

7th homework assignment; OPRPP1

Napravite prazan Maven projekt: u Eclipseovom workspace direktoriju napravite direktorij `hw07-0000000000` (zamijenite nule Vašim JMBAG-om) te u njemu oformite Mavenov projekt `hr.fer.oprpp1.jmbag0000000000:hw07-0000000000` (zamijenite nule Vašim JMBAG-om) i dodajte ovisnost prema junit. 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 razmještaja (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`). Razred treba ponuditi i javnu statičku metodu tvornicu koja `RCPosition` `parse(text)` koja prima tekstovnu specifikaciju (poput "3,7") i vraća odgovarajući objekt tipa `RCPosition`. 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). Napomena: prilikom računanja razmještaja (na kojem x-u kreće svaki od "virtualnih" stupaca te koliko je širok, i slično za retke) koristite decimalne brojeve pa njih na kraju prikladno zaokružite. Ako to ne napravite, dogodit će Vam se da prilikom finog polaganog proširenja dimenzija prozora jedno se vrijeme ništa ne mijenja a raste prazan prostor uz desni odnosno donji rub prozora, pa zatim u nekom trenutku skokovito nastane novi razmještaj. To bismo htjeli izbjeći. Primijetite također da zbog toga svi stupci ne moraju biti jednako široki (moguće je odstupanje od jednog slikovnog elementa), a isto vrijedi i za visine redaka. Primjerice, ako je širina prozora 200, te imate 7 stupaca, širina stupca bi bila $200/7=28.57\dots$; ako to zaokružimo, dobivamo 29, no $29*7=203$ a toliko mjesta nemamo. Stoga će jedan dio stupaca stvarno biti širok 29, a jedan dio stupaca 28, i ove bi različitosti trebalo uniformno distribuirati kroz razmještaj (u ovom slučaju primjerice $29+28+29+28+29+28+29$, a nikako $29+29+29+29+28+28+28$).

1,1 					1,6	1,7
2,1	2,2	2,3	2,4	2,5	2,6	2,7
3,1	3,2	3,3	3,4	3,5	3,6	3,7
4,1	4,2	4,3	4,4	4,5	4,6	4,7
5,1	5,2	5,3	5,4	5,5	5,6	5,7

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 dodavanja 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 ili pak iste komponente uz više ograničenja. Napišite novu iznimku `CalcLayoutException` koju izvedite iz `RuntimeException`; tu iznimku bacite u svim prethodno navedenim slučajevima. Detaljniji opis ponašanja dan je kasnije uz opis odgovarajuće metode upravljača.

Napišite junit testove koji će provjeriti:

- 1) da se iznimka baca ako se pokuša koristiti ograničenje (r,s) gdje je $r < 1 \parallel r > 5$ ili $s < 1 \parallel s > 7$,
- 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.

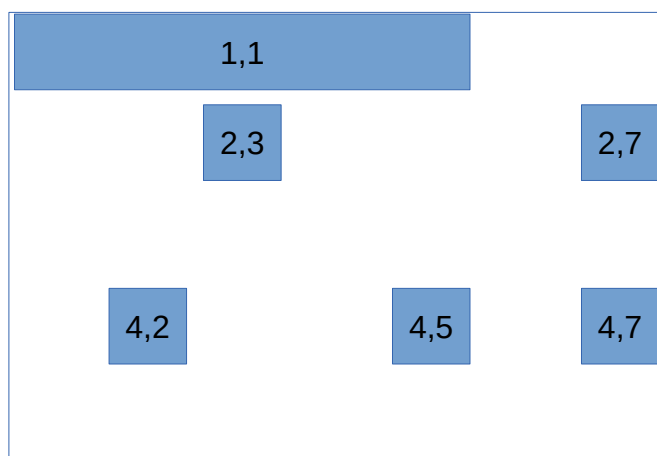
Pri izračunu **preferiranih dimenzija** razmještaja držite se sljedećih pretpostavki. Visina svih redaka je ista i određuje se kao **maksimalna visina od preferiranih visina svih dodanih komponenata**. Širina svih stupaca je ista i određuje se kao **maksimalna širina od preferiranih širina svih komponenata (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 (u skladu s prethodnim naputkom o uporabi decimalnih brojeva za proračune).

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.



← Površina komponente

`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.

Evo u nastavku par primjera. Neka je prozor definiran na sljedeći način:

```
public class DemoFrame1 extends JFrame {
    public DemoFrame1() {
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        initGUI();
        pack();
    }

    private void initGUI() {
        Container cp = getContentPane();
```

```

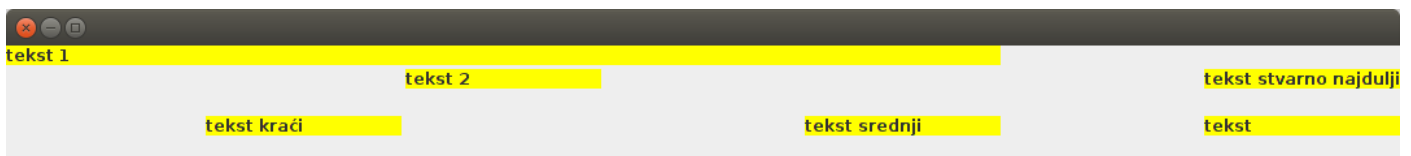
        cp.setLayout(new CalcLayout(3));
        cp.add(l("tekst 1"), new RCPosition(1,1));
        cp.add(l("tekst 2"), new RCPosition(2,3));
        cp.add(l("tekst stvarno najdulji"), new RCPosition(2,7));
        cp.add(l("tekst kraći"), new RCPosition(4,2));
        cp.add(l("tekst srednji"), new RCPosition(4,5));
        cp.add(l("tekst"), new RCPosition(4,7));
    }

    private JLabel l(String text) {
        JLabel l = new JLabel(text);
        l.setBackground(Color.YELLOW);
        l.setOpaque(true);
        return l;
    }

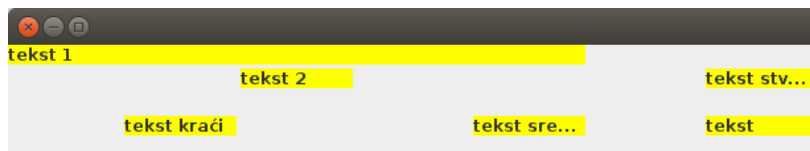
    public static void main(String[] args) {
        SwingUtilities.invokeLater(()->{
            new DemoFrame1().setVisible(true);
        });
    }
}

```

Po pokretanju, trebao bi se prikazati prozor kao na slici u nastavku.



Primimo li donji desni ugao mišem pa malo smanjimo dimenzije prozora:



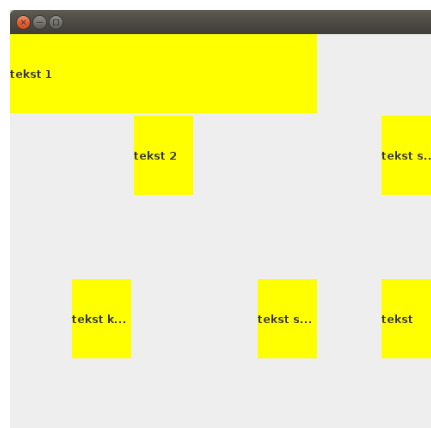
Ako pak promijenimo konstruktor tako da bude:

```

public DemoFrame1() {
    setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    setSize(500, 500);
    initGUI();
}

```

pokretanjem ćemo dobiti (slika je umanjena da stane na dno stranice):



Vezano uz izradu raspoređivača razmještaja, evo nekoliko općenitih napomena.

Raspoređivač razmještaja pri vraćanju informacija o minimalnim, maksimalnim i preferiranim dimenzijama rasporeda treba uvažiti i činjenicu da ako kontejner ima border (ili neki drugi razlog zbog kojeg postoje okolne margine), vraćene dimenzije to trebaju uzeti u obzir. Pri razmještanju isto tako treba paziti da se eventualno postojeće margine uzmu u obzir (pa komponenta primjerice ne može doći već na lokaciju (0,0)).

Metode **preferredLayoutSize**, **minimumLayoutSize** i **maximumLayoutSize** računaju i vraćaju preferirane dimenzije kontejnera, minimalne dimenzije kontejnera odnosno maksimalne dimenzije kontejnera na temelju komponenata koje su dodane u njega i povezanih ograničenja. Nitko ne kaže da će kontejner doista i biti tih dimenzija. Ovo su samo metode kojima raspoređivač razmještaja može izraziti svoje želje pozivatelju. Ove sve tri metode konceptualno rade vrlo slične stvari, te ih se lijepo daje napisati tako da ne duplicirate kod; razmislite kako!

U metodi **addLayoutComponent(String name, Component comp)** slobodno bacite iznimku **UnsupportedOperationException**; naime, ne želimo da itko poziva tu metodu.

U metodi **addLayoutComponent(Component comp, Object constraints)** zapamtite da je dodana komponenta uz predano ograničenje. Pri tome provjerite pa ako ograničenje nije tipa **RCPosition** i nije tipa **String**, bacite iznimku **IllegalArgumentException**. Ako je ograničenje tipa **String**, isparsirajte ga u objekt tipa **RCPosition** (pa ako nije parsabilno, bacite **IllegalArgumentException**). Ako nešto ne štima s ograničenjem (ali ono jest korektnog tipa) ili ako već postoji komponenta povezana s istim ograničenjem, bacite **CalcLayoutException**. Ako je kao ograničenje ili kao komponenta predan **null**, bacite **NullPointerException**.

U metodi **removeLayoutComponent(Component comp)** uklonite sve informacije koje ste imali o predanoj komponenti jer ona više nije u kontejneru, pa isti to ovom metodom dojavljuje i Vama.

U metodama **getLayoutAlignmentX** i **getLayoutAlignmentY** možete vratiti 0.

Zadaca metode **layoutContainer(Container parent)** je da doista napravi razmještaj, odnosno da nad svakom komponentom pozove metodu **setBounds(x, y, w, h)**. Sam kontejner koji dobivate kroz argument metode Vam zna reći koje su njegove trenutne dimenzije: one ne moraju odgovarati ničemu što ste prethodno izračunali – ne moraju odgovarati preferiranim, minimalnim ili maksimalnim dimenzijama. Vaš je zadatak na temelju aktualnih dimenzija dobivenog kontejnera proračunati širine stupaca i visine redaka (u skladu s prethodnim napomenama), i zatim svaku od komponenata pozicionirati na proračunati dio površine kontejnera.

Dodatna napomena (ili: kako stvari rade). Kada nad kontejnerom pozivamo metodu **add(komponenta, ograničenje)**, kontejner će komponentu dodati u svoju internu kolekciju, provjerit će je li nad njim instaliran **LayoutManager2**, i ako je, nad njim će pozvati metodu **addLayoutComponent(komponenta, ograničenje)** kako bi mu dojavio da je dodana nova komponenta o kojoj treba brinuti. Stoga je važno da se nad kontejnerom kao prvi korak upravo postavi raspoređivač razmještaja, a tek potom obavi dodavanje komponenata. U takvom scenariju objekte koji su raspoređivači razmještaja nikada nećemo dijeliti između više kontejnera. Ako imamo raspoređivač razmještaja koji interno ništa ne pamti, takav se raspoređivač teoretski može dijeliti između više kontejnera, što je razlog zašto metoda **layoutContainer** kao argument dobiva referencu na kontejner nad kojim treba napraviti razmještaj. Takav bi dijeljeni raspoređivač razmještaja predani kontejner pitao za sve potrebne informacija (kolike su mu dimenzije, koje su komponente dodane u njega i slično) te bi proveo raspoređivanje.

Zadatak 2.

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

Uporabom prethodno razvijenog upravljača razmještajem 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 potpake. Skica prozora kalkulatora prikazana je u nastavku.

-273.351					=	clr
1/x	sin	7	8	9	/	res
log	cos	4	5	6	*	push
ln	tan	1	2	3	-	pop
x^n	ctg	0	+/-	.	+	☒ Inv

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 $32 \cdot 2$, naklikat ćete 3, 2, puta, 2, = i dobiti 64. Ako naklikate 3, 2, *, 2, + 1, =, na ekranu će se prikazati redom, "3", "32", "32" (jer ćete trenutno prikazani broj zamrznuti, a sustav pamti operaciju *), "2" (jer je slanjem znamenke briše zamrznuti prikaz pa se prikazuje trenutni broj), "64" (jer se rezultat zamrzava za prikaz a interno se trenutni broj postavlja na 0 te sustav pamti operaciju +), "1" (jer se slanjem znamenke odmrzava prikaz), "65". Ovaj kalkulator ne podržava nikakve napredne mogućnosti niti poštuje prioritete operatora – izraz se računa direktno kako se utipkava; izraz $3+2 \cdot 5$ stoga se interpretira kao $3+2=5$, pa $5 \cdot 5=25$. 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^x , *ln* u e^x , 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 7 domaću zadaću stavio sam ZIP arhivu s definicijama dvaju sučelja, jedne iznimke te jednim razredom s određenim brojem junit testova koja kroz same testove specificira ponašanje tražene implementacije kalkulatora napisanog prema modelu specificiranom u nastavku. 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);
    boolean isEditable();
    void clear();
    void clearAll();
    void swapSign() throws CalculatorInputException;
    void insertDecimalPoint() throws CalculatorInputException;
    void insertDigit(int digit)
        throws CalculatorInputException, IllegalArgumentException;
    boolean isActiveOperandSet();
    double getActiveOperand() throws IllegalStateException;
    void setActiveOperand(double activeOperand);
    void clearActiveOperand();
    DoubleBinaryOperator getPendingBinaryOperation();
    void setPendingBinaryOperation(DoubleBinaryOperator op);
    void freezeValue(String value);
    boolean hasFrozenValue();
}
```

Implementacija modela bi mogla imati pet članskih varijabli: zastavicu koja govori je li model editabilan (u smislu da korisnik može dopisati novu znamenku, dodati decimalnu točku te promijeniti predznak broja), zastavicu koja govori je li broj pozitivan ili negativan, string varijablu koja pamti unesene znamenke, numeričku varijablu koja čuva decimalnu vrijednost usklađenu s onom iz string varijable te člansku varijablu tipa String koja čuva zamrznutu vrijednost prikaza (početno null). Sadržaj string varijable je inicijalno prazan string, a decimalna vrijednost je 0. Metoda insertDigit ako model nije editabilan baca iznimku CalculatorInputException; ako je model editabilan, u string varijablu na kraj lijepi poslanu znamenku, ali samo u slučaju time nastaje string koji se i dalje može isparsirati u konačni decimalni broj i tu vrijednost također pamti u varijabli koja čuva numeričku vrijednost; ako to nije slučaj, pohranjeni string se ne mijenja, a metoda treba baciti iznimku CalculatorInputException. Metoda toString, ako postoji zamrznuta vrijednost (nije null) za prikaz (metoda freezeValue!), vraća tu vrijednost. U suprotnom vraća "0" ili "-0" ako je zapamćeni string prazan (i ovisno o predznaku), a inače vraća zapamćeni string uz prikladan predznak. Metoda getValue pri svakom pozivu vraća zapamćenu numeričku vrijednost (ništa ne parsira – očekuje se da je zapamćena numerička vrijednost usklađena s tekstom koji se pamti). Metoda setValue primljeni double zapisuje u numeričku varijablu, a primljenu vrijednost dodatno konvertira u string i taj string također pamti. Pozivom ove metode model prestaje biti editabilan, tako da bi izravno nakon nje poziv metode poput swapSign() trebao baciti iznimku. Primijetite da broj koji se predaje u

`setValue` može biti i pozitivno ili negativno beskonačno te `NaN`, u kojem slučaju se od metode `toString` očekuje `Infinity`, `-Infinity` odnosno `NaN`. Poziv bilo koje metode kojom korisnik mijenja upisani broj (promjena predznaka, unos znamenke, unos točke) automatski uklanja zamrznutu vrijednost. Metoda `swapSign` mijenja predznak zapamćenog broja. Metoda `insertDecimalPoint` u broj upisuje decimalnu točku, ako ona ne postoji (inače baca iznimku `CalculatorInputException`). Ove obje metode u slučaju da model nije editabilan ne rade nikakvu promjenu već bacaju iznimku `CalculatorInputException`.

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 (vidi među predanim testovima test imena `exampleFromHomeworkAssignment`). 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 (operacijom brisanja model ponovno postaje editabilan). 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).

Ako kalkulator ima postavljenu zamrznutu vrijednost, klik na gumb koji odgovara unarnoj ili binarnoj operaciji treba baciti iznimku `CalculatorInputException` (primijetite da model kalkulatora ne zna za gumb – ovo ćete implementirati izravno u kodu GUI-ja).

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.

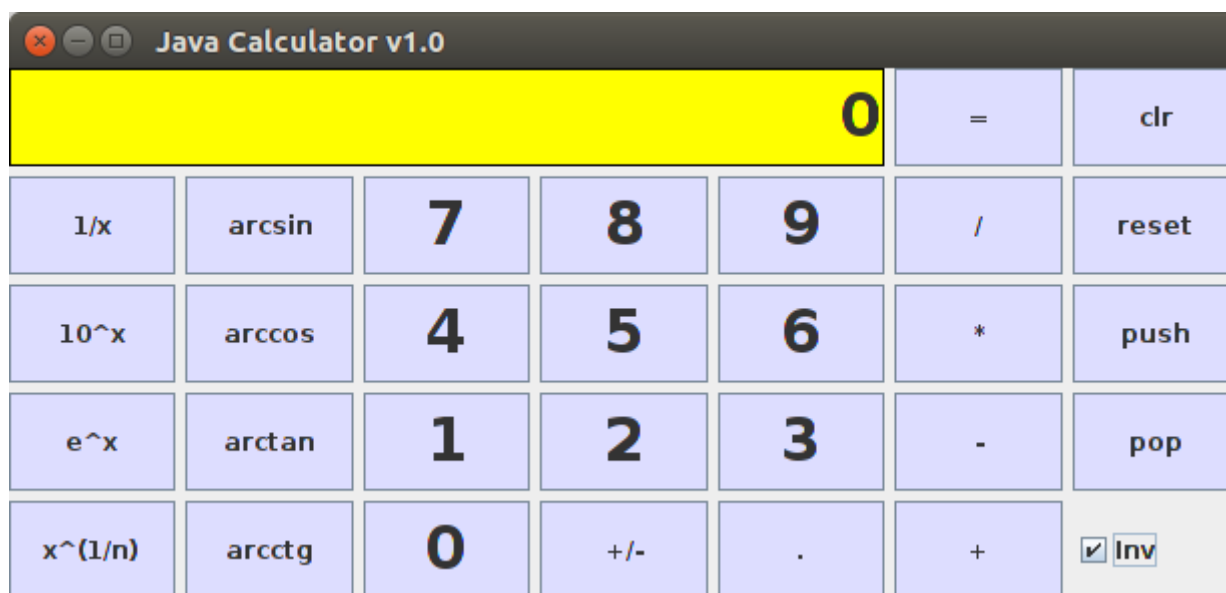
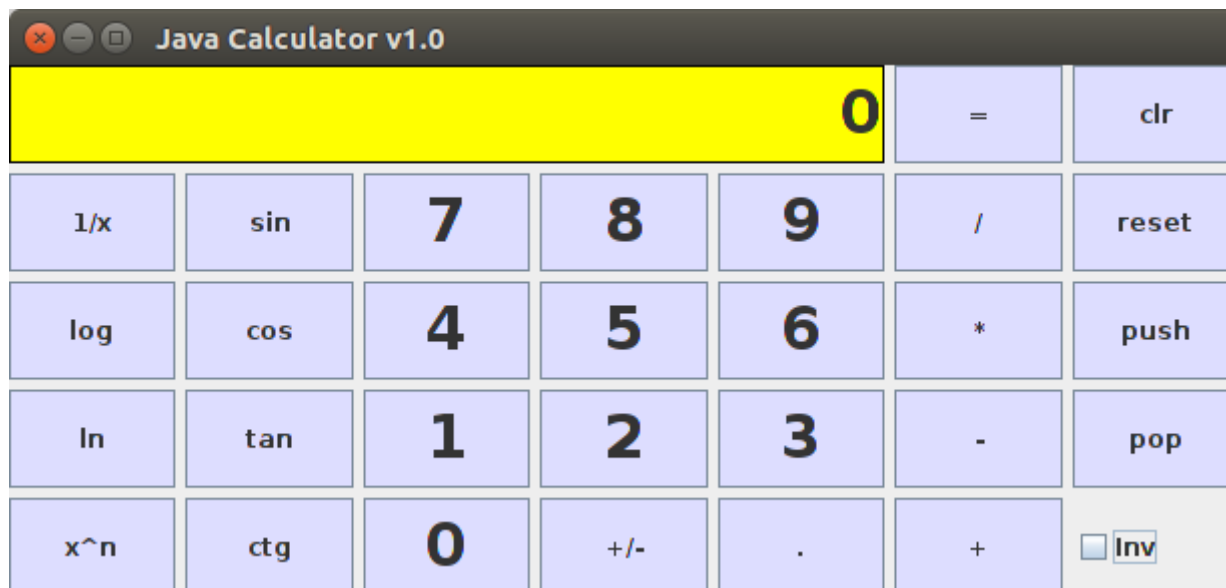
Ekran (engl. display) kalkulatora modelirajte kao `JLabel` koji je promatrač nad modelom kalkulatora. Pritisak na gumb koji modeliraju znamenke 0 – 9 mora rezultirati pozivom metode `insertDigit` nad modelom; uporaba `switch`-a i drugih srodnih konstrukata koji potiču unošenje redundancija u kodu za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

Podrška za igranje sa stogom nije uključena u model jer se lagano dodaje bez da je model toga svjestan. U nastavku su dani i mogući prikazi izgleda implementiranog rješenja.

Gumbima pozadinu možete definirati metodom `setBackground`. Veći font u nekim komponentama možete postići pozivom:

```
komponenta.setFont(komponenta.getFont().deriveFont(30f));
```

Kako korisnik može unijeti bilo što je parsabilno u double, to znači da može naklikati preko 300 znamenaka. Zanimajte činjenicu da se taj broj neće ispravno prikazati u labeli na vrhu kalkulatora zbog premalo mjesta (nemojte na rješavanje toga trošiti vrijeme).



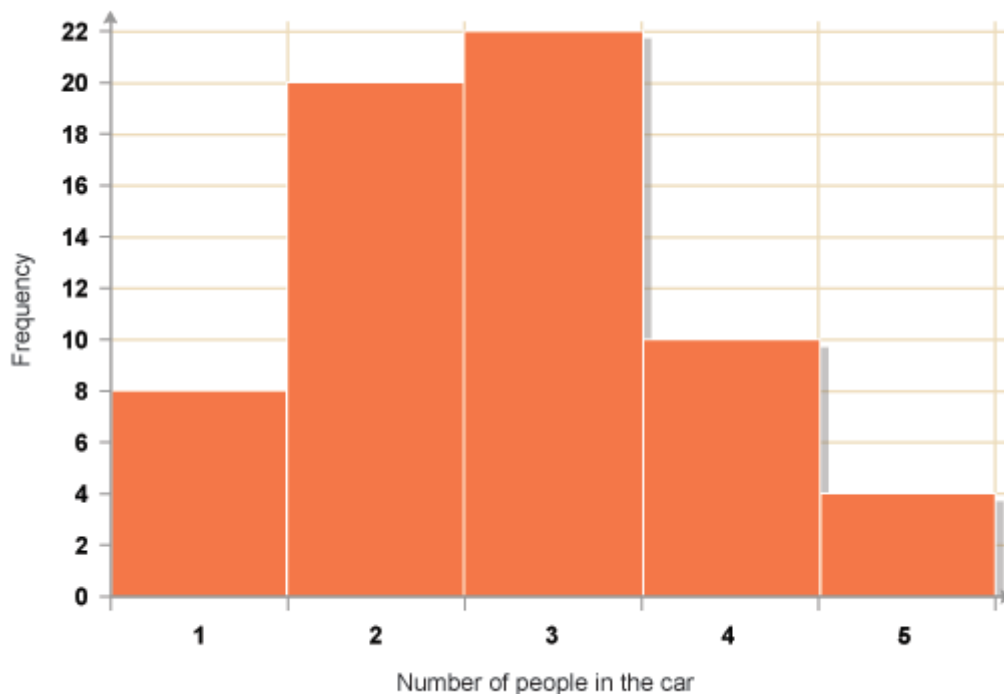
Zadatak 3.

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 (baciti iznimku ako je negativan), maksimalni `y` koji se prikazuje na osi (baciti iznimku ako nije strogo veći od predanog minimuma) te razmak između dva susjedna `y`-a koji se prikazuju na osi (ako $y_{\max} - y_{\min}$ ne dijeli taj razmak, pri crtanju radite s prvim većim `y`-om koji je na cijelom broju razmaka od y_{\min}). Konstruktor od `BarChart` treba napraviti provjeru jesu li `y`-vrijednosti u svim podacima predanim u listi veće ili jednake od predanog y_{\min} pa ako postoji neka koja je manja, treba baciti iznimku. Ne radi se provjera za maksimume.

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 stupčasti 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 prikazivo/zapisivo tipom `int`). Udaljenost osi y od lijevog ruba mora "plesati" ovisno o stvarno potrebnom prostoru za prikaz brojeva. Rješenje koje unaprijed alocira fiksni broj (npr. 300 piksela, za svaki slučaj) u nadi da će svi brojevi biti prikazivi nije prihvatljivo.

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 podaci dohvaćeni (spomenuta labela je primjerak od `JLabel`, komponenta tipa `BarChartComponent` ne prikazuje na sebi na vrhu nikakvu labelu); ove dvije komponente rasporedite u prozoru prikladnim upravljačem razmještaja.

```
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 (jednim ili više), a komponente točke zarezom (ispred/iza zareza ne može biti razmaka). Ako bilo što ne štima s formatom datoteke koju program dobije, javite to korisniku i prekinite izvođenje. Ako datoteka ima više od 6 redaka, sve ostale retke treba zanemariti (tako da si korisnik teoretski u nastavku može pisati različite bilješke o grafu koje njemu nešto znače, a mi ih ignoriramo). x-vrijednosti tretirajte doslovno kao "nazive"; ako skup podataka ima primjerice samo tri podatka, i to za $x=1$, za $x=17$ i $x=35$, dijagram će i dalje imati samo tri stupca (svaki širine poprilično jedne trećine raspoloživog prostora), i ispod prvog će pisati 1, ispod drugog 17, a ispod trećeg 35.

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.

Slika dijagrama koju generirate treba imati sljedeće **obavezne** elemente:

1. prikazane koordinatne osi zajedno sa strelicama na vrhu (bilo kakvim; popunjenim, nepopunjenim)
2. prikazane popunjene stupce do visine koja odgovara vrijednosti koju reprezentiraju (mogu, a ne moraju puniti čitavu raspoloživu širinu stupca; mogu, a ne moraju imati prikazane sjene)
3. prikazane brojeve po osima koji:
 1. za horizontalnu os napisani u centru svakog od stupaca
 2. za okomitu os sredina visine broja podudara se s crticom na osi koja predstavlja tu vrijednost; desna strana svakog od ovih brojeva je jednako udaljena od okomite osi (drugim riječima, svi brojevi mjesto “jedu” u lijevo – vidi sliku!)
4. udaljenost okomite osi od lijevog ruba komponente je “minimalno potrebna” (u smislu da se prilagođava prikazanim brojevima: ako su svi jednoznačenosti, os će biti bliža lijevom rubu no kada su brojevi primjerice troznačenosti)
5. kako se rastežu dimenzije komponente (mijenjajte mišem veličinu prozora), tako se mora mijenjati i prikazani dijagram, ali pri tome mora vrijediti sljedeće:
 1. s promjenom veličine prozora veličina fonta se ne mijenja
 2. okomita os ostaje jednako udaljena od lijeve strane komponente
 3. vodoravna os ostaje jednako udaljena od dna komponente (ta udaljenost uvijek je jednaka visini naziva osi plus visini ispisanih brojeva plus još malo razmaka ali koji ne ovisi o dimenzijama komponente, uz pretpostavku da komponenta nije premala)
6. Prikaz mreže (izgrađene od vodoravnih i okomitih linija kao pozadina dijagrama) nije nužan.

Točka (3) osigurava da se rastezanjem komponente površina distribuira u prostor za crtanje stupaca (njihovu širinu i visinu), dok udaljenosti osi od lijevog odnosno donjeg ruba ostaju fiksne.

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 i taj se prikazuje u listama). Na svaki poziv metode `next()` model u popis brojeva dodaje prvi sljedeći prim broj (primjerice, nakon tri poziva metode `next()` očekujemo da je sadržaj modela 1, 2, 3, 5). 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!

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 `hw07-0000000000.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.