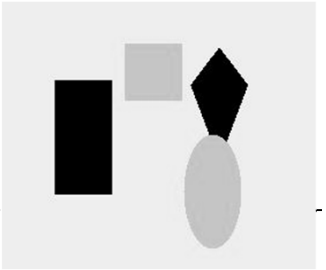
CompSci 230
Object Oriented Software Development

A1 Help

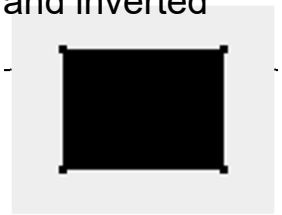
A1 – A3

- The aim is to give you experience with object-oriented programming, principles of inheritance and polymorphism.
- A3 is a bouncing program designed to let different shapes move around along various paths. Users will be able
  - to create shapes based on the classes you will write, and
  - to select individual existing shapes on the bouncing area and change their properties.

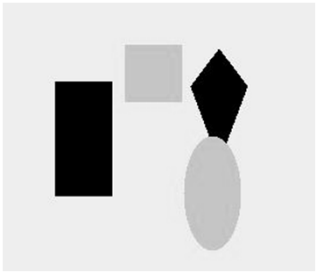
- A1 : Text based version of the program only
  - Simply prints the details of each shape to the console.
  - Complete ALL questions in CodeRunner



A1 - Java Classes



- Download A1code.zip
- `Shape` – abstract superclass
  - (x,y) top left corner, width, height, color, selected, inverted
  - `panelWidth`, `panelHeight` : bouncing area of a shape
  - path: `MovingPath` (inner local class) – defines a moving path
    - `BouncingPath`
    - `DiagonalPath`
- `A1` – The Application
- `AnimationViewer`
  - is the bouncing area
  - contains current values (e.g. current colour, current width, current height etc. for creating a new shape)



panelWidth x panelHeight

Stage 1: enum

- `PathType` – defines the moving path type
- `ShapeType` – defines the shape type

An enum is a special "class" that represents a group of constants

```
enum PathType { BOUNCE, ...
```

```
enum ShapeType { RECTANGLE,SQURAE,OVAL,...
```

- Steps:
  - Create two enums in A1.
    - The `ShapeType` enum represents the types of a shape.
    - The `PathType` enum represents the types of a path.
  - Modify the Shape class to use the above two enums.
  - Modify the `AnimationViewer` class to use the above two enums.

Stage 2: The Shape class

- To illustrate the inheritance hierarchy we use the class Shape as the super class and the classes Rectangle, Square, Oval and Kite inherit from Shape. Shape has the following instance variables:
  - `(x, y)` defines the top left corner of a shape,
  - `width` defines the width of a shape,
  - `height` defines the height of a shape,
  - `panelWidth` defines the width of a panel (i.e. bouncing area) of a shape,
  - `panelHeight` defines the height of a panel (i.e. bouncing area) of a shape,
  - `color` defines the fill color of a shape,
  - `path` defines the current moving path of a shape. If the moving path is BOUNCE, then the constructor should create a bouncing path with deltaX is 1 and deltaY is 2.
  - `selected` represents if the shape is selected or not. If a shape is selected, the program should draw the shape with 4 handles.
  - `inverted` represents if the shape is inverted in colour. If a shape is inverted, the fill colour is black.

The Shape class

- Complete the following:
  - 2 constructors
  - 3 abstract methods (`draw`, `contains` and `getArea`)
  - 1 concrete method (`toString`)
  - 1 concrete method (`move`)

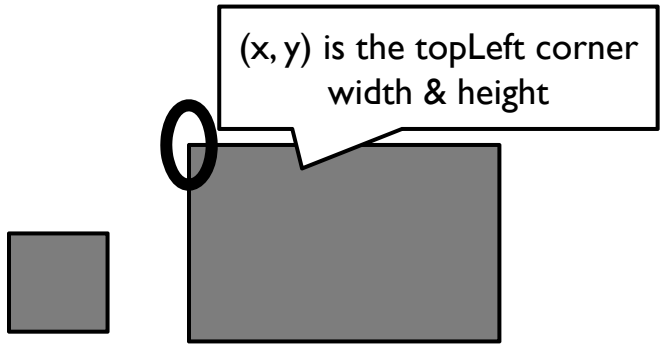
Stage 3: Subclasses

- You are required to define four subclasses. They are for rectangles, ovals, squares and kites. The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
  - When you define the `SquareShape` class, should you extends Shape or `RectangleShape`?
  - Should you implement the `draw`(), `contains`(), `getArea`() in each subclass?
  - Can you reuse the existing methods?

RectangleShape

Remember to add "import java.awt.Point; in you Java source file

- RectangleShape: create a new class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends Shape
  - Implement TWO Constructors
    - default
    - with 8 parameters
  - `draw`(): draws a rectangle shape
    - print the colour
    - use the toString() method to print the details
  - `contains`(): checks if the parameter point is within the rectangle
    - check 4 conditions
      - if the x-coordinate of the parameter point is less than … and … and … and
  - `get_area`(): returns the area of a rectangle

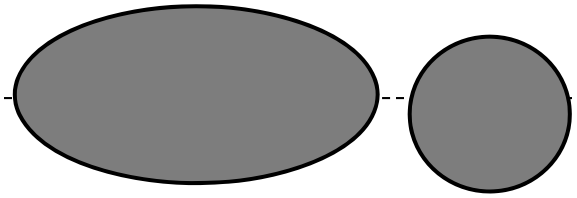


SquareShape

- SquareShape: create a new class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Should you extends `Shape` or `RectangleShape`?
    - Should you implement the `draw`()/`contains`()/`getArea`()?
    - Can you reuse the existing methods?
  - Implement Two constructors
    - default
    - with 7 parameters only

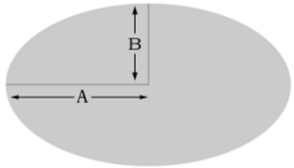
...

OvalShape



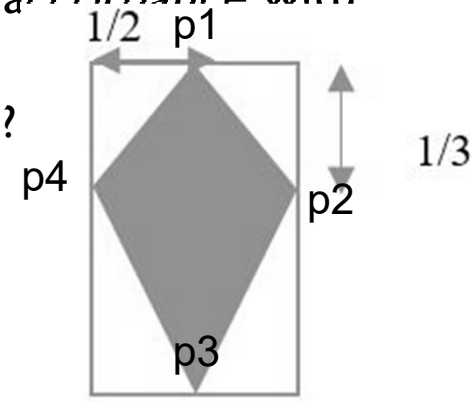
- OvalShape: create a new class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Should you extends `Shape` or `RectangleShape`?
    - Should you implement the `draw`()/`contains`()/`getArea`()?
  - Implement Two constructors
    - default
    - with 8 parameters
  - `draw`(): draws an oval shape
    - print the colour
    - use the `toString`() method to print the details
  - `contains`(): checks if the parameter point is within the oval/ellipse,
    - use the given formula
  - `getArea`(): returns the area of an oval/ellipse = Pi * A * B

```
dx = (2 * mx - x - x1) / w
dy = (2 * my - y - y1) / h
d =  dx * dx + dy * dy
```



KiteShape

- KiteShape: create a new class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Should you implement the **`draw`**()/**`contains`**()/**`getArea`**()?
  - Implement Two constructors
    - default
    - with 8 parameters
  - **`printCoordinates`**()
    - use 2 array of integers to create a **`Polygon`** object (four points to create a kite)
    - xarray/yarray contains the x-coordinate/y-coordinate of the four points
  - **`draw`**(): draws a kite shape
    - print the colour
    - use the **`toString`**() method to print the details
  - **`contains`**(): checks if the parameter point is within the polygon object
    - Use the contains() in the Polygon class
  - **`getArea`**(): returns the area of a kite (width * height) / 2



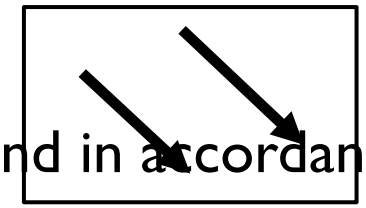
Create New Shape

- Complete the `createNewShape(int x, int y)` method
  - create a new shape based on the value of `currentShapeType`
    - `RECTANGLE` -> create a new rectangle and so on
  - Values:
    - `x` and `y` (parameters of the method) = top left corner of a shape.
    - `currentWidth` and `currentHeight`
    - `panelWidth` and `panelHeight`
    - `currentcolor`
    - `currentPathType`

from the AnimationViewer class

- Add two statements to update the `currentShapeType` and `currentPathType`
  - `RECTANGLE` -> `SQUARE` -> `OVAL` -> … etc
  - `BOUNCE` -> `DIAGONAL` (when you complete the last 2 questions)

Stage 4 - The DIAGONALPath



- Create a new **Inner** Class
  - The class hierarchy should be developed sensibly and in accordance with good object-oriented programming practice.
    - Extends the `MovingPath` class
    - inside the `Shape` class
  - Implement a constructor
  - Override the `move` method
    - If over the boundary, need to restart from x = 0 or y = 0

Stage 4 - The DIAGONALPath

- Modify the constructor in the `Shape` class
  - create a new path based on the value of `currentPathType`
    - **`BOUNCE`**:
      - set the current path to a new BouncingPath path
        - deltaX =1 and deltaY = 2
    - **`DIAGONAL`**
      - set the current path to a new DiagonalPath path
        - deltaX =2 and deltaY = 2