GetCornerPoints method. Then we will change draw method of Star, instead of using the complicated calculation, using the GetCornerPoints. Then we will do the same for Rectangle.
Although the change is not big, testing and compiling the code on this point should be done. We need to know if the GetCornerPoints indeed calculate the correct corner points of a star and a rectangle. Also it is crucial that the draw methods of star and rectangle still draw the same star and rectangle. These two points should be tested.

4.4 Introducing Abstract Class: ParserImport

The original program has only the class Parser, with the method ParseShapes. We will make it abstract and create the subclass ParseXML.

4.4.1 Rename Method and Push Down Method

The original Parser class already had the possibility to import a XML file with the method ParseShapes and thus we use that method in the subclass ParseXML. We choose to use this method but we will rename it to ParseFromFile, to make its purpose clearer. We will also define it already in the

Abstract class, every new subclass (added in the future) should also have that method.

4.4.2 Change Method in ShapeDrawingForm

We changed the name and the place of the original ParseShapes method. The ShapeDrawingForm classes called that method in his openFileHandler method. So we need to change the calling of the old ParseShape method into a calling for the new ParseFromFile method.

Now it is very important that we test and compile the code. We did not really rewrite the original code from the method ParseShapes, but we need to make sure the reading from an XML file still works before we try to make the SVG export part. The class ShapeDrawingForm also need to call the right ParseFromFile method, instead of the old ParseShapes method which at this point doesn't exists any more.

4.5 Introducing Abstract Class: ParserExport

We will introduce a new abstract class called the ParserExport with the method ParseToFile.

4.6 Add new subclass: ParseSVG

Then we can add the subclass ParseSVG, which will implement a way to export every shape. The method ParseToFile will use the ExportSVG method for every Shape it needs to export, So there is no need for a switch in this method!

4.6.1 Change Method in ShapeDrawingForm

We need a way to call the new introduced method, ParseToFile. This will be done from the ShapeDrawingForm class if the user want to export the current file. Introducing a new method in ShapeDrawingForm called exportFileHandler, will handle the event if the user clicks export.

Then we will have to compile and test the code! Because this is the point we introduces a whole new code, the export of the shapes into a SVG file. We need to make sure it works, and the shapes are exporting the right way.