10. homework assignment; JAVA, Academic year 2017/2018; FER

Napravite prazan Maven projekt, kao u 1. zadaći: u Eclipsovom workspace direktoriju napravite direktorij `hw10-0000000000` (zamijenite nule Vašim JMBAG-om) te u njemu oformite Mavenov projekt `hr.fer.zemris.java.jmbag0000000000:hw10-0000000000` (zamijenite nule Vašim JMBAG-om) i dodajte ovisnost prema `junit:junit:4.12`. Importajte projekt u Eclipse. Sada možete nastaviti s rješavanjem zadataka.

Zadatak 1.

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 upravljač razmještajem (engl. *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 upravljač 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 razmještaja je prikazan na slici u nastavku, a u komponentama su upisane njihove koordinate oblika (redak, stupac).

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) koje se stoga ne mogu koristiti; to znači da pokušaj dodavanje komponenti uz ograničenja (1,2) do (1,5) treba izazvati iznimku (kao i bilo koja druga nelegalna pozicija, tipa (-2,0) ili (1,8) ili ...), a iznimku treba izazvati i pokušaj dodavanja više komponenti uz isto ograničenje. Napišite novu iznimku CalcLayoutException koju izvedite iz RuntimeException; tu iznimku bacite u svim prethodno navedenim slučajevima.

Napišite junit testove koji će provjeriti:
1) da se iznimka baca ako se pokuša koristiti ograničenje (r,s) gdje je r<1 || r>5 ili s<1 || s>7,
2) da se iznimka baca ako se pokuša koristiti ograničenje (1,s) gdje je s>1 && s<6,
3) da se iznimka baca ako se uz isto ograničenje pokuša postaviti više komponenti (komponente uspoređivati s ==, ne pozivanjem metode equals).

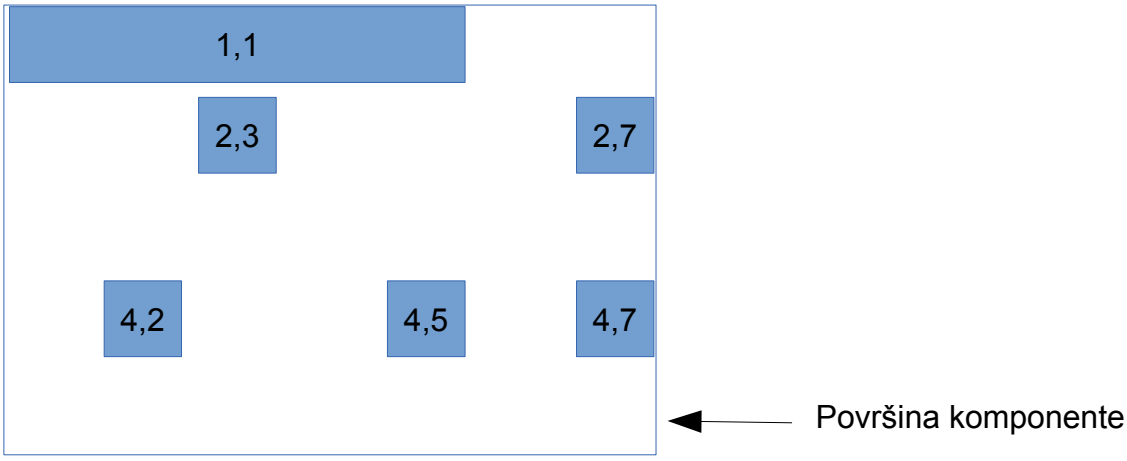
Pri izračunu *preferiranih* dimenzija razmještaja držite se sljedećih pretpostavki. Visina svih redaka je ista i određuje se kao maksimalna visina od preferiranih visina svih dodanih komponenata. Širina svih stupaca je ista i određuje se kao maksimalna širina od preferiranih širina svih komponenata (izuzev one na poziciji (1,1) – pazite kako taj broj morate interpretirati). Pri stvarnom raspoređivanju, moguće je da komponenta neće biti baš preferirane veličine: vaša je zadaća da im promijenite širinu i visinu tako da u cijelosti popunjavaju za njih predviđen prostor.

Prilikom raspoređivanja, legalno je da nisu prisutne sve komponente razmještaja; dapače, legalno je i da su čitavi retci ili stupci prazni – to ništa ne mijenja (u razmještaju na tim mjestima stvarno ostaju prazna polja).

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 jedan broj - željeni *razmak* između redaka i stupaca (u pikselima; tip `int`; isti broj koristi se i za razmak između redaka, i za razmak između stupaca), 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 razmještaj nužno je najprije nad komponentom instalirati ovaj upravljač (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 razmještaj 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, upravljač bi trebao izazvati iznimku. Ako se upravljač instalira u kontejner koji već sadrži druge komponente (za koje naš upravljač ne zna), slobodan ih je ignorirati.

Metode upravljača kojima bi on trebao kontejneru vratiti informacije o preferiranoj veličini razmještaja, te minimalnoj i maksimalnoj veličini, 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 razmještaja mora garantirati svakoj komponenti koja se izjasni da ima barem toliko mjesta – razmislite što to znači. Analogno vrijedi i za maksimalnu veličinu.

Napišite junit test koji će za sljedeći ispitni scenarij:

```
JPanel p = new JPanel(new CalcLayout(2));
Jlabel l1 = new Jlabel(""); l1.setPreferredSize(new Dimension(10,30));
Jlabel l2 = new Jlabel(""); l2.setPreferredSize(new Dimension(20,15));
p.add(l1, new RCPosition(2,2));
p.add(l2, new RCPosition(3,3));
Dimension dim = p.getPreferredSize();
```

provjeriti da je dim.width iznosa 152 te dim.height iznosa 158. Napišite junit test koji će za sljedeći ispitni scenarij:

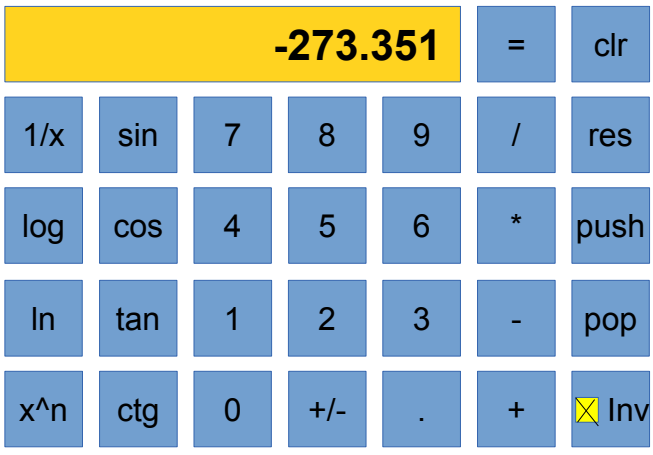
```
JPanel p = new JPanel(new CalcLayout(2));
Jlabel l1 = new Jlabel(""); l1.setPreferredSize(new Dimension(108,15));
Jlabel l2 = new Jlabel(""); l2.setPreferredSize(new Dimension(16,30));
p.add(l1, new RCPosition(1,1));
p.add(l2, new RCPosition(3,3));
Dimension dim = p.getPreferredSize();
```

provjeriti da je dim.width iznosa 152 te dim.height iznosa 158.

Zadatak 2.

Rješavanje nastavljate u istom projektu u kojem ste riješili prethodni zadatak.

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 (iz Windowsa XP) 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". Ovaj kalkulator ne podržava nikakve napredne mogućnosti niti poštuje prioritete operatora – izraz se računa direktno kako se utipkava; izraz 3+2*5 stoga se interpretira kao 3+2=6, pa 6*5=30. Također, za razliku od modernijih kalkulatora, ovdje nije moguće unijeti kompletan izraz (koji bi se i prikazao na ekranu) pa ga tek na "=" izračunati; računanje se provodi odmah po pritisku na "=" ili na sljedeći operator (ako je definiran prethodni i dva argumenta).

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. Za pohranu brojeva koristite tip *double* (ne trebate implementirati podršku za velike brojeve; nije poanta ovog zadatka).

Uočite da dosta prikazanih tipki konceptualno radi vrlo slične obrade; npr.

- svi gumbi koji predstavljaju znamenake 0 do 9 na pritisak rade gotovo isto – razlika je samo u znamenci koju treba dodati
- svi gumbi koji predstavljaju unarne operacije koje odmah djeluju poput "sin/arcsin", "cos/arccos" itd; oni bi konceptualno mogli imati reference na dvije unarne operacije i na pritisak ovisno o stanju "Inv" primijeniti jednu ili drugu
- svi gumbi koji predstavljaju binarne operacije (+, -, *, /).

Prilikom rješavanja ovog zadatka stoga se poslužite prikladnim oblikovnim obrascima (na lijep način možete iskoristiti Strategiju). Primijetite također da ne morate nužno u razmještaj dodavati direktno primjerke razreda JButton; možda redundancije u kodu možete izbjeći tako da napravite neki Vaš novi razred koji nasljeđuje JButton i koji kroz prikladan konstruktor dobije relevantne informacije. Takvih porodica razreda može biti i više.

Naputak za rješavanje ovog zadatka

U dodatak uz 10 domaću zadaću stavio sam ZIP arhivu s definicijama dvaju sučelja te jednim razredom s određenim brojem junit testova. Skinite tu arhivu i dodajte njezin sadržaj izravno u Vaš projekt (raspakirajte / prekopirajte datoteke).

Krećemo od sučelja `CalcModel` koje definira model jednog kalkulatora: Vi ćete napisati razred koji implementira taj model. Sučelje je prikazano u nastavku.

```
public interface CalcModel {
	void addCalcValueListener(CalcValueListener l);
	void removeCalcValueListener(CalcValueListener l);
	String toString();
	double getValue();
	void setValue(double value);
	void clear();
	void clearAll();
	void swapSign();
	void insertDecimalPoint();
	void insertDigit(int digit);
	boolean isActiveOperandSet();
	double getActiveOperand();
	void setActiveOperand(double activeOperand);
	void clearActiveOperand();
	DoubleBinaryOperator getPendingBinaryOperation();
	void setPendingBinaryOperation(DoubleBinaryOperator op);
}
```

Implementacija modela u varijabli tipa `String` (inicijalno `null`) čuva broj koji je korisnik utipkao; metoda `insertDigit` u taj string na kraj lijepi poslanu znamenku. Metoda `toString` vraća "0" ako je zapamćeni string `null`, a inače vraća zapamćeni string. Metoda `getValue` pri svakom pozivu zapamćeni string parsira u `double` i vraća (ako je isti različiti od `null`), odnosno inače vraća `0.0`. Metoda `setValue` primljeni `double` konvertira u string (ako nije beskonačnost ili NaN) i taj string postavlja kao zapamćenu vrijednost.

Metoda `swapSign` mijenja predznak zapamćenog broja (ali samo je barem jedna znamenka broja unesena); inače ne radi ništa. Metoda `insertDecimalPoint` u broj upisuje decimalnu točku, ako ona ne postoji (inače ne radi ništa).

Model pamti još dva podatka: aktivan operand (activeOperand) te zakazanu operaciju (pendingOperation). Ako aktivan operand nije postavljen, metoda `getActiveOperand` baca iznimku `IllegalStateException`.

Vrijednost, aktivan operand te zakazana operacija interagiraju na sljedeći način. Zamislite da smo tek upalili kalkulator, i želimo zbrojiti 58 i 14. Kad pošaljemo znamenke 5 pa 8, vrijednost će biti broj 58; kada kroz kalkulator korisnik sada pritisne gumb +, vrijednost 58 se mora zapisati u aktivan operand, mora se zakazati operacija zbrajanja (koju još uvijek nije moguće provesti jer nedostaje drugi argument), a vrijednost se mora očistiti kako bi korisnik unio drugi broj. Korisnik će zatim poslati znamenke 1 pa 4 čime će vrijednost biti 14. Ako sada korisnik pritisne gumb =, dohvatit će se aktivan operand (58), trenutna vrijednost 14 i zakazana operacija (+); ona će se provesti i rezultat 72 će se postaviti kao vrijednost (aktivan operand će se izbrisati, zakazana operacija će se izbrisati).

U složenijem slučaju, kada korisnik unosi 58 + 14 - 2 =, pri obradi znaka "-" možemo vidjeti što se općenito mora događati s binarnim operacijama. U trenutku kada korisnik pritisne "-", aktivan operand je 58, zakazana operacija je "+", vrijednost je 14 i nova željena operacija je "-"; kako već postoji zakazana operacija, ona se prvo mora izvesti (pa ćemo zbrojiti 58+14); rezultat se postavlja kao novi aktivni operand (72), zakazuje se nova operacija (oduzimanje) i vrijednost se čisti kako bi korisnik mogao početi unositi 2.

Poziv metode `clear` čisti trenutnu vrijednost (ali ništa osim toga). Poziv metode `clearAll` čisti trenutnu vrijednost, aktivan operand te zakazanu operaciju.

Napišite razred `CalcModelImpl` koji je implementacija opisane funkcionalnosti. Provjerite da Vam ista prolazi sa svim testovima koje sam pripremio. Slobodno dopišite još slučajeva koji Vam se učine zanimljivima.

Praćenje trenutne vrijednosti naš model omogućava uporabom oblikovnog obrasca Promatrač u kojem je on subjekt. Promatrači su modelirani sučeljem `CalcValueListener`. Svaki puta kada bilo od metoda modela uzrokuje promjenu trenutne vrijednosti, implementacija modela mora o tome obavijestiti sve zainteresirane promatrače.

Jednom kada ste ovo složili, možete krenuti na GUI.

Ekran (engl. display) kalkulatora modelirajte kao `JLabel` koji je promatrač nad modelom kalkulatora. Pritisak na gumbe koji modeliraju znamenke 0 – 9 mora rezultirati pozivom metode `insertDigit`; uporaba `switch`-a i drugih srodnih konstrukata koji potiču unošenje redundancija u kodu za ove je stvari zabranjena!

Podrška za igranje sa stogom nije uključena u model jer se lagano dodaje bez da je model toga svjestan.

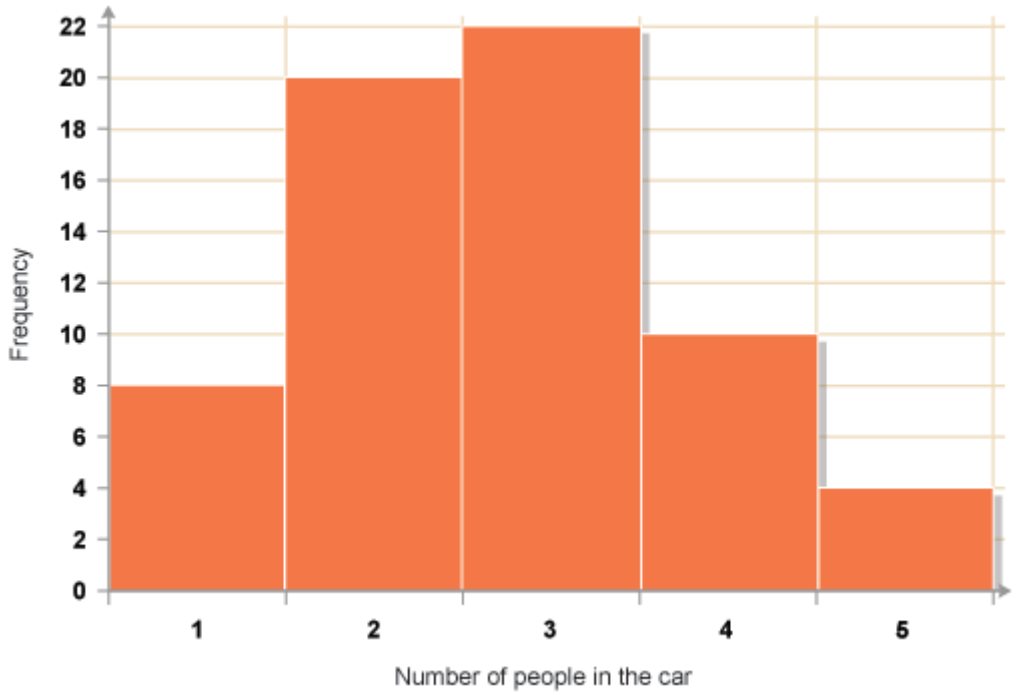
Zadatak 3.

Rješavanje nastavljate u istom projektu u kojem ste riješili prethodni zadatak.

U paketu `hr.fer.zemris.java.gui.charts` definirajte razred `XYValue` koji ima dva *read-only* property-ja: `x` i `y`, oba tipa `int`. Definirajte potom u istom paketu razred `BarChart` koji kroz konstruktor prima listu `XYValue` objekata, opis uz x-os te opis uz y-os, minimalni y koji se prikazuje na osi, maksimalni y koji se prikazuje na osi te razmak između dva susjedna y-a koji se prikazuju na osi (ako ymax-ymin ne dijeli taj razmak, pri crtanju radite s prvim većim y-onom koji je na cijelom broju razmaka od ymin).

Primjerice, podatke za dijagram prikazan na sljedećoj slici konstruirali bismo pozivom:

```
BarChart model = new BarChart(
  Arrays.asList(
    new XYValue(1,8), new XYValue(2,20), new XYValue(3,22),
    new XYValue(4,10), new XYValue(5,4)
  ),
  "Number of people in the car",
  "Frequency",
  0,      // y-os kreće od 0
  22,     // y-os ide do 22
  2
);
```



U paketu `hr.fer.zemris.java.gui.charts` definirajte razred `BarChartComponent` koji je izveden iz razreda `JComponent`. Ima jedan konstruktor koji prima referencu na `BarChart` i pamti je. Komponenta na svojoj površini stvara prikaz podataka koji su definirani primljenim objektom (tj. crta stupičasti dijagram). Pri izradi crteža, vodite se sljedećim zahtjevima.

- Komponenta s lijeve strane ispisuje naziv podataka s y-osi, slijedi neki (fiksni) razmak, vrijednosti po y-osi koje su desno poravnate (kao na slici), neki fiksni razmak, pa y-os.
- Komponenta s donje strane (gledano od samog dna) ispisuje naziv podataka s x-osi, slijedi neki

(fiksni) razmak, vrijednosti po x-osi koje su horizontalno centrirane ispod stupića (kao na slici), neki fiksni razmak, pa x-os.
- Za dio osi x i y koji su na kraju (gdje je strelica) uzmite također neki fiksni iznos (ili ga prilagodite po potrebi; tu imate slobodu).

Posljedica ova tri zahtjeva je sljedeća: kada se komponenta razvlači, udaljenost osi y od lijevog ruba te osi x od dna ostaje fiksna: mijenja se samo visina y-osi, širina x-osi te sam prostor u kojem se prikazuju stupići (čime i stupići postaju širi/uži, viši/niži). Slika uvijek mora popunjavati cjelokupnu površinu komponente. Pri razvlačenju komponente nemojte ništa mijenjati oko fontova: font (i veličina fonta) neka uvijek bude ista.

Ne smijete unaprijed ništa pretpostavljati o broju znamenaka vrijednosti koje se ispisuju na y-osi (osim da je sve lijepo prikazivo int-om).Udaljenost osi y od lijevog ruba mora "plesati" ovisno o stvarno potrebnom prostoru za prikaz brojeva. Rješenje koje unaprijed alocira fiksan broj (npr. 300 piksela, za svaki slučaj) u nadi da će svi brojevi biti prikazivi nije prihvatljivo. Kako se među vrijednostima koje crtate mogu pojavljivati i pozitivni i negativni brojevi, držat ćemo se sljedećeg dogovora: dno stupca konceptualno kreće od minus beskonačno i ide do zadanog broja; alternativa bi bila da pozitivne vrijednosti crtamo od nule prema gore a negative od nule prema dolje; to ovdje nećemo raditi.

U paketu `hr.fer.zemris.java.gui.charts` definirajte razred `BarChartDemo` koji je izveden iz `JFrame` i prikazuje preko čitave svoje površine jedan primjerak grafa. Dodajte tom razredu i metodu `main` tako da sve možete pokrenuti iz naredbenog retka. Program prima jedan argument: stazu do datoteke u kojoj je opis grafa. Primjer takve datoteke (za prethodno prikazani graf) prikazan je u nastavku (sve između vodoravnih crta). Program otvara datoteku, temeljem sadržaja stvara objekt `BarChart`, njega predaje prozoru koji dalje radi što treba. Uz gornji vrh prozora stavite jednu labelu u kojoj će biti ispisana (vodoravno centrirano) putanja do datoteke iz koje su podatci dohvaćeni.

```
Number of people in the car
Frequency
1,8 2,20 3,22 4,10 5,4
0
22
2
```

U datoteci gledate prvih 5 redaka. Točke su međusobno odvojene razmacima a komponente točke zarezom. Ako bilo što ne štima s formatom datoteke koju program dobije, javite to korisniku i prekinite izvođenje.

Pomoć: vertikalni ispis teksta:
http://www.codejava.net/java-se/graphics/how-to-draw-text-vertically-with-graphics2d
(nemojte nakon aktiviranja rotacije i ispisa teksta zaboraviti poništiti daljnje rotiranje).

PS. Ne morate bacati sjenu (kao na slici) ako nemate ideju kako to napraviti. Ta je slika samo okvirni prikaz onoga što želimo dobiti.

Važno: čitava komponenta koja prikazuje bar-chart zajedno s koordinatnim sustavom i nazivom obje osi je jedna komponenta koja izravno crta svoju površinu i NE sadrži druge komponente (ne smijete unutra dodavati labele i slično). Ideja ovog zadatka je da savladate uporabu objekta Graphics za izravno crtanje po površini komponente. Također, prikazana slika je samo okvirna: ne morate je reproducirati u piksel jednako, ali funkcionalnost mora biti zadovoljena.

Zadatak 4.

Rješavanje nastavljate u istom projektu u kojem ste riješili prethodni zadatak.

Razrede smještate u paket `hr.fer.zemris.java.gui.prim`.

Napišite program `PrimDemo`: to je program koji se pokreće iz naredbenog retka bez ikakvih argumenata. Po pokretanju otvara prozor u kojem se prikazuju dvije liste (jednako visoke, jednako široke, rastegnute preko čitave površine prozora izuzev donjeg ruba). Uz donji rub dodajte gumb "sljedeći".

Napišite model liste `PrimListModel` koji nudi metodu `next()`. Model inkrementalno generira prim brojeve (po stvaranju, u modelu se nalazi samo broj 1). Na svaki poziv metode `next()` model u popis brojeva doda prvi sljedeći prim broj. Prozor koji prikazuje liste treba obje liste registrirati **nad istim** primjerkom modela (drugim riječima, smijete stvoriti samo jedan primjerak razreda `PrimListModel`. Klik na gumb poziva metodu `next()` modela; kao posljedica, obje liste moraju prikazati i taj novododani broj. Skrolovi se trebaju pojaviti automatski kada su potrebni. Razred `PrimListModel` morate izvesti iz vršnog sučelja modela i sami morate implementirati svu potrebnu funkcionalnost (nije dozvoljena uporaba razreda koji već nude dio funkcionalnosti).

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), except the libraries mentioned in this homework which are available on Ferko. You can use Java Collection Framework and other parts of Java covered by lectures; if unsure – e-mail me. Document your code!

If you need any help, I have reserved a slot for consultations: Monday Noon, Tuesday 10AM, Wednesday 9AM. Feel free to drop by my office (after e-mail).

All source files must be written using UTF-8 encoding. All classes, methods and fields (public, private or otherwise) must have appropriate javadoc.

You are expected to test model created in problem 4 as well as add tests described in problem 1.
You are encouraged to write tests for other problems.

When your **complete** homework is done, pack it in zip archive with name `hw10-0000000000.zip` (replace zeros with your JMBAG). Upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is May 17th 2018. at 07:00 AM.