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

Napravite prazan Maven projekt, kao u 1. zadaći: u Eclipsovom workspace direktoriju napravite direktorij `hw04-0000000000` (zamijenite nule Vašim JMBAG-om) te u njemu oformite Mavenov projekt `hr.fer.zemris.java.jmbag0000000000:hw04-0000000000` (zamijenite nule Vašim JMBAG-om) i dodajte ovisnost prema `junit:junit:4.12`. Importajte projekt u Eclipse. Sada možete nastaviti s rješavanjem zadataka.

Problem 1.

U zadaću iskopirajte razrede paketa `hr.fer.zemris.java.custom.collections` koje ste razvili u drugoj domaćoj zadaći. Konkretno, trebate razrede `ArrayIndexedCollection` i `ObjectStack` te sve ostale koji su potrebni da bi se ovaj kod uredno iskompajlirao.

Sada u isti paket dodaje razred `Dictionary`. Radi se o razredu koji pod zadanim ključem pamti predanu vrijednost (u svijetu Jave, ovakve strukture poznate su kao mape; u PHP-u i JavaScriptu kao asocijativna polja; u Pythonu i C# kao riječnici). U ovom zadatku, s obzirom na mali broj zapisa koje ćemo pohranjivati, naglasak neće biti na učinkovitosti već na jednostavnosti. Stoga ćete modelirati razred `Dictionary` kao *adapter* (vidi 2. domaću zadaću za pojam) oko `ArrayIndexedCollection`. Zapis ćete modelirati privatnim ugniježđenim razredom koji će omogućiti pamćenje dva podatka: `key` i `value`. Ključ pri tome ne smije biti `null`; vrijednost smije. Razred `Dictionary` mora korisnicima ponuditi sljedeće metode.

```
boolean isEmpty();
int size();
void clear();
void put(Object key, Object value); // "gazi" eventualni postojeći zapis
Object get(Object key);             // ako ne postoji pripadni value, vraća null
```

Napišite junit testove za Vašu implementaciju ovog zadatka!

Problem 2.

Napravite paket `hr.fer.zemris.math` i u njega smjestite razred `Vector2D`. Razred modelira 2D vektor čije su komponente realni brojevi `x` i `y`. Razred treba ponuditi sljedeće konstruktore/metode.

```
public Vector2D(double x, double y);
public double getX();
public double getY();
public void translate(Vector2D offset);
public Vector2D translated(Vector2D offset);
public void rotate(double angle);
public Vector2D rotated(double angle);
public void scale(double scaler);
public Vector2D scaled(double scaler);
public Vector2D copy();
```

Pri tome operacije `translate`/`rotate`/`scale` direktno modificiraju trenutni vektor dok metode `translated`/`rotated`/`scaled` vraćaju novi vektor koji odgovara rezultatu primjene zadane operacije nad trenutnim vektorom (pri čemu se trenutni vektor ne mijenja).

Napišite junit testove za sve metode!

Problem 3.

Na Ferku sam stavio dostupan JAR imena lsystems.jar. Skinite je, smjestite u poddirektorij lib u Vašem projektu i dodajte u Vaš projekt (istražite kako mavenu kroz pom.xml reći da koristi jar koji nije objavljen kroz maven repozitorij). Ova biblioteka sadrži jezgru za vizualizaciju jedne vrste fraktala: Lindermayerovih sustava. S ovom vrstom fraktala možete se upoznati googleanjem, ili pak proučite u knjizi:
http://java.zemris.fer.hr/nastava/irg/knjiga-0.1.2016-03-02.pdf
kratak opis u poglavlju 13.6.

U paketu `hr.fer.zemris.lsystems` već su dostupna sljedeća sučelja:

```
public interface LSystem {
      public String generate(int level);
      public void draw(int level, Painter painter);
}

public interface LSystemBuilder {
      LSystemBuilder setUnitLength(double unitLength);
      LSystemBuilder setOrigin(double x, double y);
      LSystemBuilder setAngle(double angle);
      LSystemBuilder setAxiom(String axiom);
      LSystemBuilder setUnitLengthDegreeScaler(double unitLengthDegreeScaler);
      LSystemBuilder registerProduction(char symbol, String production);
      LSystemBuilder registerCommand(char symbol, String action);

      LSystemBuilder configureFromText(String[] lines);

      LSystem build();
}

public interface LSystemBuilderProvider {

      LSystemBuilder createLSystemBuilder();

}

public interface Painter {
      void drawLine(
        double x0, double y0, double x1, double y1, Color color, float size
      );
}
```

Sučelje modelira jedan Lindermayerov sustav. Metoda generate prima razinu i vraća string koji odgovara generiranom nizu nakon zadanog broja razina primjena produkcija. Ako je razina 0, vraća se aksiom, za razinu 1 vraća se niz dobiven primjenom produkcija na aksiom, odnosno općenito, ako je razina *k*, vraća se niz dobiven primjenom produkcija na niz dobiven za razinu *k*-1. Metoda draw uporabom primljenog objekta za crtanje linija crta rezultantni fraktal.

Sučelje `LSystemBuilder` modelira objekte koje je moguće konfigurirati i potom pozvati metodu `build()` koja vraća jedan konkretan Lindermayerov sustav prema zadanoj konfiguraciji. Konfiguracija se može napraviti na dva načina: pozivanjem odgovarajućih metoda ili pak metode `configureFromText`.

Sučelje `LSystemBuilderProvider` modelira objekte koji znaju stvarati objekte za konfiguriranje Lindermayerovih sustava.

Sučelje `Painter` modelira objekte kojima je moguće crtati linije.

Evo dva primjera koji pojašnjavaju kako se ovo sve zajedno može koristiti. Oba pretpostavljaju da kao ulaz dobivaju objekt preko kojega mogu stvarati objekte za konfiguriranje Lindermayerovih sustava.

```
private static LSystem createKochCurve(LSystemBuilderProvider provider) {
	return provider.createLSystemBuilder()
			.registerCommand('F', "draw 1")
			.registerCommand('+', "rotate 60")
			.registerCommand('-', "rotate -60")
			.setOrigin(0.05, 0.4)
			.setAngle(0)
			.setUnitLength(0.9)
			.setUnitLengthDegreeScaler(1.0/3.0)
			.registerProduction('F', "F+F--F+F")
			.setAxiom("F")
			.build();
}

private static LSystem createKochCurve2(LSystemBuilderProvider provider) {
	String[] data = new String[] {
			"origin                 0.05 0.4",
			"angle                  0",
			"unitLength             0.9",
			"unitLengthDegreeScaler 1.0 / 3.0",
			"",
			"command F draw 1",
			"command + rotate 60",
			"command - rotate -60",
			"",
			"axiom F",
			"",
			"production F F+F--F+F"
	};
	return provider.createLSystemBuilder().configureFromText(data).build();
}
```

Konfiguracija koja se temelji na nizu linija (posljednji primjer) podrazumijeva da svaka linija sadrži po jednu direktivu (ili je prazna). Direktive se sljedeće:

- origin: zadaje točku iz koje kornjača kreće; nad prozorom u kojem će biti prikazana slika bit će rastegnut virtualni koordinatni sustav čije je ishodište dolje lijevo, plus x-os ide prema desno i desni rub prozora ima x-koordinatu 1, plus y-os ide prema gore i vrh prozora ima y-koordinatu 1; centar prozora je tada (0.5, 0.5)
- angle: kut prema pozitivnoj osi-x smjera u kojem kornjača gleda; kut od 0 stupnjeva znači da kornjača gleda u desno; kut od 90° znači da kornjača gleda prema vrhu ekrana, itd.
- unitLength: koliko je dugačak jedinični pomak kornjače; primjerice, vrijednost 0.5 ako kornjača gleda u desno bi odgovaralo polovici širine prozora
- unitLengthDegreeScaler: kako bi se dimenzije prikazanog fraktala zadržale manje-više konstantnima, ako se generira niz za dubinu d, unitLength je potrebno na odgovarajući način skalirati; stoga je efektivnu duljinu koraka za kornjaču potrebno postaviti na unitLength * (unitLengthDegreeScaler^d)
  vrijednost za ovaj broj može biti jedan decimalni broj, ili pak decimalni broj / decimalni broj (uz proizvoljan broj razmaka između svega što uključuje i nula razmaka)
- command: za simbol koji slijedi definira akciju koju kornjača mora napraviti; moguće akcije popisane su u nastavku; konfiguacija može sadržavati više ovih direktiva ali moraju biti za različite simbole

- axiom: početni niz iz kojeg kreće razvoj sustava; može biti samo jedan simbol ali može biti niz
- production: za simbol definira niz kojim se isti zamjenjuje; konfiguracija može sadržavati više ovih direktiva, ali moraju biti za različite simbole

Akcije koje treba podržati su sljedeće:
- "draw s": pomiče kornjaču u smjeru u kojem je okrenuta za s efektivne duljine pomaka i pri tome ostavlja trag trenutnom bojom; ažurira njezin položaj; primjer: "draw 0.5"
- "skip s": kao draw samo ne ostavlja trag (efekt je kao da je kornjača preskočila taj dio)
- "scale s": efektivnu duljinu pomaka ažurira tako da je pomnoži danim faktorom; npr. "scale 0.75"
- "rotate a": rotira smjer u kojem gleda kornjača za dani kut u stupnjevima (rotacija je u matematički pozitivnom smjeru); npr. "rotate -90"
- "push": trenutno stanje kornjače pohranjuje na stog i kopiju stavlja na vrh
- "pop": sa stoga vraća prethodno pohranjeno stanje kornjače
- "color rrggbb": ažurira boju kojom treba crtati na zadanu; npr. "color ff0000" aktivira crvenu boju

Prilikom obrade tekstovnih zapisa sve višestruke praznine i tabove treba zanemariti.

Vaš je zadatak sljedeći. U paketu `hr.fer.zemris.lsystems.impl` definirajte razred `TurtleState` koji pamti trenutnu poziciju na kojoj se kornjača nalazi (`Vector2D`), smjer u kojem kornjača gleda (`Vector2D`, jedinične duljine!), boju kojom kornjača crta (`Color`) te efektivnu duljinu pomaka (`double`). Definirajte javnu metodu `copy()` koja vraća novi objekt s kopijom trenutnog stanja.

U istom paketu definirajte razred `Context`; primjerci ovog razreda omogućavaju izvođenje postupka prikazivanja fraktala te trebaju ponuditi stog na koji je moguće stavljati i dohvaćati stanja kornjače; za stog koristite Vaš razred `ObjectStack`. Stoga napravite javne metode:

```
TurtleState getCurrentState(); // vraća stanje s vrha stoga bez uklanjanja
void pushState(TurtleState state); // na vrh gura predano stanje
void popState(); // briše jedno stanje s vrha
```

U istom paketu definirajte sučelje `Command`; sučelje treba definirati jednu metodu:

```
void execute(Context ctx, Painter painter);
```

Napravite novi paket `hr.fer.zemris.lsystems.impl.commands`. Svaku od naredbi koje kornjača mora biti u stanju raditi modelirajte jednim razredom smještenim u ovaj paket; razred mora implementirati sučelje `Command`. Napravite tako:

- `PopCommand`: briše jedno stanje s vrha stoga
- `PushCommand`: stanje s vrha stoga kopira i kopiju stavlja na stog
- `RotateCommand(angle)`: kroz konstruktor prima `angle`; u trenutnom stanju modificira vektor smjera gledanja kornjače tako što ga rotira za zadani kut
- `DrawCommand(step)`: kroz konstruktor prima `step`; računa gdje kornjača mora otići; povlači liniju zadanom bojom od trenutne pozicije kornjače do izračunate i pamti u trenutnom stanju novu poziciju kornjače
- `SkipCommand(step)`: kao `DrawCommand` samo što ne povlači liniju
- `ScaleCommand(factor)`: kroz konstruktor prima `factor`; u trenutnom stanju ažurira efektivnu duljinu pomaka skaliranjem s danim faktorom
- `ColorCommand(Color)`: kroz konstruktor prima boju (kao primjerak razreda `java.awt.Color`); u trenutno stanje kornjače zapisuje predanu boju

U prethodnom tekstu pojam "u trenutno stanje" podrazumijeva stanje koje se iz konteksta dobije kao zapisano na vrhu – dakle metodom `Context#getCurrentState()`.

Konačno, u paket `hr.fer.zemris.lsystems.impl` smjestite `LSystemBuilderImpl` koji implementira `LSystemBuilder`. Razred `LSystemBuilderImpl` treba koristiti dva primjerka razreda `Dictionary`: preko jednog treba pamtiti registrirane naredbe; preko drugoga registrirane akcije. Ovaj razred koristi sljedeće defaultne vrijednosti (uz prazne riječnike):

```
unitLength = 0.1;
unitLengthDegreeScaler = 1;
origin = new Vector2D(0, 0);
angle = 0;
axiom = "";
```

U metodi `build()` potrebno je vratiti primjerak Vašeg novog razreda (nekog ugniježđenog). Taj razred u metodi `draw()` stvara novi kontekst, stvara novo stanje kornjače (crna boja je default), gura ga na stog konteksta, potom za zadanu dubinu poziva generiranje konačnog znakovnog niza i na kraju, za svaki znak generiranog niza iz riječnika dohvaća naredbu; ako naredbe nema, prelazi se odmah na sljedeći znak; ako naredba postoji, izvršava je i prelazi na sljedeći znak.

Napravite program `Glavni1` smješten u paket `demo`. U metodu `main` stavite samo naredbu:

```
LSystemViewer.showLSystem(createKochCurve(LSystemBuilderImpl::new));
```

i dodajte metodu koju pozivamo a dana je u ovoj uputi. Ako ste sve dobro napravili, po pokretanju ćete dobiti prozor gdje ćete moći mijenjati stupanj i vidjeti generiranu krivulju.

Napravite program `Glavni2` smješten u paket `demo`. U metodu `main` stavite samo naredbu:

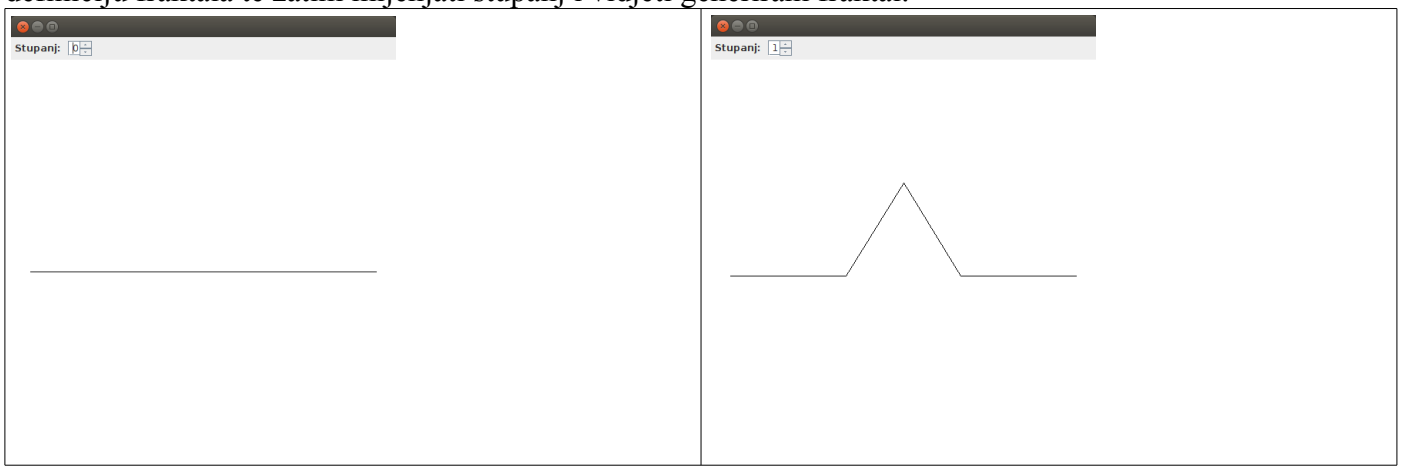
```
LSystemViewer.showLSystem(createKochCurve(LSystemBuilderImpl::new));
```

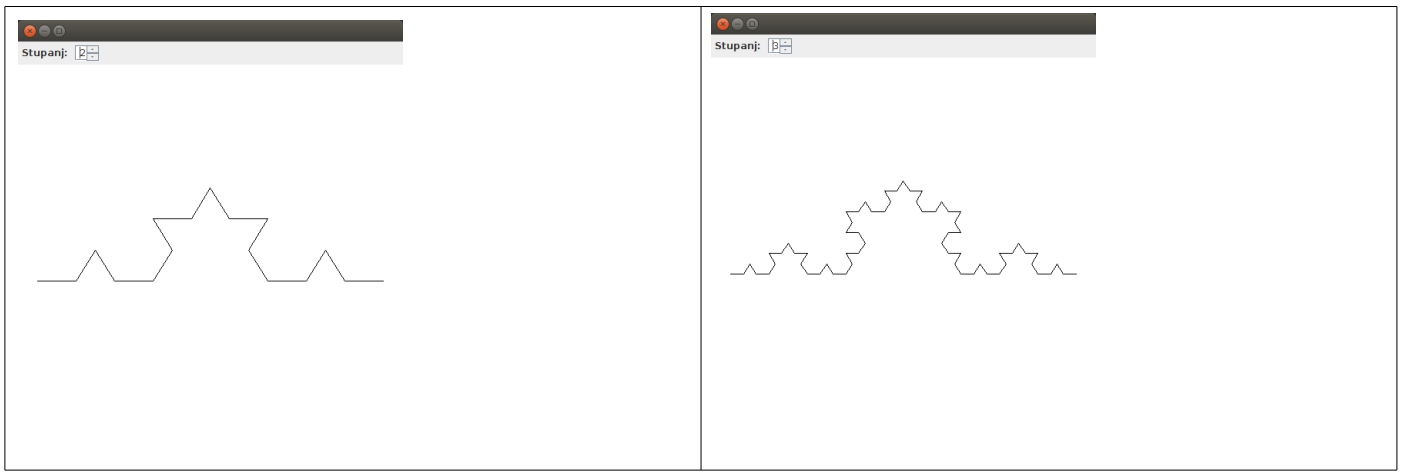
i dodajte metodu koju pozivamo a dana je u ovoj uputi. Ako ste sve dobro napravili, po pokretanju ćete dobiti prozor gdje ćete moći mijenjati stupanj i vidjeti generiranu krivulju.

Napravite program `Glavni3` smješten u paket `demo`. U metodu `main` stavite samo naredbu:

```
LSystemViewer.showLSystem(LSystemBuilderImpl::new);
```

Ako ste sve dobro napravili, po pokretanju ćete dobiti prozor gdje ćete moći iz tekstovne datoteke učitati definiciju fraktala te zatim mijenjati stupanj i vidjeti generirani fraktal.





Ako Vam se širina krivulje bude mijenjala, niste dobro implementirali opisan način izračuna efektivne duljine koraka.

Kao dodatna ZIP arhiva na Ferka je uploadan i niz tekstovnih datoteka s primjerima drugih fraktala koje bi Vaša implementacija sada morala biti u stanju prikazivati: isprobajte ih.

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); the exception are classes from 2nd homework as specified in Problem 1. You can not use any of Java Collection Framework classes which represent collections or its derivatives (its OK to use Arrays class if you find it suitable). Document your code!

The consultations are at standard times (from Tuesday). Feel free to drop by my office after email announcement.

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

When your homework is done, pack it in zip archive with name `hw03-0000000000.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 April 5th 2018. at 07:00 AM.