10. homework assignment; JAVA, Academic year 2013/2014; FER

Tekst ove zadaće dan je na hrvatskom (osim zadatka 2). Postoje dva roka za upload različitih dijelova: obavezno pročitajte uputu na kraju I pogledajte što do kada treba riješiti.

Zadatak 1.

Vratite se na Vašu implementaciju algoritma *QuickSort* iz prethodne domaće zadaće. Prekopirajte razred `QSortParallel` u razred `QSortParallel2` i modificirajte implementaciju tako da paralelizaciju obavlja koristeći ForkJoin framework dostupan u Javi (znači, u okviru zadatka 1 više nigdje ne smijete eksplicitno stvarati primjerke razreda `Thread` ili pak koristiti usluge `ExecutorService`).

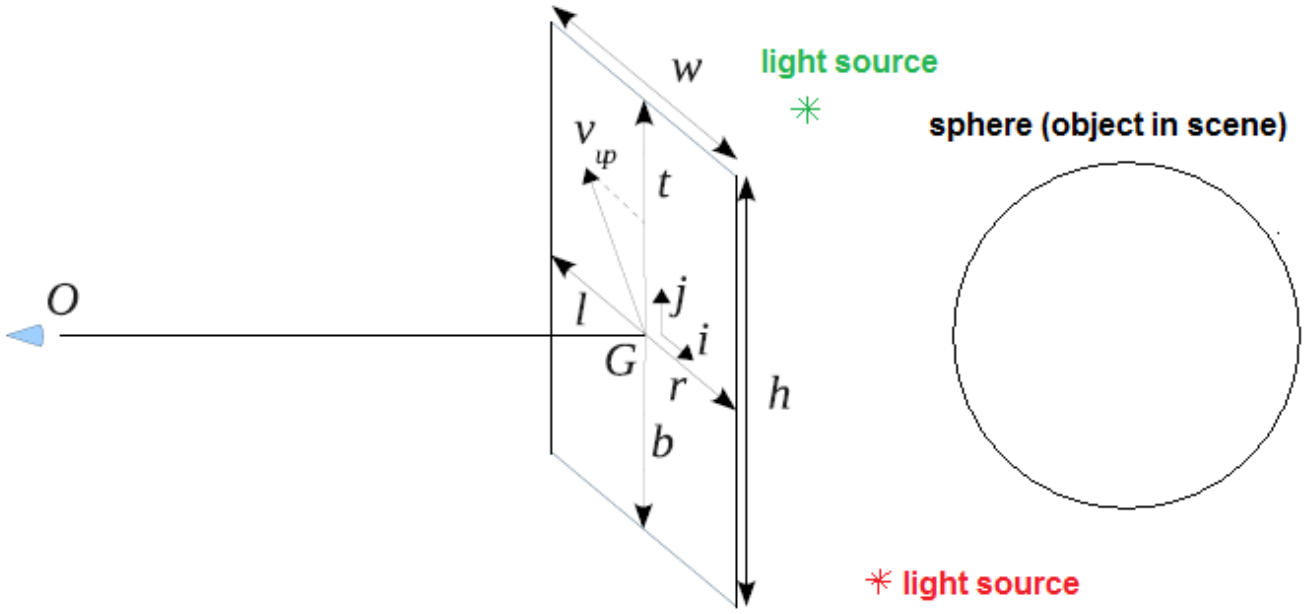
Zadatak 2.

You will write a simplification of ray-tracer for rendering of 3D scenes; don't worry – it's easy and fun. And also, we won't write a full-blown ray-tracer but only a ray-caster.

I have already prepared a lot of code for you: please download `hw06-ray01.jar` and `hw06-ray02.jar` which are available on Ferko. To better understand this, you are also advised to download the last version of the book available at:

http://java.zemris.fer.hr/nastava/irg/

(it is knjiga-0.1.2014-02-07.pdf) and read section 9.2 (Phong model, pages 225 to 228) and section 10.2 (Ray-casting algorithm, pages 235 to 238). To render an image using ray-casting algorithm, you start by defining which object are present in 3D scene, where are you stationed (eye-position: O), where do you look at (view position: G) and in which direction is “up” (view-up approximation). See next image.



Now imagine that you have constructed a plane perpendicular to vector that connects eye position (O) and

view point (G). In that plane you will create a 2D coordinate system, so you will have x-axis (as indicated by vector *i* on image) and y-axis (as indicated by vector *j* on image). If you only start with eye-position and view point, your y-axis can be arbitrarily placed in this plane. To help us fix y-axis, it is customary to specify another vector: view-up vector that does not have to lay in plane but it must not be co-linear with G-O vector. If this holds, that exists projection of view-up vector into plane: normalized version of this projection will become our vector *j* and hence determine the orientation of y-axis.

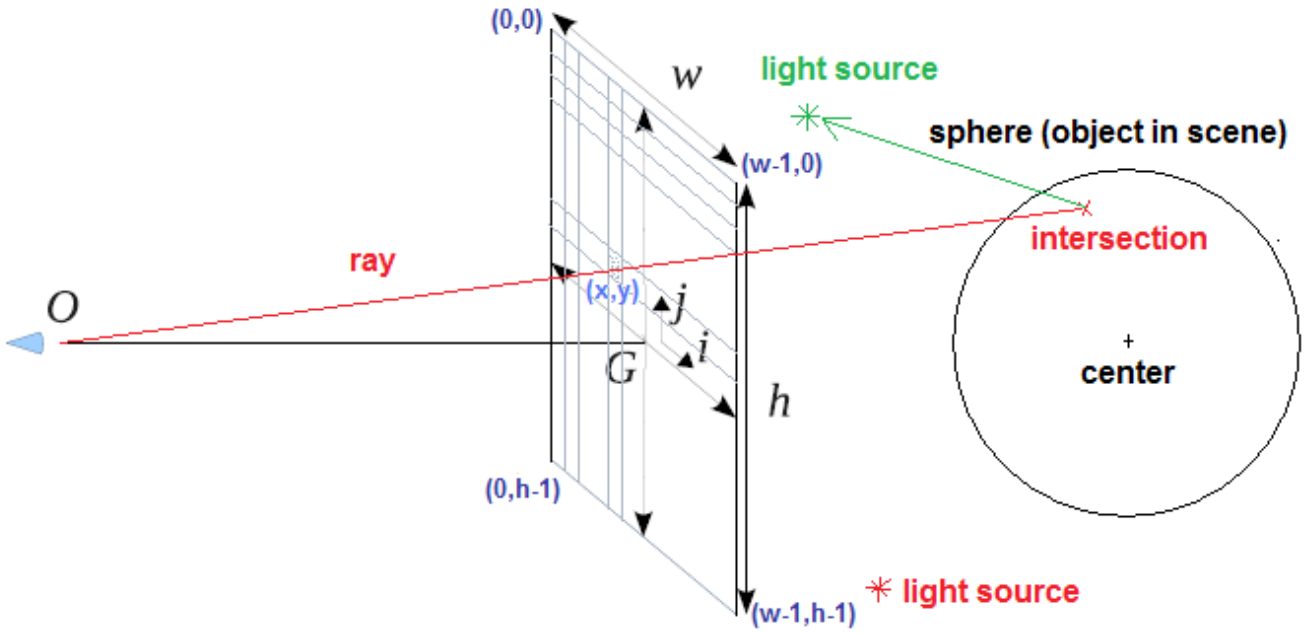
Lets start calculating. Let: $\vec{OG} = \frac{\vec{G}-\vec{O}}{\|\vec{G}-\vec{O}\|}$, i.e. it is normalized vector from $\vec{O}$ to $\vec{G}$; let $V\vec{U}V$ be normalized version of view-up vector. Then we can obtain vector $\vec{j}'$ as follows:

$\vec{j}' = V\vec{U}V - \vec{OG}(\vec{OG} \cdot V\vec{U}V)$ where $\vec{OG} \cdot V\vec{U}V$ is scalar product. Define its normalized version to be: $\vec{j} = \frac{\vec{j}'}{\|\vec{j}'\|}$. Now we can calculate vector $\vec{i}'$ that will determine orientation of x-axis as cross product: $\vec{i}' = \vec{OG} \times \vec{j}$ and its normalized version $\vec{i} = \frac{\vec{i}'}{\|\vec{i}'\|}$.

Once we determined where the plane is and what are the vectors determining our x-asis (i.e. $\vec{i}$) and y-axis (i.e. $\vec{j}$), we have to decide which part of this plane will be mapped to our screen. We will asume it to be rectangle going left from $\vec{G}$ (i.e. in direction $-\vec{i}$) for *l*, going right from $\vec{G}$ (i.e. in direction $\vec{i}$) for *r*, going up from $\vec{G}$ (i.e. in direction $\vec{j}$) for *t*, and finally going down from $\vec{G}$ (i.e. in direction $-\vec{j}$) for *b*. To simplify things further, lets assume that $l = r = \frac{horizontal}{2}$ and $t = b = \frac{vertical}{2}$ where we introduced two parameters: *horizontal* and *vertical*.

In archive `hw06-ray01.jar` I have already prepared the class `Point3D` with implemented methods for calculation of scalar products, cross-products, vector normalization etc. so use it.

Now we will define final screen coordinate system, as shown on next picture.



We will define (0,0) to be upper left point of our rectangular part of plane; x-axis will be oriented just as

vector $\vec{i}$ is, and y-axis will be oriented opposite from vector $\vec{j}$. We can obtain 3D coordinates of our upper-left corner as follows:

$$\vec{corner} = \vec{G} - \frac{horizontal}{2} \cdot \vec{i} + \frac{vertical}{2} \cdot \vec{j}$$

Now for each x from 0 to w-1 and for each y from 0 to h-1 we can calculate the position of pixel (x,y) in our plane as follows:

$$\vec{point}_{xy} = \vec{corner} + \frac{x}{w-1} \cdot horizontal \cdot \vec{i} - \frac{y}{h-1} \cdot vertical \cdot \vec{j}$$

And now its simple: we define a ray of light which starts at $\vec{O}$ and passes through $\vec{point}_{xy}$. We calculate if this ray which is specified by starting point $\vec{O}$ and normalized directional vector $\vec{d} = \frac{\vec{point}_{xy} - \vec{O}}{\| \vec{point}_{xy} - \vec{O} \|}$ has any intersections with objects in scene! If an intersection is found, then that is exactly what will be visible for our pixel (x,y). If no intersection is found, pixel will be rendered black (r=g=b=0). However, if an intersection is found, we must determine what color to use. If multiple intersections are found, we must chose the closest one to eye-position since that is what we will see. For coloring we will use Phongs model which assumes that there is one or more light sources present in scene. In our example there are two light sources. Each light source is specified with intensities of r, g and b components it radiates.

Here is the pseudo code:

```
for each pixel (x,y)
  calculate ray r from eye-position to pixel_xy
  determine closest intersection S of ray r and any object in scene (in front of observer)
  if no S exists, color (x,y) with rgb(0,0,0) else use rbg(determineColorFor(S))
```

The procedure `determineColorFor(S)` is given by following pseudocode:

```
set color = rgb(15,15,15) // i.e. ambient component
for each light source ls
  define ray r' from ls.position to S
  find closest intersection S' of r' and any objects in scene
  if S' exists and if closer to ls.position than S, skip this light source (it is obscured by that object!)
  else color += diffuse component + reflective component
```

Details

Go through sources of `IrayTracerProducer`, `IrayTracerResultObserver`, `GraphicalObject`, `LightSource`, `Scene`, `Point3D`, `Ray` and `RayIntersection`. Create package `hr.fer.zemris.java.tecaj_06.rays` in your homework and add class `Sphere`:

```
package hr.fer.zemris.java.tecaj_06.rays;

public class Sphere extends GraphicalObject {

	...

	public Sphere(Point3D center, double radius, double kdr, double kdg,
		double kdb, double krr, double krg, double krb, double krn) {
		...
```

```
	}

	public RayIntersection findClosestRayIntersection(Ray ray) {
		...
	}
}
```

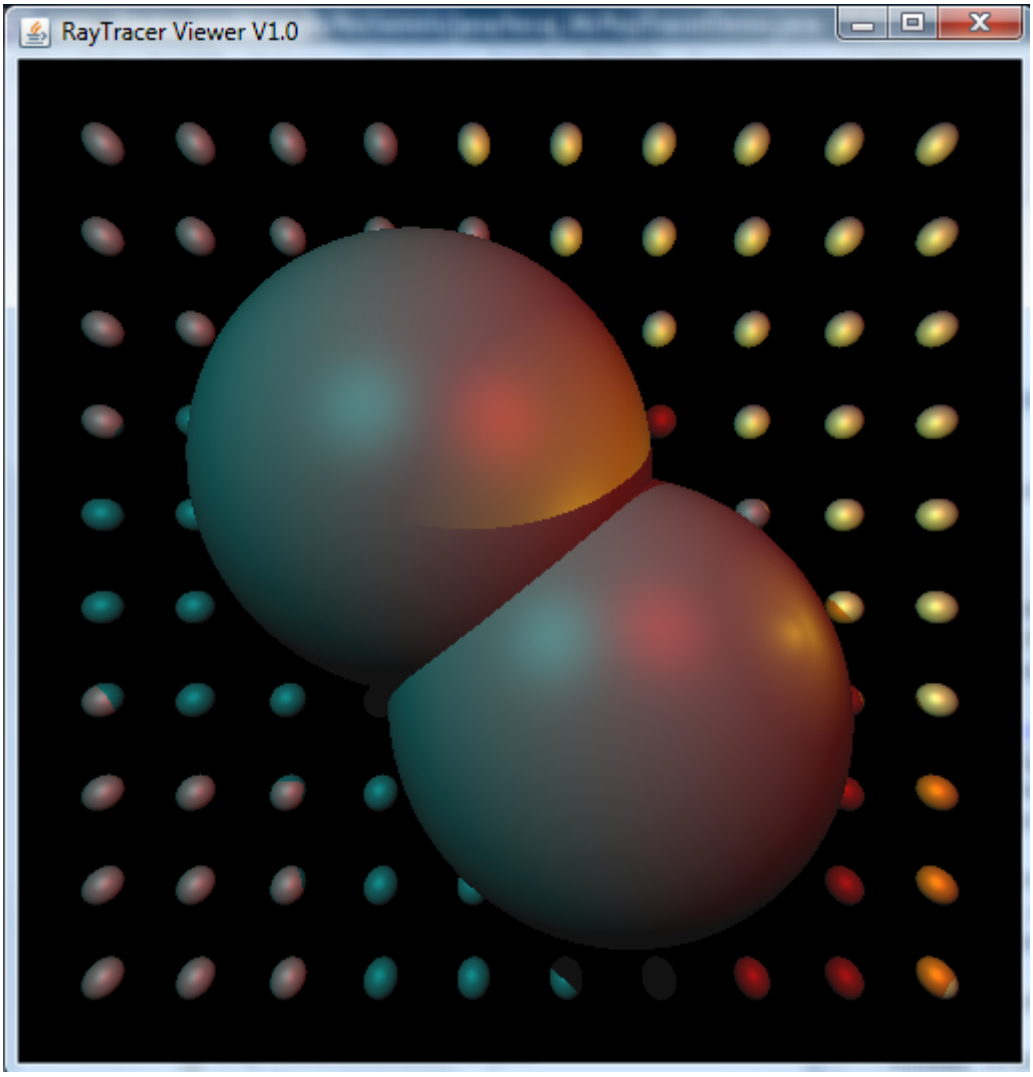
and implement all that is missing. Until you do that, Eclipse will report errors that `RayTracerViewer` references a class `Sphere` that does not exists. Coeficients *kd** determine object parameters for diffuse component and *kr** for reflective components.

Write a main program `hr.fer.zemris.java.hw06.part2.RayCaster`. The basic structure of it should look like this:

```
public static void main(String[] args) {
	RayTracerViewer.show(getIRayTracerProducer(),
	new Point3D(10,0,0),
	new Point3D(0,0,0),
	new Point3D(0,0,10),
	20, 20);
}

private static IRayTracerProducer getIRayTracerProducer() {
	return new IRayTracerProducer() {

		@Override
		public void produce(Point3D eye, Point3D view, Point3D viewUp,
				double horizontal, double vertical, int width, int height,
				long requestNo, IRayTracerResultObserver observer) {

			System.out.println("Započinjem izračune...");
			short[] red = new short[width*height];
			short[] green = new short[width*height];
			short[] blue = new short[width*height];

			Point3D zAxis = ...
			Point3D yAxis = ...
			Point3D xAxis = ...

			Point3D screenCorner = ...

			Scene scene = RayTracerViewer.createPredefinedScene();

			short[] rgb = new short[3];
			int offset = 0;
			for(int y = 0; y < height; y++) {
				for(int x = 0; x < width; x++) {
					Point3D screenPoint = ...
					Ray ray = Ray.fromPoints(eye, screenPoint);

					tracer(scene, ray, rgb);

					red[offset] = rgb[0] > 255 ? 255 : rgb[0];
					green[offset] = rgb[1] > 255 ? 255 : rgb[1];
					blue[offset] = rgb[2] > 255 ? 255 : rgb[2];

					offset++;
				}
			}
```

```
                    System.out.println("Izračuni gotovi...");
                    observer.acceptResult(red, green, blue, requestNo);
                    System.out.println("Dojava gotova...");
                }
        };
}
```

Fill the missing parts! If you do this OK, you will get the following image.



Now if this goes OK, please observe that calculation of color for each pixel is independent from other pixels. Using this knowledge write a main program `hr.fer.zemris.java.hw06.part2.RayCasterParallel` which parallelizes the calculation using Fork-Join framework and `RecursiveAction`.

Zadatak 3.

Proučiti:

http://docs.oracle.com/javase/tutorial/uiswing/layout/custom.html

Sve komponente razvijete u okviru ovog zadatka stavite u paket `hr.fer.zemris.java.gui.layouts`. Napravite vlastiti *layout manager* naziva `CalcLayout`, koji će implementirati sučelje

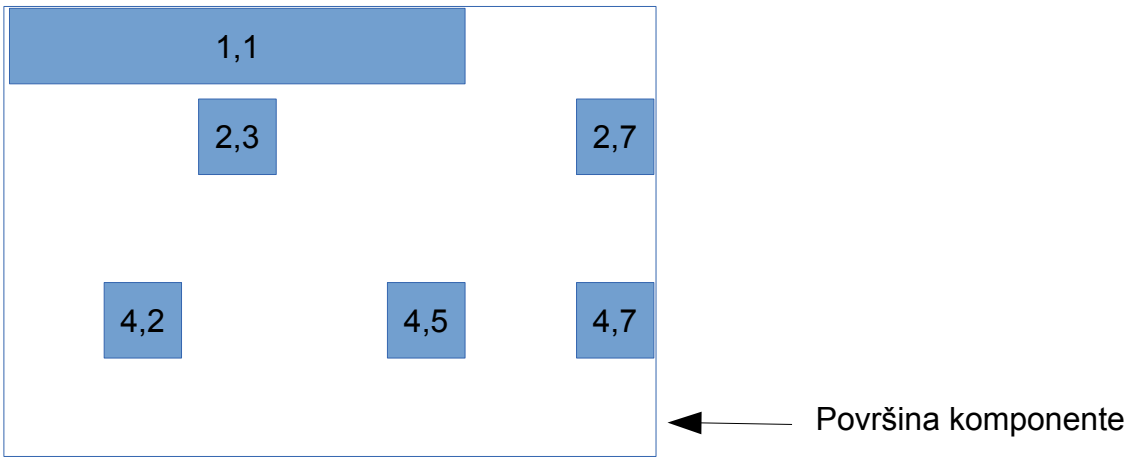
`java.awt.LayoutManager2`. Ograničenja s kojima on radi trebaju biti primjerci razreda `RCPosition` koji nudi dva *read-only* svojstva: `row` te `column` (oba po tipu `int`). Za raspoređivanje komponenti layout manager konceptualno radi s pravilnom mrežom dimenzija 5 redaka i 7 stupaca (ovo je fiksirano i nije moguće mijenjati). Numeracija redaka i stupaca kreće od 1. Izgled layouta je prikazan na slici u nastavku, a u komponentama su upisane njihove koordinate.

Svi retci mreže su jednako visoki; svi stupci mreže su jednako široki. Podržano je razmještanje najviše 31 komponente. Pri tome se komponenta koja je dodana na poziciju (1,1) uvijek razmješta tako da prekriva i pozicije (1,2) do (1,5); to znači da pokušaj dodavanje komponenti uz ograničenja (1,2) do (1,5) treba izazvati odgovarajuću iznimku (kao i bilo koja druga nelegalna pozicija, tipa (-2,0) ili (1,8) ili ...). Visina svih redaka je ista i određuje se kao maksimalna visina od preferiranih visina svih dodanih komponenti. Širina svih stupaca je ista i određuje se kao maksimalna širina od preferiranih širina svih komponenata (izuzev one na poziciji (1,1) – pazite kako taj broj morate interpretirati).

Prilikom raspoređivanja, legalno je da nisu prisutne sve komponente layouta; dapače, legalno je i da su čitavi retci ili stupci prazni – to ništa ne mijenja. Primjerice, sljedeći kod:

```
JPanel p = new JPanel(new CalcLayout(3));
p.add(new JLabel("x"), new RCPosition(1,1));
p.add(new JLabel("y"), new RCPosition(2,3));
p.add(new JLabel("z"), new RCPosition(2,7));
p.add(new JLabel("w"), new RCPosition(4,2));
p.add(new JLabel("a"), new RCPosition(4,5));
p.add(new JLabel("b"), new RCPosition(4,7));
```

bi trebao generirati razmještaj prikazan u nastavku.



`CalcLayout` mora podržati dva konstruktora: jedan koji prima željeni razmak između redaka i stupaca (u pikselima; tip `int`), tako da se može postići razmještaj u kojem komponente nisu zalijepljene jedna za drugu – on je prikazan u prethodnom primjeru. Drugi bez argumenata koji ovaj razmak postavlja na 0.

Prilikom stvaranja komponente koja koristi ovaj layout nužno je najprije nad komponentom instalirati ovaj layout manager (bilo kroz konstruktor ako to komponenta podržava, ili pozivom `.setLayout(...)`), i tek potom krenuti u dodavanje komponenti.

Potrebno je podržati i ograničenja koja se zadaju u obliku stringa koji tada mora biti propisane strukture. Primjerice, sljedeći kod bi morao stvoriti identičan layout prethodno prikazanome.

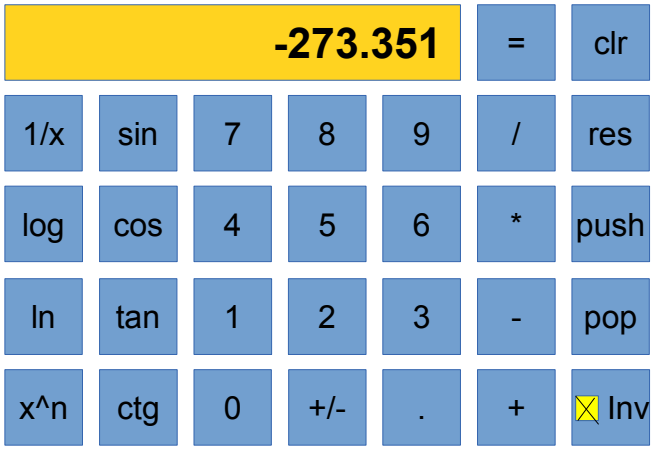
```
JPanel p = new JPanel(new CalcLayout(3));
p.add(new JLabel("x"), "1,1");
p.add(new JLabel("y"), "2,3");
p.add(new JLabel("z"), "2,7");
p.add(new JLabel("w"), "4,2");
p.add(new JLabel("a"), "4,5");
p.add(new JLabel("b"), "4,7");
```

Ako korisnik pokuša dodati dvije komponente pod istim ograničenjem, layout manager bi trebao izazvati iznimku. Ako se layout manager instalira u kontejner koji već sadrži druge komponente (za koje naš layout manager ne zna), slobodan ih je ignorirati.

Metode layout managera kojima bi on trebao kontejneru vratiti informacije o preferiranoj veličini layouta, te minimalnoj i maksimalnoj potrebno je izvesti malo pažljivije (za početak, obratite pažnju da neka komponenta kada ju pitate za neku od tih informacija može vratiti i `null`, čime poručuje da joj to nije bitno, odnosno nema nikakvog zahtjeva). Minimalna veličina layouta mora garantirati svakoj komponenti koja se izjasni da ima barem toliko mjesta – razmislite što to znači. Analogno vrijedi i za maksimalnu veličinu.

Zadatak 4.

Uporabom prethodno razvijenog layout managera napišite program `Calculator` (razred `hr.fer.zemris.java.gui.calc.Calculator`). Budete li stvarali dodatne razrede, stavite ih u isti paket u kojem je ovaj razred ili u njegove potpakete. Skica prozora kalkulatora prikazana je u nastavku.



Kalkulator funkcionira kao onaj standardni Windows kalkulator: broj se unosi klikanjem po gumbima sa znamenkama. "Ekran" koji prikazuje trenutni broj je komponenta `JLabel`: nije omogućen unos broja utipkavanjem preko tipkovnice. Primjerice, da biste izmnožili 32*2, naklikat ćete 3, 2, puta, 2, = i dobiti 64. Ako naklikate 3, 2, *, 2, + 1, =, na ekranu će se prikazati redom, "3", "32", "32" (sustav pamti operaciju *),

"2", "64" (i sustav pamti operaciju +), "1", "65".

Tipka "clr" briše samo trenutni broj (ali ne poništava operaciju; primjerice, 3, 2, *, 5, 1, clr, 2, = će i dalje izračunati 64; krenuli smo množiti s 51, predomislili se i pomnožili s 2). Tipka "res" kalkulator resetira u početno stanje. Tipka push trenutno prikazani broj stavlja na stog; npr. 3, 2, * 2, push, =, na stog stavlja 2, na jednako ispisuje 64. Tipka pop trenutni broj mijenja brojem s vrha stoga (ili dojavljuje da je stog prazan). Checkbox Inv (komponenta JcheckBox) obrće značenje operacija sin, cos, tan, ctg (u arkus funkcije), log u 10^, ln u e^, x^n u n-ti korijen iz x. Trigonometrijske funkcije podrazumijevaju da su kutevi u radijanima.

Uočite da dosta prikazanih tipki konceptualno radi vrlo slične obrade (tipa: pritisak na znamenku; pritisak na neku od tipki koje odmah djeluju poput "sin" koji odmah računa sinus trenutno prikazanog broja i na ekran zapisuje rezultat, i slično). Identificirajte takve grupe sličnih operacija i napišite generički kod u kojem ćete konkretno djelovanje (tipa: računam li sinus ili kosinus ili njima inverzne operacije) modelirati odgovarajućom *strategijom* (oblikovni obrazac Strategija).

Important notes

Solve problems 1 and 2 in one Eclipse project (*HW10a-yourJMBAG*) and problems 3 and 4 in another Eclipse project (*HW10b-yourJMBAG*). For 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 are required to write the adequate number of unit tests for this project.

You must equip your project with `build.xml` script so that the project can be build from the command line. In that script, you must integrate all of the quality-checking tools which I have described in the book (by integrate, I mean have a new target for each tool so that you can call each tool separately). It should be possible to run at least the following targets: `init`, `compile`, `compile-tests`, `jar`, `run-tests`, `quality`, `reports`, `clean`.

Target `quality` must run unit tests and all of the quality checks (i.e. *checkstyle*, *pmd*, *findbugs*): make it just a target which depends on all tool-targets. Unit tests must be run with the code coverage analysis. Target `reports` is a wrapper that will run all of the unit tests, the quality checks and the javadoc generation.

All of the classes should have appropriate javadoc.

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 which is not part of Java standard edition (Java SE) unless explicitly provided by me. This means that from this point on, you can use Java Collection Framework classes or its derivatives (moreover, I recommend it). 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) `HW10a-`*yourJMBAG* or `HW10b-`*yourJMBAG* (for the two projects you have to write). 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 for uploading and locking the second project (problems 3 and 4, project HW10b-yourJMBAG) is May, 17th 2014. at 07:00 (in the morning). Deadline for uploading and locking the first project (problems 1 and 2, project HW10a-yourJMBAG) is May, 19th 2014. at 11:59 PM.