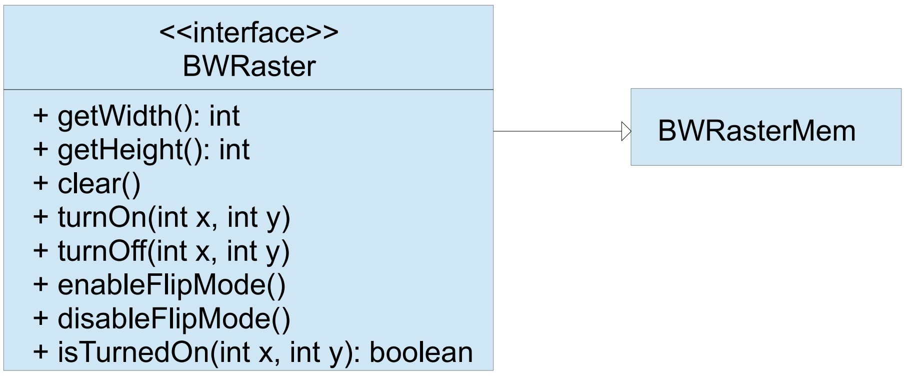
2. homework assignment; JAVA, Academic year 2012/2013; FER

A usual, please see the end of the last page. I mean it! You are back? OK. This homework consists of one problem which is explained in this document and another problem which is described in document `hw02b.pdf` (in Croatian).

Problem 1.

You will implement a hierarchy of various geometric shapes and provide a procedure for its drawing on raster devices. So will start with the latter, as illustrated on following image.



We will model a raster device using interface `BWRaster`. Letters `BW` stand for Black-and-White raster. This is an abstraction for all raster devices of fixed width and height for which each pixel can be painted with only two colors: black (when pixel is turned off) and white (when pixel is turned on). Methods `getWidth` and `getHeight` should return appropriate dimensions of used raster. Method `clear` should turn off all pixels in raster. Method `turnOff` should turn off pixel at the specified location, or if the location is invalid throw an appropriate exception (use some appropriate exception available in JRE).

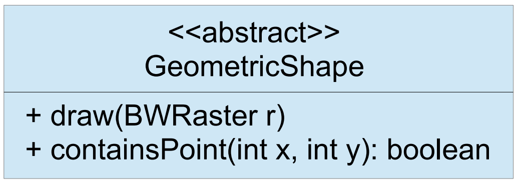
The working of `turnOn` method is closely controlled with flipping mode of raster. If flipping mode of raster is disabled, then the call of the `turnOn` method turns on the pixel at specified location (again, if location is valid). However, if flipping mode is enabled, then the call of the `turnOn` method flips the pixel at the specified location (if it was turned on, it must be turned off, and if it was turned off, it must be turned on).

Methods `enableFlipMode` and `disableFlipMode` control the flipping mode which is initially disabled. Method `isTurnedOn` checks if the pixel at the given location is turned on and if it is, returns `true`, otherwise returns `false`. If location is invalid, it should throw an appropriate exception.

The class `BWRasterMem` is an implementation of this interface which keeps all of its data in computer memory. You should use arrays of appropriate type for this. Upon creation of new objects of this class it is expected that all pixels will be turned off.

You should put previous interface and class in package `hr.fer.zemris.java.graphics.raster`.

Each geometric shape should be part of the inhertance tree rooted at the abstract class `GeometricShape` which is specified on the next page.



Method `containsPoint` checks if given point belongs to specified geometric shape. The general contract for this method is that it must return `false` only if the location is outside of the geometric shape. For all other cases it must return `true`. Additionally, since for the class `GeometricShape` we have no idea what shape this actually is, think whether and how to implement it (or what to do with it).

However, assuming that concrete classes that inherit from this class know how to do this, you should provide in `GeometricShape` an implementation of method `draw` that will be able to draw any possible geometric shape (efficiency is at this point of no concern). Please note that this method must draw filled image of the geometric shape, not only its outline.

You should provide an implementations of the following geometric shapes (all coordinates and parameters must be integers).

- `Rectangle` (which must be specified by its `x`,`y` coordinates of its top-left corner and by its `width` and `height`); rectangles of width or height equal to zero are not allowed (and they would be invisible).
- `Square` (which must be specified by its `x`,`y` coordinates of its top-left corner and by its `size`); squares of size equal to zero are not allowed (and they would be invisible).
- `Ellipse` (which must be specified by its center and horizontal and vertical radius); ellipses with any radius smaller than 1 are not allowed.
- `Circle` (which must be specified by its center and radius); circles with radius smaller than 1 are not allowed. Circle with radius 1 would be rendered as a single turned-on pixel in circle's center. Circle with radius 2 would be rendered to with one additional pixel to the right and left (and top and bottom).
- `Triangle` (which must be specified by `x`,`y` coordinates of the three points that represents its vertexes; for example, a triangle could be specified by points (1,1), (20,4), (15,17)).

Once created, each of those geometric shapes should provide appropriate getters and setters for reading and updating of parameters initially given in constructor. Some of the parameters does not have to be positive numbers. For example, it is OK to have a rectangle whose top left corner is at (-2, -1).

Please note that during the design of inheritance tree you must try to satisfy following:
- minimize code duplication by pushing shared code toward the top of the inheritance tree (but not too high!)
- consider good OO practices, especially the Liskov Substitution Principle[1] which we commented during the previous class; in order to solve this correctly, you can consider the list of geometric shapes I prescribed here as a lower limit and you are free to supplement it as you see fit.

In each derived geometric shape do not override method `GeometricShape` at this point.

All of the previously described geometric shapes should be placed in package `hr.fer.zemris.java.graphics.shapes`.

Create a package `hr.fer.zemris.java.graphics.views`. Place inside an interface `RasterView`. It should provide a single method:

1 http://en.wikipedia.org/wiki/Liskov_substitution_principle

```
void produce(BWRaster raster);
```

Classes which implement this interface will be responsible for visualization of created images.

You should create class `SimpleRasterView` which is an implementation of this interface and which outputs the textual representation of image to standard output. Place this class into the same package. Class `SimpleRasterView` should have two constructors. One constructor should allow user to specify which character will be used to represent pixels that are turned on and which character will be used to represent pixels that are turned off. The other constructor should be the default constructor that delegates the call to first constructor and provides '*' and '.' as required characters. Please see link[2] for more details on how to do that.

When user calls method produce on `SimpleRasterView` object, the image should be displayed and no extra newlines should be emmited. For example:

```
Rectangle rect1 = new Rectangle(0, 0, 4, 4);
Rectangle rect2 = new Rectangle(1, 1, 2, 2);

BWRaster raster = new BWRasterMem(6, 5);
raster.enableFlipMode();

rect1.draw(raster);
rect2.draw(raster);

RasterView view = new SimpleRasterView();
view.produce(raster);
view.produce(raster);

System.out.println();

RasterView view2 = new SimpleRasterView('X','_');
view2.produce(raster);
```

should result with following being written to standard output:

```
****..
*..*..
*..*..
****..
......
****..
*..*..
*..*..
****..
......

XXXX__
X__X__
X__X__
XXXX__
______
```

2 http://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

If you got this, now you should override `draw` methods for all geometric shapes where it makes sense to do so in order to achieve much more efficient rendering of geometric shapes. However, please note that this does not mean to override it in each geometric shape since it would be most likely a lot of code duplication. Think carefully where you can provide a better implementation and leave the polymorphic behavior for inherited classes do its work. Also, since we are working with raster devices and our geometric shapes are not rasters in nature, do not worry too much about shape boundaries (it some pixel included in it or not; implement it as you see fit and during the homework review we won't count each pixel). During the development of the draw method you may not expect that all parts of the shape will be visible – it is OK to have a circle whose center is placed at (0,0); a call to `draw` method should in end result with only one quarter of circle being shown.

Now create a program `Demo` and place it in package `hr.fer.zemris.java.graphics`. Program expects user to provide either a single argument or two arguments. In case that user provides a single argument, its value is interpreted as width and height of raster. In case that user provides two arguments, first is treated as width of raster and second as height of raster. In case there are zero arguments or more than two arguments program should write appropriate message end terminate. In case arguments can not be interpreted as numbers (or are inappropriate), again write a message and terminate program.

User will tell you what he wants to create by typing, one shape per line. First line will contain a number of shapes that follow. Here is an example:

```
java -cp bin  hr.fer.zemris.java.graphics.Demo 40 30
7
FLIP
RECTANGLE 0 0 10 20                // x y w h
SQUARE 0 0 10                      // x y size
FLIP
ELLIPSE 10 10 5 10                 // cx cy radiusX radiusY
CIRCLE 10 10 5                     // cx cy radius
TRIANGLE 1 1 10 10 8 20            // t0x t0y t1x t1y t2x t2y
```

In the above example, if user provided more than seven additional lines, you should discard all but first seven (not counting the first line with the number 7). Your program should create an array with references to instances of specified objects. When you see "FLIP", add `null` reference. In each line arguments are separated by one or more spaces (ascii 32).

When processing of input is done, you have an array with references to your shapes, some of which can be `null`. You should start drawing your shapes, one by one. Each time you encounter `null` reference in array, change flipping mode of raster object.

Once all shapes are rendered, you should write final image to standard output using '*' and '.' for pixel representation.

If you encounter any problem during shape creation (user entered invalid shape name, wrong number of arguments, etc. write appropriate message and terminate program). User should be allowed to create only the prescribed five types of shapes.

Your program should not dump any stack traces to user!

Additional notes: (0,0) is for raster top-left corner (same as computer screen); x-coordinated grow from left to right, y-coordinates from top to bottom. You must take precaution when implementing object drawing not to try to turn-on nonexistent pixels (for example, for raster of width 10 to try to turn on pixel on x>=10).

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries for this homework (whether it is yours old code or someones else). Document your code!

Important notes

Solve all of the problems described in this document in a single Eclipse project named `homework02a`. Solve all of the problems described in the document `hw02b.pdf` in a single Eclipse project named `homework02b`. In each project configure Eclipse to use two source directories: `src/main/java` for your source files and `src/test/java` for sources files of unit tests. You must document your implementation and you must write appropriate number of tests. Just as in previous homework, you must equip your projects with `build.xml` file so that everything can be done from command line.

Once you are done, create a single ZIP archive containing both projects and upload this archive on Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted.