**Shape Factory:**

**Introduction:**

The Shape Factory code is a C# class that is used for creating instances of different shapes. The class has been created to remove any dependencies that might exist between the shapes and their implementation. This documentation will explain how the Shape Factory works, the input parameter, and the output that is expected from this code.

**Class Definition:**

The Shape Factory class has been defined as "ShapeFactory". This class contains a single public method named "getShape". The getShape method is used for retrieving the shape objects to be instantiated based on the string passed as input.

**Method Description:**

The getShape method is used to retrieve the shape object that is to be created. This method takes a string as input, which is used to compare against different cases. Each case corresponds to a shape object that is to be created.

**Method Input Parameter:**

The getShape method takes a single string parameter named "shapeType". This parameter is used to compare against different cases in the switch statement.

**Method Output:**

The getShape method returns an instance of the "Shape" class. The Shape class is an abstract class, which is extended by different shapes such as Circle, Rectangle, Triangle, and Line. The return type is determined based on the switch case that matches the input string.

**Method Exceptions:**

If none of the cases are matched, the method throws an ArgumentException. The ArgumentException contains an error message that is to be displayed to the user. The message is "factory error: " + shapeType + "Does not exist". The ArgumentException is defined in the System namespace.

**Conclusion:**

The Shape Factory code is an essential class that is used for creating instances of different shapes. This code helps in removing any dependencies that might exist between the shapes and their implementation. The documentation provided in this file should help in understanding how the Shape Factory code works, the input parameters, the output, and the exceptions that can be thrown.

**File name: Shapes.cs**

**Namespace: ASEProject**

**Classes: None**

**Interfaces: Shapes**

**Summary:**

This file contains an interface called Shapes, which defines two methods: set and draw. This interface is used by various shape classes to implement their functionality.

**Class/Interface: Shapes**

**Methods:**

void set(Color c, params int[] list)

**Summary:**

This method is used to set the color and coordinates of the shape. It takes a Color object and a variable number of integers as input.

**Parameters:**

c: A Color object representing the color of the shape.

list: A variable number of integers representing the coordinates of the shape. The number of integers depends on the type of shape.

Return type: void

void draw(Graphics g)

**Summary:** This method is used to draw the shape. It takes a Graphics object as input.

Parameters:

g: A Graphics object representing the graphics context on which the shape is to be drawn.

Return type: void

**Parser Class:**

This is a C# class called **Parser** which is used to parse commands for a graphics program.

The class has a constructor which takes an object of type **Canvas** as a parameter, and a public method called **Read** which takes a string parameter representing the command to be parsed.

The **Read** method first formats and tidies the input command. Then it splits the command into a command string and an array of parameter strings. The parameter strings are then parsed to integers where necessary.

Next, the method uses a switch statement to check the command and execute the appropriate method on the **Canvas** object passed in to the constructor. If the command is not recognized, an **InvalidOperationException** is thrown.

If there are any issues with the parameters of the command, the method displays an error message using the **MessageBox** class from the **System.Windows.Forms** namespace.

The class seems to provide a good foundation for parsing commands and drawing graphics on a canvas.