Geometry + JSON

A JSON file with the following structure is given as input:

```
{
        “geometry” : {
                “shape” : {
                        “id” : “shape1”,
                        “point” : [
                                {
                                        “x” : 0,
                                        “y” : 0
                                },
                                {
                                        “x” : 10,
                                        “y” : 0
                                },
                                {
                                        “x” : 10,
                                        “y” : 10
                                },
                                {
                                        “x” : 0,
                                        “y” : 10
                                }
                        ]
                }
        }
}
```

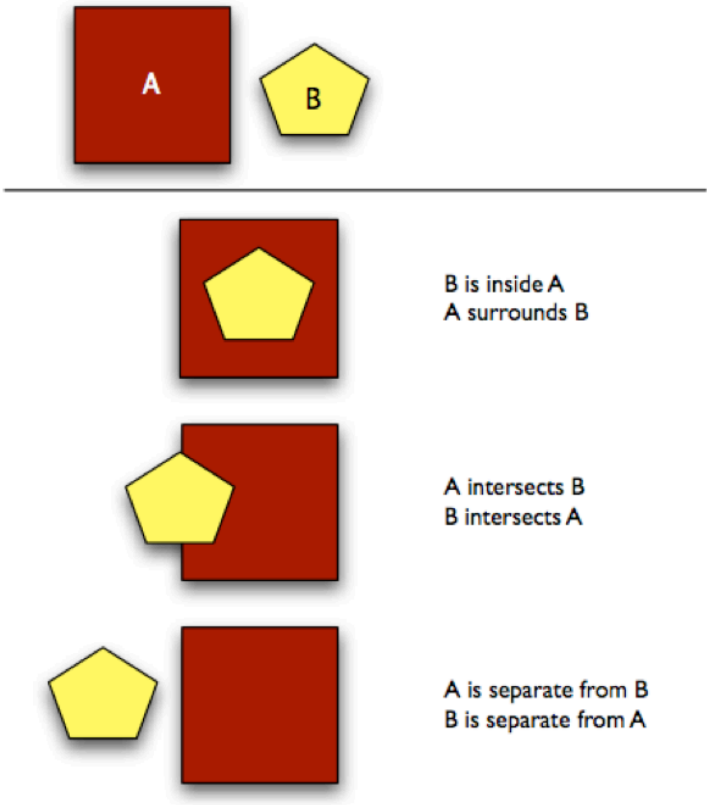
Notes About the JSON

The `point` nodes represent points on a two dimensional coordinate plane. These nodes are also ordinal in that their order defines how the shape is logically drawn. In other words, the first node is connected by a line to the second node, the second node is connected to the third and so on. The last node is implicitly connected to the first node. The sample above defines a simple square. You can also assume:

- ⇒ The JSON will be well-formed
- ⇒ The `id` attribute of the `shape` node will be a unique string
- ⇒ There will be at least three `point` children of the `shape` node
- ⇒ Each `point` child will always have an `x` and `y` attribute that will be an integer
- ⇒ There is no need to account for chords (curved lines)
- ⇒ There will be at least one `shape` child of the `geometry` node, but there is an unlimited maximum

The Program

The program must be written in Ruby with a command line interface that will accept the JSON file as its only argument. Upon ingesting the JSON file, the program must validate that the shape defined by each of the `point`s in each `shape` node represents a polygon. In order to be valid, the shape has to be a strictly convex polygon (http://en.wikipedia.org/wiki/Convex_polygon). If an invalid shape is detected, a non-fatal error should be displayed to the user describing the problem (please refer to the *example program execution* below). Shapes should be referred to by their `id` attribute. After examining the shape nodes, if there are two or more valid polygons available, the program must compare the polygons to each other and describe their relationship in one of three ways (“A” and “B” are the id’s of shapes):



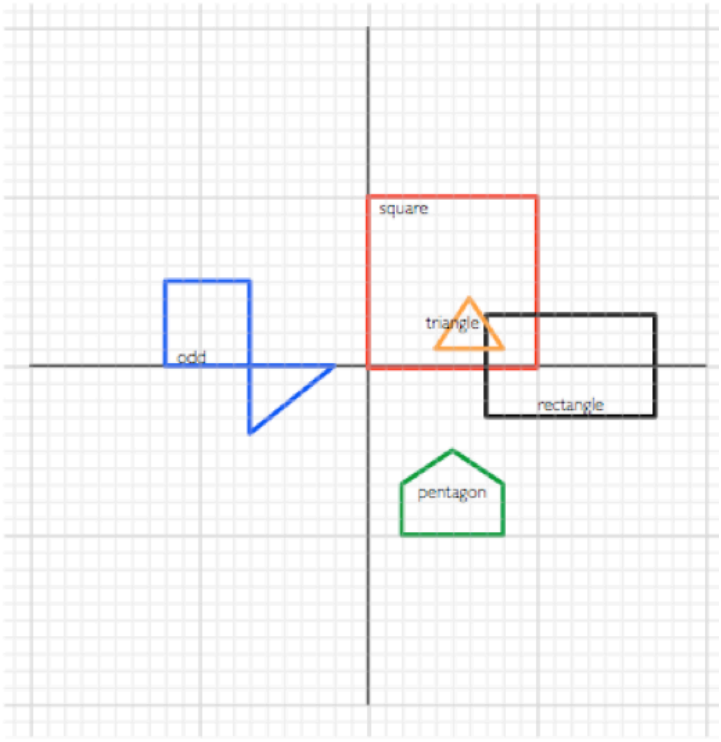
Design

The program should make use of good object-oriented design. Although the concept is simple, one should consider possible extensions to the architecture. For example:

- ⇒ What other kinds of things might one want to do with shapes besides relate them to each other?
- ⇒ What kinds of operations might be performed on shapes?
- ⇒ Are there different ways that one may want to specify shapes? The goal is an elegant solution. Remember that elegance is the combination of power and simplicity. You are free to use any built-in functions. Otherwise, please use original code.

Example Input

```
{
	“geometry” : {
		“shape” : [ {
			“id” : “square”,
			“point” : [
				{
					“x” : 0,
					“y” : 0
				},
				{
					“x” : 10,
					“y” : 0
				},
				{
					“x” : 10,
					“y” : 10
				},
				{
					“x” : 0,
					“y” : 10
				}
			]
		},
		{
			“id” : “odd”,
			“point” : [
				{
					“x” : -2,
					“y” : 0
				},
				{
					“x” : -12,
					“y” : 0
				},
				{
					“x” : -12,
					“y” : 5
				},
				{
					“x” : -7,
					“y” : 5
				},
				{
					“x” : -7,
					“y” : -4
				}
			]
		},
		{
			“id” : “triangle”,
			“point” : [
				{
					“x” : 4,
					“y” : 1
				},
				{
					“x” : 8,
					“y” : 1
				},
				{
					“x” : 6,
					“y” : 4
				}
			]
		},
		{
			“id” : “rectangle”,
			“point” : [
				{
					“x” : 7,
					“y” : 3
				},
				{
					“x” : 17,
					“y” : 3
				},
				{
					“x” : 17,
					“y” : -3
				},
				{
					“x” : 7,
					“y” : -3
				}
			]
		},
		{
			“id” : “pentagon”,
			“point” : [
				{
					“x” : 2,
					“y” : -7
				},
				{
					“x” : 2,
					“y” : -10
				},
				{
					“x” : 8,
					“y” : -10
				},
				{
					“x” : 8,
					“y” : -7
				},
				{
					“x” : 5,
					“y” : -5
				}
			]
		} ]
	}
}
```



Example Program Execution

“odd” is not a polygon
“square” surrounds “triangle”
“square” intersects “rectangle”
“square” is separate from “pentagon”
“triangle” is inside “square”
“triangle” intersects “rectangle”
“triangle” is separate from “pentagon”
“rectangle” intersects “square”
“rectangle” intersects “triangle”
“rectangle” is separate from “pentagon”
“pentagon” is separate from “square”
“pentagon” is separate from “triangle”
“pentagon” is separate from “rectangle”