4. homework assignment; JAVA, Academic year 2015/2016; FER

Problem 1.

In the book I have written about some implementation details of class `String` in Java (in book version 2015-09-30, starting from page 82). After this text was written, the implementation of class String has changed (the internal character array are no longer shared and operations such substring are no longer performed in O(1) but in O(n)).

The implementation of the class `String` as described in the book allowed performing operations such as `substring` in constant time (independent on the substring length). The general idea of the implementation was for multiple instances of strings to share a single character array and to remember which part of the array belongs to each instance. Since `String` instances didn't allow user to change current data (instead, such methods created new objects), this implementation was safe. Starting with Java 7 update 6, the internal implementation of `String` has changed; Strings are still unmodifiable, but substring operation now creates new objects with it own copy of character array which only contains characters belonging to new `String`.

Your job is to create a class `CString` (put it in package `hr.fer.zemris.java.cstr`) which offers similar functionality as the old official implementation of the `String` class: it should represent unmodifiable strings on which substring methods (and similar) must be executed in O(1) complexity, which you can achieve by sharing the character array. Here is the official list of methods your `CString` must support:

- Constructors:
  - `CString(char[] data, int offset, int length);`
  - `CString(char[] data);`
  - `CString(CString original);` if originals internal character array is larger than needed, your new instance must allocate its own character array of minimal required size and copy data; otherwise it must reuse original's character array
- Static methods:
  - `Cstring fromString(String s);` returns new CString object which has the same character data as given Java's `String` object.
- Instance methods:
  - `length(): int;`
  - `charAt(int index): char;`
  - `toCharArray(): char[];` allocates a new array of length equals to length of this CString object (not its internal array which may be larger!), copies string content into it and returns it
  - `toString(): String;` used for conversion of CString into String
  - `indexOf(char c): int;` returns index of first occurrence of char or -1
  - `startsWith(CString s): boolean;` returns `true` if this string begins with given string, `false` otherwise
  - `endsWith(CString s): boolean;` returns `true` if this string ends with given string, `false` otherwise
  - `contains(CString s): boolean;` returns `true` if this string contains given string at any position, `false` otherwise
  - `substring(int startIndex, int endIndex): CString;` returns new `CString` which represents a part of original string; position `endIndex` does not belong to the substring; it must hold: `startIndex>=0`, `endIndex>=startIndex;` its complexity must be O(1)
  - `left(int n): CString;` returns new `CString` which represents starting part of original string and is of length `n`; throw an exception if this can not be constructed; `n>=0`

- `right(int n): CString`; returns new `CString` which represents ending part of original string and is of length `n`; throw an exception if this can not be constructed; `n>=0`
- `add(CString s): CString`; creates a new `CString` which is concatenation of current and given string
- `replaceAll(char oldChar, char newChar): CString`; creates a new `CString` in which each occurrence of old character is replaces with new character
- `replaceAll(CString oldStr, CString newStr): CString`; creates a new `CString` in which each occurrence of old substring is replaces with the new substring

Testing hint: do not forget to check what will be the result of:

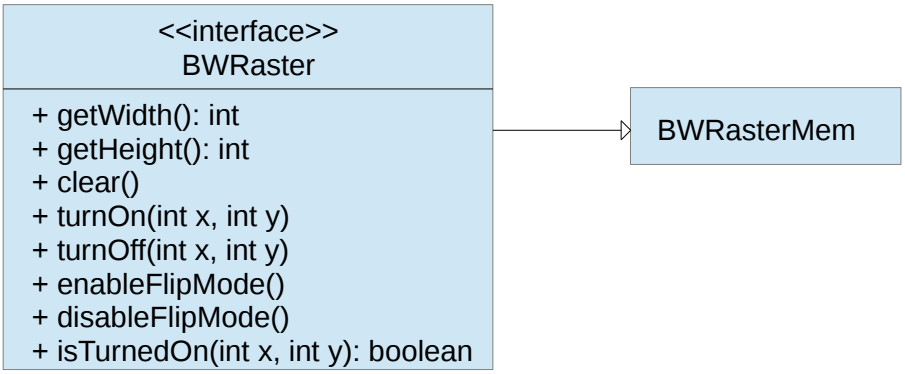
```
new CString("abababab").replaceAll(new CString("ab"), new Cstring("abab"))
```

Before you start writing the solution, using Eclipse wizard create a new (additional) source folder tests/cstring, create in it package `hr.fer.zemris.java.cstr` and put in that package class CStringTests. In this class you are required to write appropriate number of unit tests for methods and constructors of CString class. Use this unit tests as help while developing CString. In order to do so, you will also need to add a JUnit library into your project (as described in previous homework).

You do not have to javadoc junit tests. Each test should be named appropriately (names **can be** long and descriptive). If some of the tests are convoluted and hard to understand, write a simple comment (not javadoc-comment) to explain it.

Problem 2.

You will implement a hierarchy of various geometric shapes and provide a procedure for its drawing on raster devices. We will start with the latter, as illustrated in the 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. Methods `turnOn`/`turnOff` should turn pixel on/turn pixel off, at the specified location. Methods `turnOn`, `turnOff` and `isTurnedOn` should throw `IllegalArgumentException` if (x,y) is invalid with respect to raster dimensions. The coordinate system for raster has (0,0) at the top-left corner of raster; positive x-axis is to the right and positive y-axis is toward the bottom.

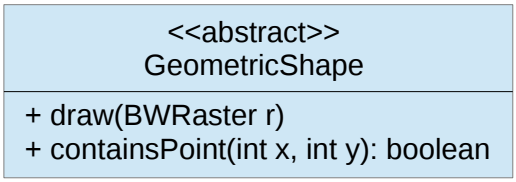
The working of `turnOn` method is closely controlled with flipping mode of raster. If flipping mode of raster is disabled, then the call of the `turnOn` method turns on the pixel at specified location (again, if location is valid). However, if flipping mode is enabled, then the call of the `turnOn` method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

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. On creation of new objects of this class it is expected that all pixels will be initially turned off. This class must provide a single public constructor which accepts raster width and raster height; both must be at least 1.

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

Each geometric shape should be part of the inheritance tree rooted at the abstract class `GeometricShape`.



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).

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). However, please make sure that visualization process of such geometric shapes renders only the part of the shape which is inside the given raster (no exceptions must be thrown).

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 (e.g. add intermediate classes, etc.).

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

At this point, none of the previously defined concrete geometric shapes should have `draw` method overriden. Leave it that way for now.

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

```
Object 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

1 http://en.wikipedia.org/wiki/Liskov_substitution_principle; also read on this in book!

the textual representation of image to standard output and returns `null` as result. You should also create class `StringRasterView` which is another implementation of this interface and which returns a single String which contains textual representation of image (pixel rows should be delimited by '\n'). Place these classes into the same package. Both classes 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__
______
```

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

2 http://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html and read the appropriate topic in book!

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). Additionally, for this homework you can not use any of Java Collection Framework classes or its derivatives. Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. You must name your project's main directory (which is usually also the project name) `HW04-`*yourJMBAG*; for example, if your JMBAG is 0012345678, the project name and the directory name must be `HW04-0012345678`. Once you are done, export the project as a ZIP archive and upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is April 7th 2016. at 08:00 AM.