Version 1.2 March 17, 2019

Design Document

SWE 205 PROJECT – TEAM 3

Table of Contents

Introduction

I. Introduction to the product

MyPaintShop is a tool that allow users to select shapes and draw them on a canvas. Once the shape is drawn on the canvas, it can be selected and edited or filled with color. Information about the shape is displayed at the bottom of the screen such as area.

II. Purpose of this document

The purpose of this document is to document the design of MyPaintShop. Documentation of the design of the software will include the software architecture, user interface design and component design.

III. Overview of the remainder of the document

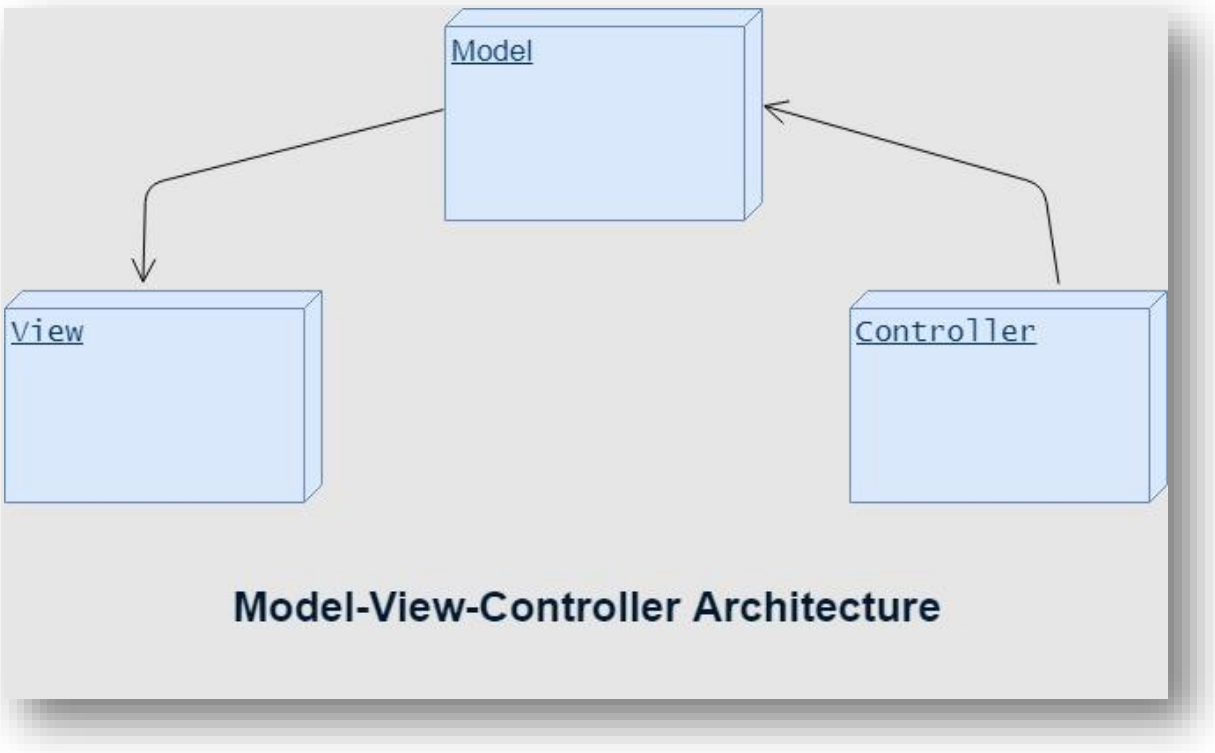
The remaining sections of the document will be as follows:

- **Section 2** will discuss the software architecture through block diagrams of sub-systems with brief descriptions of them and the justification for architecture choices.
- **Section 3** will preview the user interface design to demonstrate how the user accesses the major functionalities in the software.

- **Section 4** will discuss the component design of the software and express it in terms of class diagrams and descriptions for each of them (attributes and methods).
- **Section 5** is where all the references used to write this document would be listed.
- **Section 6** is where all the related appendices to the software would be written.

Software Architecture

❖ **Block diagram of the architecture :**



❖ **Justifications for the architecture choice**:

We have chosen the Model-View-Controller (MVC) architecture primarily for three reasons:

1. **Divide-and-Conquer Strategy**:
   This architecture is an example of this valuable strategy, where huge systems are fragmented into smaller sub-systems, each with distinctive roles, leading to a more maintainable and organized software.

2. **Separation and Independency**:
   In MVC architecture, changes in one sub-system are independent of the other sub-systems. For instance, same data in the Model can be represented in different Views.

3. **Suitable for Graphic User Interface (GUI) systems**:
   As the name of the architecture indicates, it is very useful for systems that have graphical visualization of the data. Since this project depends largely on the user interface, it would be very helpful to have distinct sub-system –the View- that deals with the entire nuisance of GUI.

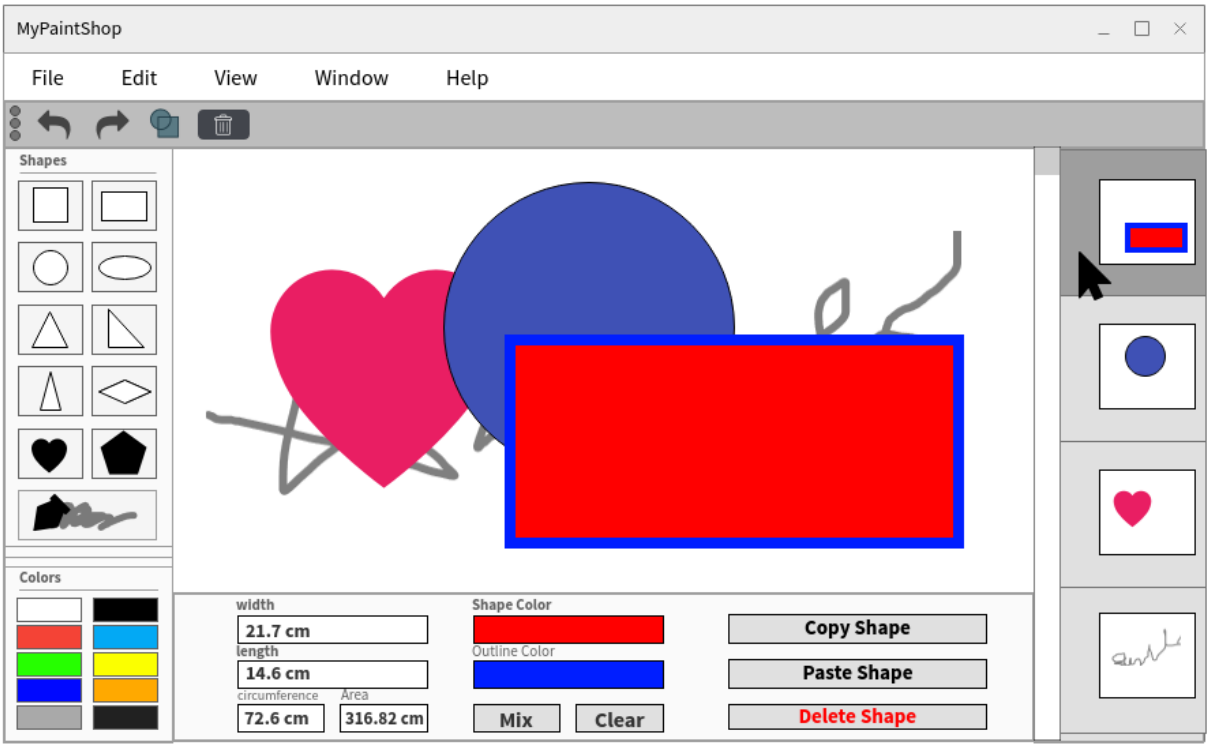
❖ **Brief descriptions of the sub-systems:**

1. **Model sub-system:**
   The model contains the core logic of the software, and manages the system data.

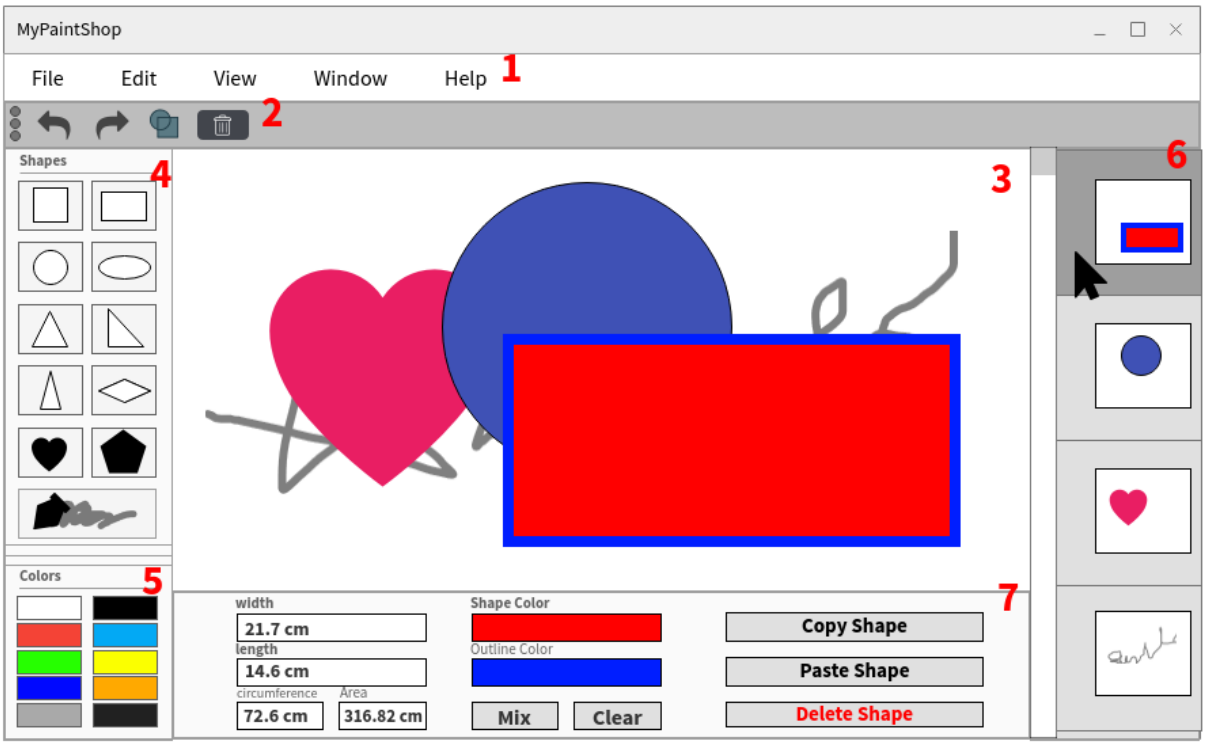
2. **View sub-system:**
   The view is responsible for the data representation; it display the current state of the model.

3. **Controller sub-system:**
   The controller is responsible for the user interactions with the systems; it conveys the presses and clicks to the model, and changes the model state.

User Interface Design



While designing the user interface, we tried to follow the most important principles and guidelines to create a simple and efficient interface. To explain more, we can divide this interface into seven parts as shown below:



1 – **Menu Bar**

As shown in the picture above, the bar is divided into **5 parts**: *File*, *Edit*, *View*, *Window*, *and Help*. We tried to reduce the number of lists to keep everything clear in the user's short-term memory as we are going to explain right now.

- ***File***
  - **New**: *for creating a new canvas.*
  - **Open**: *to open a saved file.*
  - **Save**: *to save the progress of the current file.*
  - **Save as**: *to save it as a different file.*
  - **Exit**: *to exit the software.*
- ***Edit***
  - **Shape:** for every command related to the shapes. Shape menu contains:
    - **New Shape:** *to create a new shape.*
    - **Edit Shape:** *to edit the properties of the current shape.*
    - **Copy Shape:** *to copy the current shape.*
    - **Paste Shape:** *to paste the copied shape if there is any.*
    - **Delete Shape:** *to delete the current shape.*

  - **Color:** for every command related to colors and their properties. It contains:
    - **Shape Color:** *to modify the current color of the shape.*
    - **Outline Color:** *to modify the current color of the shape's outline.*
    - **Mix Color:** *to mix between two colors then using it to fill the shape.*
    - **Clear Color:** *to set the shape's color to None and outline's to Black.*
  - **Layer:** for every command related to layers. Layers menu contains:
    - **Add Layer:** *adds a new empty layer to the canvas.*
    - **Clone Layer:** *copies the current layer and pasts it in one move.*
    - **Delete Layer:** *delete the current layer.*
  - **Undo:** to go back one operation at time.
  - **Redo:** to go forward one operation at a time.
- ***View***
  - **Shape List:** *If not checked, clicking on it will show the shape list (No. 4)*
  - **Color List:** *if not checked, clicking on it will show the color list (No. 5)*
  - **Layer List:** *if not checked, clicking on it will show the layer list (No. 6)*
  - **Zoom:** *you can adjust the field of your vision by using this button.*
- ***Window***
  - **Minimize:** *to hide the software to Window's Taskbar.*
  - **Maximize/Restore Down:** *to make the software take the Fullscreen/part of the screen.*
  - **Close:** *to close the software.*
- ***Help***
  - *To provide the user with the help needed to use the software efficiently.*

As you can see above, every menu is designed with **less than two clicks** to get the job done for the user. While maintaining the important parts that appears to the user to be **less than seven** in total.

2 – **Toolbar**

This bar, which is located directly below the menu bar, contains the most useful commands that the user may use, from left to right we can see the *Undo, Redo, Paste Shape, and Delete Shape* buttons. The toolbar here is wide to allow the user to add the features that that feels important to the user`s work*.

**this feature is not available in this version*

3 – **Canvas**

4 – **Shapes List**

This list contains **ten** shapes and one tool that the user may use. These tools are:

- ***Square***
- ***Rectangle***
- ***Circle***
- ***Ellipse***
- ***Right Triangle***
- ***Equilateral Triangle***
- ***Isosceles Triangle***
- ***Rhombus***
- ***Pentagon***
- ***Heart***
- ***Free Drawing Pen***

5 – **Color List**

This list will provide multiple of choices when it comes to colors. When implementing later, we'll add more colors and choices. For now, the list will provide pre-defined colors, and the user will choose one or two* of them to fill the shape with it.

**selecting two colors at once will result in mixing the colors, thus filling the shape with the resultant color.*

6 – **Layer List**

This list will display the layers of the drawing in order, from top to bottom. Every time you draw a shape, a new layer will be added to the layer list. You can't

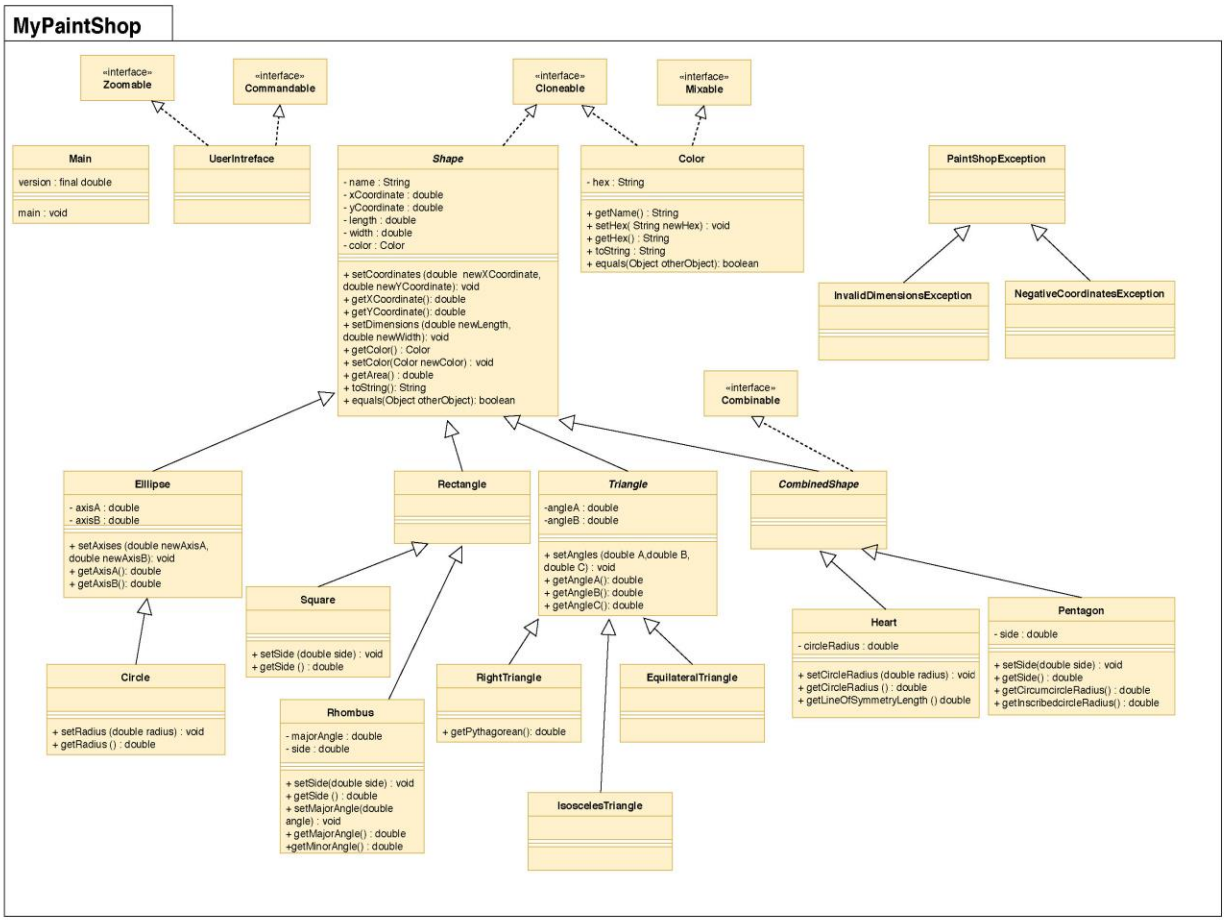
have multiple shapes in one layer for now, thus, copying and pasting the shape will affect the layers as well.

7 – **Properties Box**

When double-clicking an object, a box at the bottom of the screen will appear. This box will show many properties like displaying the width, length, circumference, area, shape's color, outline's color, the choice to mix colors or clear the shape, copying, pasting, and deleting the shape. As you change the properties of the shape, the results will be rendered in real time in the screen.

Component Design

I. Class Diagrams



II. Descriptions of Classes

- **<u>Interfaces</u>**:
  - **Zoomable**: This interface tags the class as “zoomable” so it would have the required methods to be zoomed-in or out.

- **Commandable**: This interface contains all the methods and commands that that the object implementing it should have.
- **Cloneable**: The shape that implements this interface have the required methods to be copied "cloned".
- **Mixable**: The Color class implements this interface for its objects to be mixed to result in a new color.
- **Combinable**: classes implementing this interface can be use other shapes and combine them to result in a new shape.

- **Exceptions**:
  - **PaintShopException**: Super class for all other exceptions related to the program.
  - **InvalidDimensionsException**: subclass of PaintShopException, it indicates that a problem related to a shape`s dimensions occurred. For example: negative dimensions.
  - **NegativeCoordinatesException**: The canvas is in the 1st coordinate which is positive in the x and y axis, so inputting a negative coordinate will result in an exception.

- **Abstract** Classes:
  - **Shape**: This abstract class contains all the data shared by all child classes, its fields are:
    - ***name***: the name of the shape as a String.
    - ***xCoordinate***: the x coordinate of the shape as a double.
    - ***yCoordinate***: the y coordinate of the shape as a double.
    - ***length***: the length of the layer of the shape as a double.
    - ***width***: the width of the layer of the shape as a double.
    - ***color***: the represented color of the shape as a Color (Color class described further on).

    It also contains the required constructers, getters, setters, toString and equals methods.
  - **Triangle**: This abstract class inherits the Shape abstract class, so it has all its fields and methods. It contains all the data shared by different kinds of triangles and contains the following fields:
    - ***angleA***: represents one of the angles of the triangle as a double (in degrees).

- ***angleB***: represents the other angle of the triangle as a double (in degrees).

  It also contains the required constructers, getters, setters, toString and equals methods.

- **CombinedShape**: represents a shape consisting of multiple combined shapes. It inherits the Shape class and has no extra fields. It has a single method that takes the given shapes, as instance objects, and combine them.

- **Concrete Classes**:
  - **Color**: This class represents the color of the shape using a hex code, it implements the Mixable interface to be able to generate a new Color by mixing two Colors. It also implements the Cloneable interface for it to be copied when copying a shape. It also contains the following fields:
    - ***hex***: the hex code of the desired color as a String.

    It also contains the required constructers, getters, setters, toString and equals methods.
  - **Ellipse**: This class represents the shape "ellipse", it inherits the Shape class, so it has all its fields and methods and implements all the interface the Shape class implements. It also contains the following fields:
    - ***axisA***: which represents one of the axes of the ellipse as a double.
    - ***axisB***: which represents the other axis of the ellipse as a double.

    It also contains the required constructers, getters, setters, toString and equals methods.
  - **Circle**: This class represents a special case of an ellipse, the "circle". It inherits and implements all what the superclass has and all what it implements. It needs no extra fields since a circle occurs when both axes of the ellipse are equal. it uses one of them and represent it as a "radius".

    It also contains the required constructers, getters, setters, toString and equals methods.

- **Rectangle**: This class represents the shape “rectangle” it inherits the Shape class and does not need any extra fields because it uses the width and height of its layer as its dimensions.
  It also contains the required constructers, getters, setters, toString and equals methods.
- **Square**: This class represents a special case of a rectangle, the “square”. It inherits the Rectangle class. It needs no extra fields since a square occurs when the length and width of a rectangle are equal. It uses one of them and represents it as a “side” length.
  It also contains the required constructers, getters, setters, toString and equals methods.
- **Rhombus**: This class represents a special case of a rectangle, the “Rhombus”. It inherits the Rectangle class and has the following extra fields:
  - majorAngle: represents the grater angle as a double (in degrees).

  It also contains the required constructers, getters, setters, toString and equals methods.
- **RightTriangle**: This class represents a right triangle and inherits the class “Triangle”, it requires no extra fields and has a single method “getPhythagorean’ which does apply Pythagorean theorem to get the missing side.
- **EquilateralTriangle**: This class represents a triangle with equal sides and inherits the class “Triangle”, it requires no extra fields and methods.
- **IsoscelesTriangle**: This class represents a triangle with two equal sides and inherits the class “Triangle”, it requires no extra fields and methods.
- **Heart**: This class represents the combination of two circle and an isosceles triangle. It inherits the CombinedShape class and has the following extra fields:
  - circleRadius: which represents the radius of the circles as a double

  Methods It also contains the required constructers, getters, setters, toString and equals methods.

- **Pentagon**: This class represents the combination of five isosceles triangles. It inherits the CombinedShape class and has the following extra fields:
  - side: which represents the sides of the combined shapes.

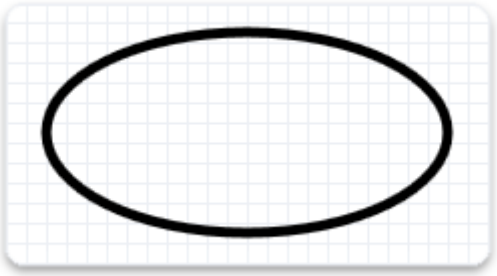
  It also contains the required constructers, getters, setters, toString and equals methods.

References

- **How to use geometry in code**: https://docs.microsoft.com/en-us/windows/desktop/direct2d/direct2d-geometries-overview
- **Information about ellipses**: https://www.mathsisfun.com/geometry/ellipse.html
- **Sample of a circle**: https://commons.wikimedia.org/wiki/File:Circle_-_black_simple.svg
- **Sample of a rectangle**: https://en.wikipedia.org/wiki/File:Rectangle_Geometry_Vector.svg
- **Information about rectangles**: https://www.mathsisfun.com/geometry/rectangle.html
- **Information about squares**: https://en.wikibooks.org/wiki/Geometry_Course/Polygon/Quadrilateral/Square
- **Sample of a rhombus**: https://en.wikipedia.org/wiki/File:Rhombus.svg
- **Information about rhombuses**: https://www.mathsisfun.com/geometry/rhombus.html
- **Information about triangles**: https://www.mbacrystalball.com/blog/2015/10/16/triangles-properties-types-geometry/
- **Sample of a right triangle**: https://commons.wikimedia.org/wiki/File:Triangle.Right.svg
- **Sample of an isosceles triangle**: https://en.wikipedia.org/wiki/File:Triangle.Isosceles.svg
- **Information about Pythagorean theorem**: https://www.mathsisfun.com/pythagoras.html
- **Sample of a polygon**: https://en.wikipedia.org/wiki/File:Regular_polygon_5_annotated.svg
- **Information about pentagons**: http://mathworld.wolfram.com/RegularPentagon.html
- **Sample of a pentagon**: https://en.wikipedia.org/wiki/File:Regular_pentagon_1.svg
- **SWE205 course book**: Sommerville, I. (2016). *Software engineering*. Harlow: Pearson Education.
- **ICS201 course book**: Savitch, W. J., & Mock, K. (2016). *Absolute Java*. Brantford, Ontario: W. Ross MacDonald School Resource Services Library.

Appendices

Ellipse:

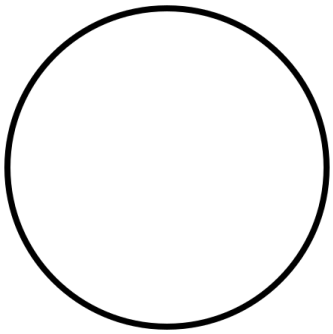


The Major Axis is the longest diameter. It goes from one side of the ellipse, through the center, to the other side, at the widest part of the ellipse. And the Minor Axis is the shortest diameter (at the narrowest part of the ellipse).

The Semi-major Axis is half of the Major Axis, and the Semi-minor Axis is half of the Minor Axis.

The area of an ellipse is: π × (Semi-major Axis) × (Semi-minor Axis)

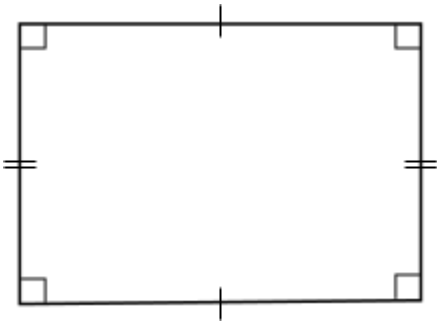
Circle:



If (Major Axis == Minor Axis) in Ellipse, then the Ellipse will be a Circle.

From the formula of ellipse area, we can get the area of circle by: π × raiduis2

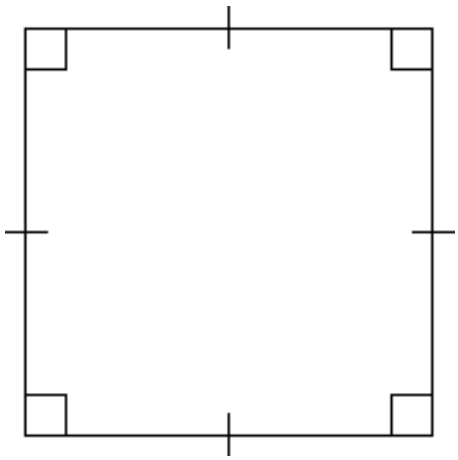
Rectangle:



A rectangle is a four-sided flat shape where every angle is a right angle (90°).

Area = width × height

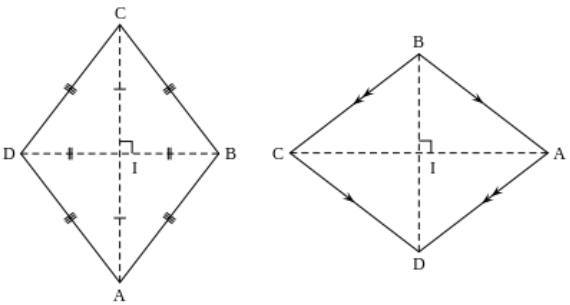
Square:



If (width == height) in Rectangle, then the Rectangle will be a Square.

From the formula of Rectangle area, we can get the area of Square by: Area =side^2

Rhombus:

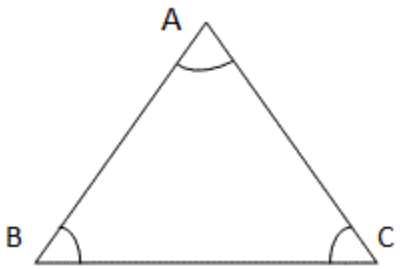


A Rhombus is a flat shape with 4 equal straight sides.

The Area can be calculated by multiplying the lengths of the diagonals and then dividing by 2:

Area = (Diagonal_A × Diagonal_B)/2

Triangles:



A triangle is a closed figure made up of three-line segments. In the figure above, AB, BC, CA are the three-line segments and ∠A, ∠B, ∠C are the three angles.

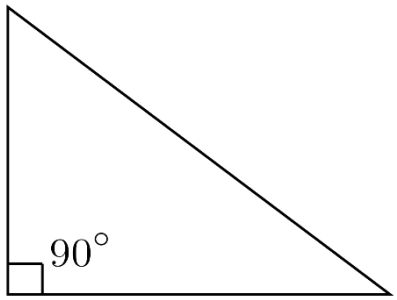
Basic properties of triangles:

- The sum of the angles in a triangle is 180
- The sum of the lengths of any two sides of a triangle is greater than the length of the third side. Similarly, the difference between the lengths of any two sides of a triangle is less than the length of the third side.

We are using, in this project, 3 types of triangles based on sides and three based on angles.

Equilateral triangle: A triangle having all the three sides of equal length is an equilateral triangle. (See the figure above)

Right triangle:



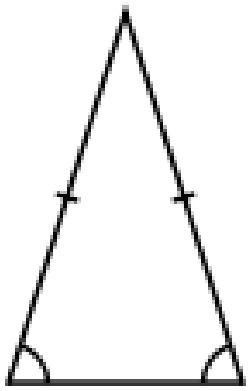
A triangle whose one angle is a right-angle is a Right-angled triangle or Right triangle.

Pythagoras' Theorem: In a right-angled triangle: the square of the hypotenuse is equal to the sum of the squares of the other two sides.

Assume: c is the longest side of the triangle and a and b are the other two sides

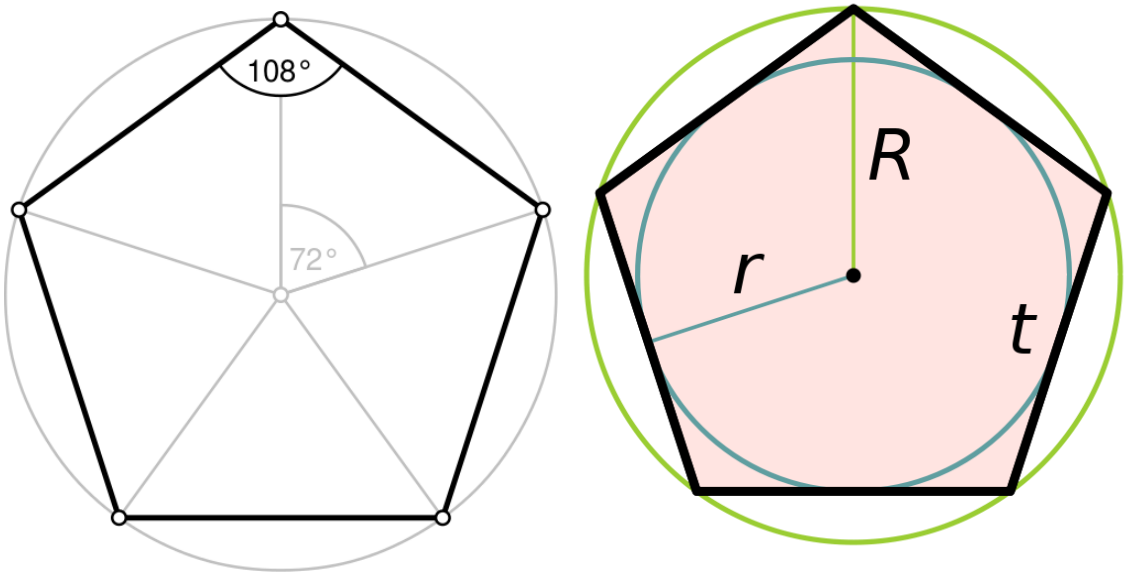
So: a2+b2=c2

Isosceles triangle: A triangle having two sides of equal length is an Isosceles triangle.



The two angles opposite to the equal sides are equal.

Regular pentagon:



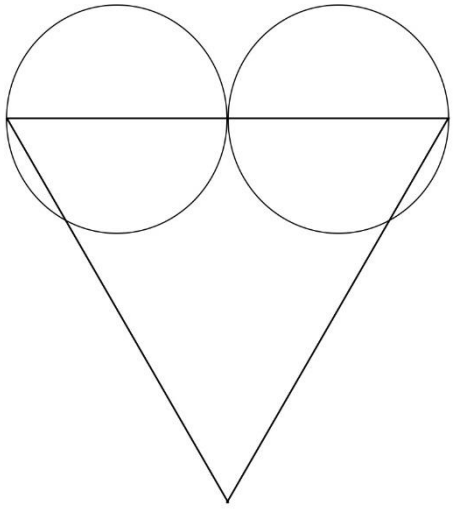
The Regular pentagon is the regular polygon with five sides

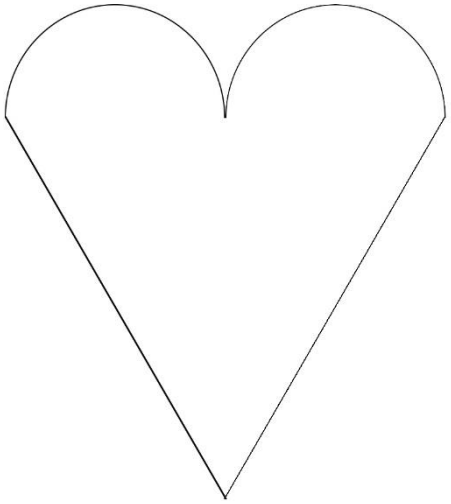
A number of distance relationships between vertices of the regular pentagon can be derived by similar triangles in the above figure

To form a pentagon, we must use 5 Isosceles triangles with two **54**° angles and a **72**°

We can have an inner circle with radius r and an outer circle with radius R (See the above figure)

Triangular hurt:





Our team invented a new type of hurt shape which made implementing hart easy in this project.

This hurt is made of two semicircles and one isosceles triangle with side length equals to two diameters (four radiuses).

Its area is the area of one circle + the area of the triangle.

Distribution of Work

Member	Tasks	Participation
HAITHAM ALSAEED	▪ Classes Diagrams ▪ Appendices	20%
HASHIM ALGHAMDI	▪ Introduction ▪ References	20%
HUSSAIN HAJJI	▪ Descriptions of Classes and Methods.	20%
SALMAN ALGHAMDI	▪ User Interface Design	20%
SALEM BAMUKHIER [Team Leader]	▪ Software Architecture ▪ Document Editing	20%

Team Meetings

Date	Duration	Attendance
March 11, 2019	8:00 – 11:00 PM	HAITHAM ALSAEED HASHIM ALGHAMDI HUSSAIN HAJJI SALMAN ALGHAMDI SALEM BAMUKHIER