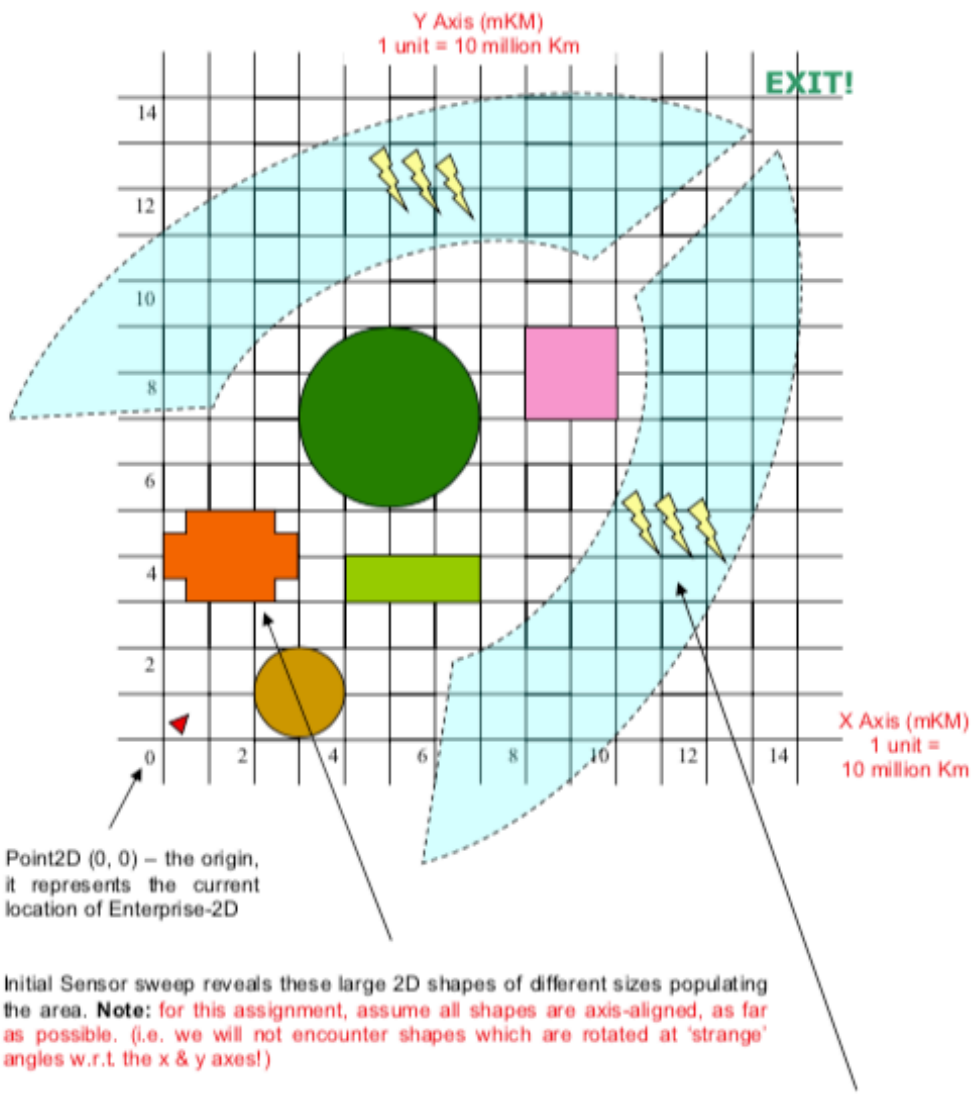
**Background**

As a Science Officer aboard Enterprise-2D, you need to develop a program that has the following capabilities:

a) read in sensor data on the strange 2D shapes (via manual input)
b) compute area ('mass') of these shapes
c) print shapes report (e.g. list of points: on its perimeter, or totally within shape's area)
d) sort shapes data (sorted by special type and area)

The next section provides information about the requirements for developing this program:

<u>Coordinate System</u>



Point2D (0, 0) – the origin, it represents the current location of Enterprise-2D

Initial Sensor sweep reveals these large 2D shapes of different sizes populating the area. **Note:** for this assignment, assume all shapes are axis-aligned, as far as possible. (i.e. we will not encounter shapes which are rotated at 'strange' angles w.r.t. the x & y axes!)

Enterprise-2D is trapped in this huge ring of electrical plasma storm. The only opening to exit this storm is located in the far 'upper-right' of the coordinate system, for e.g. at Point2D (14, 14) !

## Description of Sensor Data

### Name

The name of the 2D shape reveals the general type of shape encountered. Currently, the values consist of : “Square”, “Rectangle”, “Cross” and “Circle”.

### Special Type

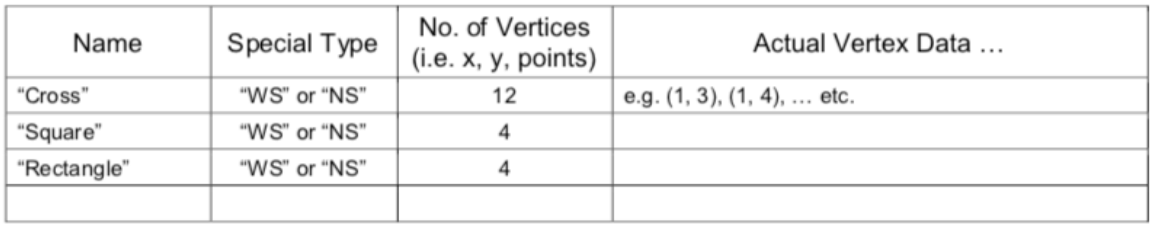
Enterprise-2D’s sensor has detected that some shapes encloses a ‘warp-space’ with the amazing ability to teleport any objects that touches one of its vertex, to any other vertices of the same shape, instantaneously !

This makes it highly desirable, for Enterprise-2D to travel towards this kind of shape, in the hopes of travelling faster, and saving precious fuel at the same time!

There are only 2 values for ‘special type’ : “WS” (Warp-Space) or “NS” (Normal-Space).

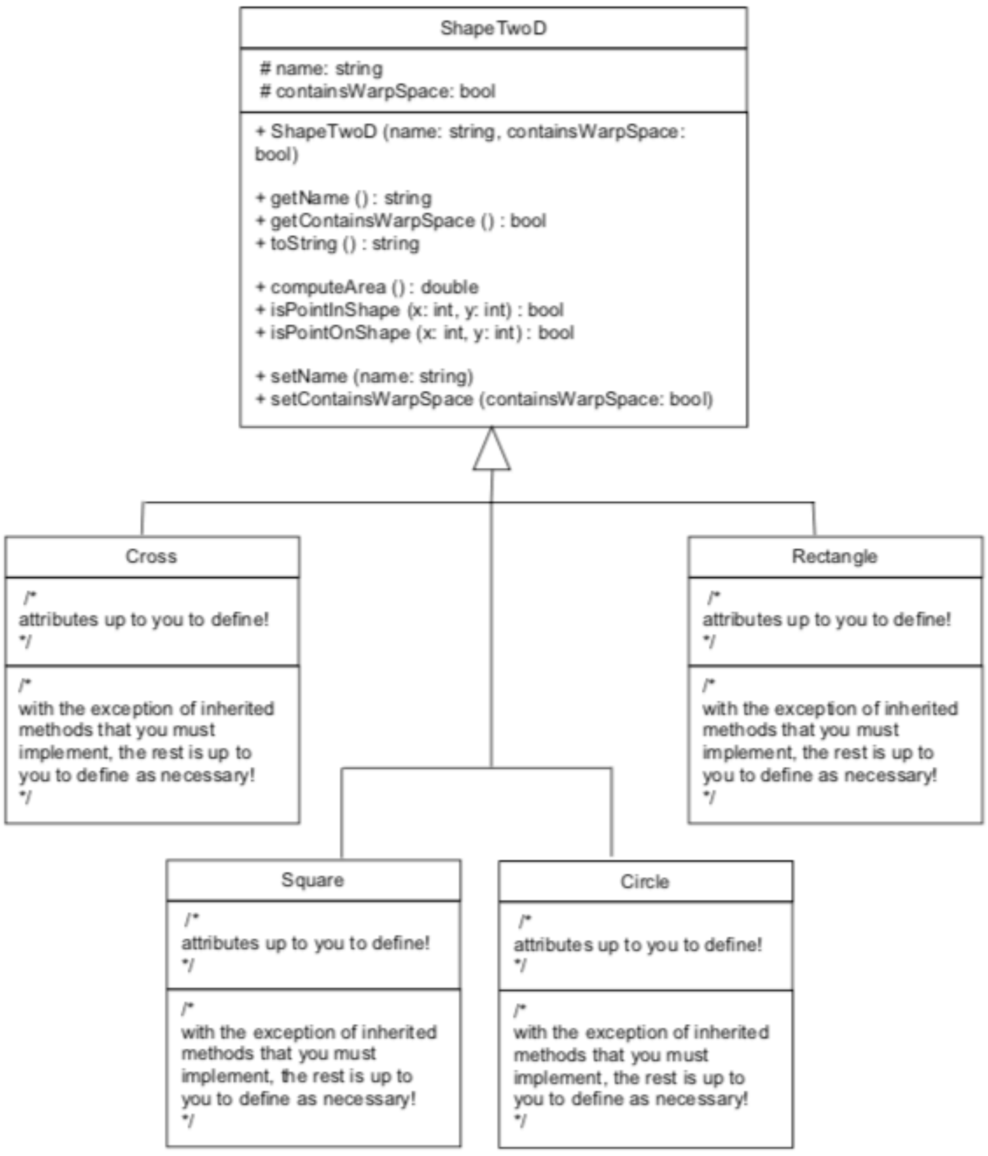
### Vertices

The vertices is actually a set of (x, y) points, that describes the outline of the 2D shape. The number of points in the set, depends on the name of the shape. The table below summarizes the kind of sensor data your program expects to receive.



| Name | Special Type | No. of Vertices (i.e. x, y, points) | Actual Vertex Data … |
|---|---|---|---|
| “Cross” | “WS” or “NS” | 12 | e.g. (1, 3), (1, 4), … etc. |
| “Square” | “WS” or “NS” | 4 | |
| “Rectangle” | “WS” or “NS” | 4 | |
| | | | |

## Class Implementation


ShapeTwoD
# name: string
# containsWarpSpace: bool
+ ShapeTwoD (name: string, containsWarpSpace: bool)
+ getName () : string
+ getContainsWarpSpace () : bool
+ toString () : string
+ computeArea () : double
+ isPointInShape (x: int, y: int) : bool
+ isPointOnShape (x: int, y: int) : bool
+ setName (name: string)
+ setContainsWarpSpace (containsWarpSpace: bool)
Cross
Rectangle
Square
Circle
/* attributes up to you to define! */
/* with the exception of inherited methods that you must implement, the rest is up to you to define as necessary! */

## Compulsory Requirements

#1 The parent class is '`ShapeTwoD`'. Any attributes, constructors and methods specified in the diagram must be implemented, with the exact same name, parameter, type and access!

#2 The sub-classes of `ShapeTwoD` must be named '`Cross`', '`Square`', '`Circle`' and '`Rectangle`'.

#3 The method '`toString()`' in class `ShapeTwoD` is a virtual function that returns a string containing all the values of the attributes in the shape, excluding the array of vertex data. (However, sub-classes of `ShapeTwoD` must output the array of vertex data, inclusive of any other attribute's values it inherited)

#4 The method '`computeArea()`' in class `ShapeTwoD` is a virtual function. It must be override by the sub-classes and implemented individually.

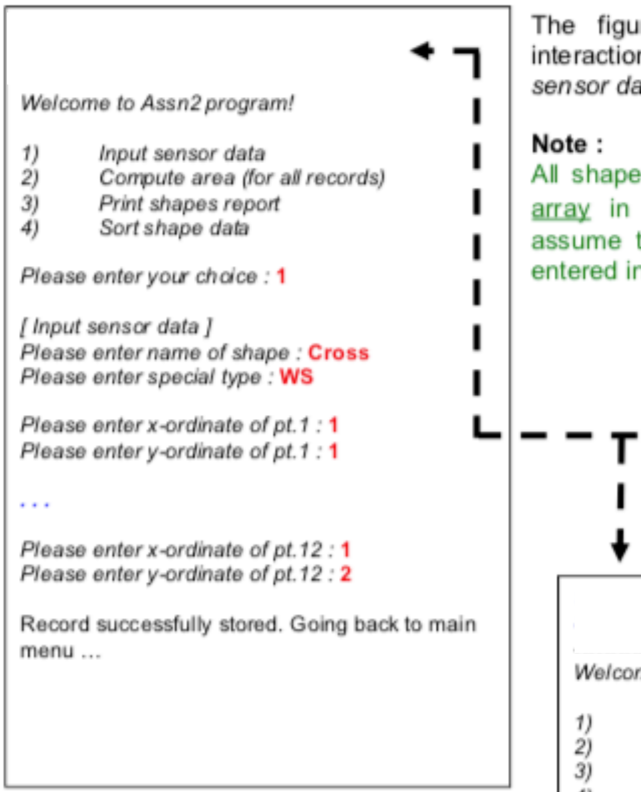
> For example, because each sub-class has different number of vertices and values, sub- class `Cross's computeArea()` implementation would use a different algorithm / formula from sub-class `Square`'s `computeArea()` implementation!
>
> Note : The sensor data will only provide the locations (vertices) of each 2D shape encountered. You will be required to do the necessary research to derive the formula to compute the area for each shape, based on the set of vertex data (i,e. a set of [ x, y ] points) provided!

#5 The method '`isPointInShape()`' in class `ShapeTwoD` is a virtual function. It takes in a [ x, y ] location and returns a boolean value indicating whether the location is totally within the shape's area. It must be over-ridden by the sub-classes and implemented individually. (Pls refer to sample output)

#6 The method '`isPointOnShape()`' in class `ShapeTwoD` is also a virtual function. It takes in a [ x, y ] location and returns a boolean value indicating whether the location is found on any lines joining the shapes' vertices! It must be over-ridden by the sub-classes and implemented individually. (Pls refer to sample output)

## Sample output

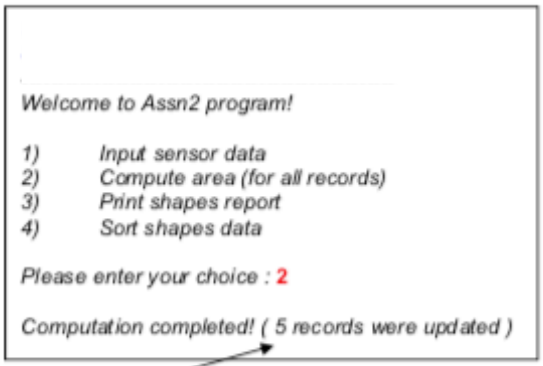
This class contains the **main ()** method which declares and instantiates all other classes (i.e. `Shape2D`, `Square`, ... etc) and sets up all the necessary interactions to perform its task.



```
Welcome to Assn2 program!

1)      Input sensor data
2)      Compute area (for all records)
3)      Print shapes report
4)      Sort shape data

Please enter your choice : 1

[ Input sensor data ]
Please enter name of shape : Cross
Please enter special type : WS

Please enter x-ordinate of pt.1 : 1
Please enter y-ordinate of pt.1 : 1

. . .

Please enter x-ordinate of pt.12 : 1
Please enter y-ordinate of pt.12 : 2

Record successfully stored. Going back to main
menu ...
```

The figure on the left describes a sample interaction between the main menu and '*Input sensor data*' sub-menu (1st example)

**Note :**
All shapes data should be stored in a `Shape2D` array in the `Assn2` driver class. You may assume that no more than 100 shapes will be entered into your program at any one time!



```
Welcome to Assn2 program!

1)      Input sensor data
2)      Compute area (for all records)
3)      Print shapes report
4)      Sort shapes data

Please enter your choice : 2

Computation completed! ( 5 records were updated )
```

The figure on the right describes a sample interaction between the main menu and '*Compute area (for all records)*' function.

**Note** : The example assumes the case where there are only 5 shapes input so far, hence, the message that '*5 records were updated*'!

In this compute area function, you must exhibit polymorphic behavior and dynamic binding by invoking the correct function for each shape stored in the `Shape2D` array !

```
Welcome to Assn2 program!

1)        Input sensor data
2)        Compute area (for all records)
3)        Print shapes report
4)        Sort shape data

Please enter your choice : 1

[ Input sensor data ]
Please enter name of shape : Circle
Please enter special type : NS

Please enter x-ordinate of center : 5
Please enter y-ordinate of center : 7
Please enter radius (units) : 2

Record successfully stored. Going back to main
menu …
```

The figure on the left describes a sample interaction between the main menu and '*Input sensor data*' sub-menu (2nd example)

Note : For circle, there is no need to enter vertices, as there are no "corners" as opposed to other multi-sided polygon shape. Instead, just prompt user to enter the [x, y] of its center and its radius, as shown on the left …

The figure on the right describes a sample interaction between the main menu and '*Print shapes report*' function.

**Notes** : The example assumes the case where there has only been 5 sensor data records input so far.

'Points on perimeter' refers to only those points lying on the line drawn between 2 vertices, for each pair of vertices defining the shape's outline.

E.g. (1, 2) lies on a line between vertices (1, 1) and (1, 3) !

You do not need to include those points which describe the vertices of the shape!

'Points within shape' refers to those points which are **totally within** the shape.

You do not need to include :

- those points which describe the vertices of the shape!
- those points on the perimeter of the shape!

```
Welcome to Assn2 program!

1)        Input sensor data
2)        Compute area (for all records)
3)        Print shapes report
4)        Sort shapes data

Please enter your choice : 3

Total no. of records available = 5

Shape [0]
Name   : Square
Special Type : WS
Area : 4 units square
Vertices :
Point [0] : (1, 1)
Point [1] : (1, 3)
Point [2] : (3, 3)
Point [3] : (3, 1)

Points on perimeter : (1,2), (2, 1), (2, 3), (3, 2)

Points within shape : (2, 2)

. . .

Shape [4]
Name   : Rectangle
Special Type : NS
Area : 8 units square
Vertices :
Point [0] : (2, 17)
Point [1] : (2, 15)
Point [2] : (6, 15)
Point [3] : (6, 17)

Points on perimeter : (2, 16), (3, 15), (4, 15), (5, 15), (6,
16), (5, 17), (4, 17), (3, 17)

Points within shape : (3, 16), (4, 16), (5, 16)
```

The figure on the right describes a sample interaction between the main menu and '*Sort shapes data*' sub-menu.

**Note** : For 'sort by special type and area' option in the sub-menu, shapes with special types 'WS' should be displayed first, followed by all shapes with special types 'NS'.

Within each group of shapes (i.e. WS or NS), display the shapes in descending order!

```
Welcome to Assn2 program!

1)        Input sensor data
2)        Compute area (for all records)
3)        Print shapes report
4)        Sort shapes data

Please enter your choice : 4

     a)        Sort by area (ascending)
     b)        Sort by area (descending)
     c)        Sort by special type and area

Please select sort option ('q' to go main menu) : b

Sort by area (largest to smallest) …

Shape [4]
Name   : Rectangle
Special Type : NS
Area : 8 units square
Vertices :
Point [0] : (2, 17)
Point [1] : (2, 15)
Point [2] : (6, 15)
Point [3] : (6, 17)

Points on perimeter : (2, 16), (3, 15), (4, 15), (5, 15), (6, 16), (5,
17), (4, 17), (3, 17)

Points within shape : (3, 16), (4, 16), (5, 16)

. . .

Shape [2]
Name   : Square
Special Type : WS
Area : 1 units square
Vertices :
Point [0] : (6, 6)
Point [1] : (6, 7)
Point [2] : (7, 7)
Point [3] : (7, 6)

Points on perimeter : none!

Points within shape : none!
```