# 4. homework assignment; JAVA, Academic year 2014/2015; FER

First: read page 10. I mean it! You are back? OK. This homework consists of four problems.

First three problems are modification and extension of problems I have given at this year's course *Object-oriented programming*, so the package shown on class diagrams won't be appropriate for this homework. Instead, place the classes in package `hr.fer.zemris.java.tecaj.hw4.collections`.

## *Problem 1.*

Napišite implementaciju razreda `SimpleHashtable`. Razred predstavlja tablicu raspršenog adresiranja koja omogućava pohranu uređenih parova (ključ, vrijednost). Postoje dva javna konstruktora: defaultni koji stvara tablicu veličine 16 slotova, te konstruktor koji prima jedan argument: broj koji predstavlja željeni početni kapacitet tablice i koji stvara tablicu veličine koja je potencija broja 2 koja je prva veća ili jednaka predanom broju (npr. ako se zada 30, bira se 32); ako je ovaj broj manji od 1, potrebno je baciti `IllegalArgumentException`. Ova implementacija koristit će kao preljevnu politiku pohranu u ulančanu listu. Stoga će broj uređenih parova (ključ, vrijednost) koji su pohranjeni u ovoj kolekciji moći biti veći od broja slotova tablice (pojašnjeno u nastavku).

Jedan slot tablice modelirajte ugniježđenim statičkim razredom `TableEntry` (razmislite zašto želimo da je razred statički i što to znači). Primjerci ovog razreda imaju člansku varijablu `key` u kojoj pamte predani ključ, člansku varijablu `value` u kojoj pamte pridruženu vrijednost te člansku varijablu `next` koja pokazuje na sljedeći primjerak razreda `TableEntry` koji se nalazi u *istom slotu* tablice (izgradnjom ovakve liste rješavat ćete problem preljeva – situacije kada u isti slot treba upisati više uređenih parova). I ključ i vrijednost mogu biti bilo kakvi objekti – ne moraju nužno biti stringovi.

Ideju uporabe ovakve kolekcije ilustrira sljedeći kod.

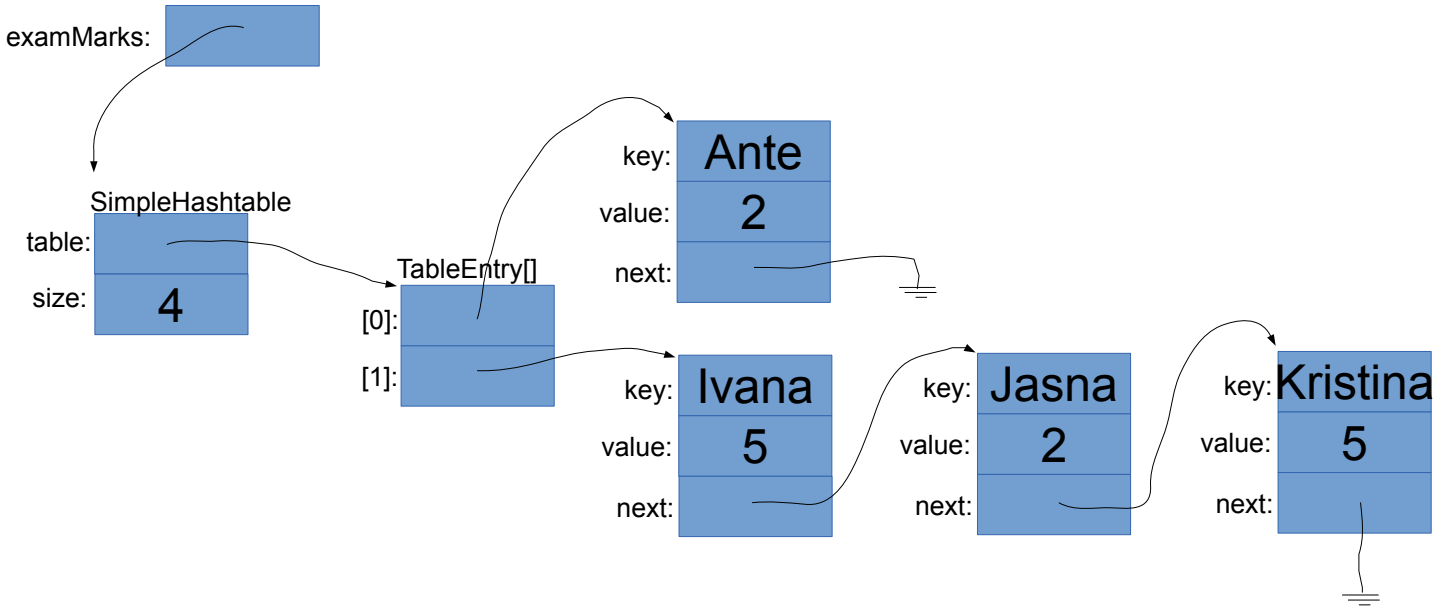
```
// create collection:
SimpleHashtable examMarks = new SimpleHashtable(2);

// fill data:
examMarks.put("Ivana", Integer.valueOf(2));
examMarks.put("Ante", Integer.valueOf(2));
examMarks.put("Jasna", Integer.valueOf(2));
examMarks.put("Kristina", Integer.valueOf(5));
examMarks.put("Ivana", Integer.valueOf(5)); // overwrites old grade for Ivana

// query collection:
Integer kristinaGrade =  (Integer)examMarks.get("Kristina");
System.out.println("Kristina's exam grade is: " + kristinaGrade); // writes: 5

// What is collection's size? Must be four!
System.out.println("Number of stored pairs: " + examMarks.size()); // writes: 4
```

Za potrebe izračuna slota u koji treba ubaciti uređeni par koristite metodu `hashCode()` ključa, pa modulo veličina tablice. Ključ uređenog para ne smije biti `null` dok vrijednost može biti `null`.

Razred `SimpleHashtable` treba imati sljedeće članske varijable:

- `TableEntry[] table`: polje slotova tablice,
- `int size`: broj parova koji su pohranjeni u tablici.

Pojednostavljeni prikaz stanja u memoriji nakon izvođenja koda iz prethodnog primjera ilustriran je na sljedećoj slici.



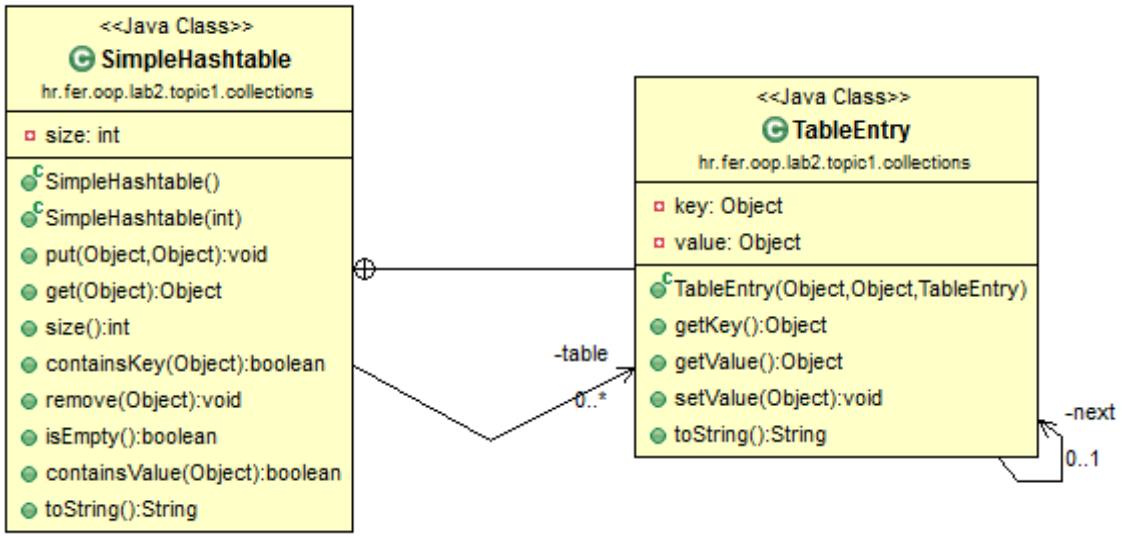

Pri tome na platformi Java 8 vrijedi:

| Objekt | hashCode() | \|hashCode()\| | slot=\|hashCode()\| % 2 |
|---|---|---|---|
| "Ivana" | 71029095 | 71029095 | 1 |
| "Ante" | 2045822 | 2045822 | 0 |
| "Jasna" | 71344303 | 71344303 | 1 |
| "Kristina" | -1221180583 | 1221180583 | 1 |

Stoga će ključevi *Ivana*, *Jasna* i *Kristina* biti u slotu 1 a ključ *Ante* u slotu 0. Razmislite odgovara li prikazana slika stvarnom stanju u memoriji ili bismo za stvarno stanje dio slike trebali drugačije nacrtati?

Dijagram razreda koji prikazuje razrede ovog zadatka prikazan je u nastavku.

Metode i




konstruktori koje razred `SimpleHashtable` mora ponuditi navedeni su u nastavku ove upute bez posebne dokumentacije (iz imena bi moralo biti jasno što se od metode očekuje).

```
public SimpleHashtable();
public SimpleHashtable(int capacity);
public void put(Object key, Object value);
public Object get(Object key);
public int size();
public boolean containsKey(Object key);
public boolean containsValue(Object value);
public void remove(Object key);
public boolean isEmpty();
public String toString();
```

Metoda `put` pozvana s ključem koji u tablici već postoji ažurira postojeći par novom vrijednošću; metoda ne dodaje još jedan par s istim ključem ali drugom vrijednosti. Ako se kao ključ preda `null`, metoda mora baciti `IllegalArgumentException`. Ako se zapis dodaje, onda se u listu dodaje na njen kraj. Za potrebe usporedbe jesu li dva ključa ista koristite metodu `equals(other)` nad ključevima.

Metoda `get` pozvana s ključem koji u tablici ne postoji vraća `null`. Ovdje je legalno kao argument predati `null` jer takav ključ doista ne postoji.

Metoda `remove` uklanja iz tablice uređeni par sa zadanim ključem, ako takav postoji (inače ne radi ništa).

Što možete zaključiti o složenosti metode `containsKey` a što o složenosti metode `containsValue` u ovako implementiranoj kolekciji (uz pretpostavku da je broj parova dodanih u tablicu dosta manji od broja slotova tablice te da funkcija sažetka radi dobro raspršenje)?

Implementirajte metodu `toString()` tako da generira popis uređenih parova koji su pohranjeni u kolekciji i koji je formatiran kako je prikazano u nastavku:

```
"[key1=value1, key2=value2, key3=value3]".
```

Uređeni parovi moraju biti prikazani redoslijedom koji se nalaze i tablici (od slota 0 prema dnu; u listi od prvog čvora prema zadnjem).

## *Problem 2.*

Modificirajte prethodno razvijenu kolekciju tako da prati popunjenost. Naime, poznato je da tablice raspršenog adresiranja nude povoljne računske kompleksnosti samo ako nisu prepunjene (odnosno ako nema previše preljeva). Stoga pratite u kodu kada popunjenost tablice postane jednaka ili veća od 75% broja slotova, i u tom trenutku povećajte kapacitet tablice na dvostruki. Obratite pažnju da ćete tada iz "stare" tablice morati izvaditi sve postojeće parove i nanovo iz ubaciti u veću tablicu (jer će se promijeniti adrese u kojima ih očekujete).

Dodajte metodu:

```
public void clear();
```

čija je zadaća izbrisati sve uređene parove iz kolekcije. Ova metoda ne mijenja kapacitet same tablice.

## *Problem 3.*

Modificirajte razred `SimpleHashtable` tako da definirate da razred implementira sučelje `Iterable<SimpleHashtable.TableEntry>`, kako je prikazano u nastavku. Ovo prikazano u zagradama `<X>` predstavlja uporabu tehnologije *Java Generics*. Obradit ćemo je kasnije, a za potrebe ove zadaće sve potrebno napisano je u ovoj uputi.

```
public class SimpleHashtable implements Iterable<SimpleHashtable.TableEntry> { … }
```

Zbog ove promjene u razred ćete morati dodati metodu tvornicu koja će proizvoditi iteratore koji se mogu koristiti za obilazak po svim parovima koji su trenutno pohranjeni u tablici, i to redoslijedom kojim se nalaze u tablici ako se tablica prolazi od slota 0:

```
Iterator<SimpleHashtable.TableEntry> iterator() { … }
```

Ideja je osigurati da možete napisati sljedeći isječak koda.

```
public class Primjer {

	public static void main(String[] args) {
		// create collection:
		SimpleHashtable examMarks = new SimpleHashtable(2);

		// fill data:
		examMarks.put("Ivana", Integer.valueOf(2));
		examMarks.put("Ante", Integer.valueOf(2));
		examMarks.put("Jasna", Integer.valueOf(2));
		examMarks.put("Kristina", Integer.valueOf(5));
		examMarks.put("Ivana", Integer.valueOf(5)); // overwrites old grade for Ivana

		for(SimpleHashtable.TableEntry pair : examMarks) {
			System.out.printf("%s => %d%n", pair.getKey(), pair.getValue());
		}

	}
}
```

Ovaj kod trebao bi rezultirati sljedećim ispisom:

```
Ante => 2
Ivana => 5
Jasna => 2
Kristina => 5
```

Kod prikazan u nastavku također bi morao raditi i ispisati kartezijev produkt uređenih parova:

```
for(SimpleHashtable.TableEntry pair1 : examMarks) {
	for(SimpleHashtable.TableEntry pair2 : examMarks) {
		System.out.printf(
			"(%s => %d) - (%s => %d)%n",
			pair1.getKey(), pair1.getValue(),
			pair2.getKey(), pair2.getValue(),
		);
	}
}
```

Razred koji ostvaruje iterator modelirajte kao ugniježđeni razred razreda `SimpleHashtable`:

```
private class IteratorImpl implements Iterator<SimpleHashtable.TableEntry> {
	boolean hasNext() { … }
	SimpleHashtable.TableEntry next() { … }
	void remove() { … }
}
```

Napisani iterator mora pri tome raditi direktno nad tablicom raspršenog adresiranja. Nije dopušteno da pri stvaranju (ili u bilo kojem trenutku kasnije) iterator prekopira sadržaj tablice u neku drugu strukturu pa radi nad njom.

Vaša implementacija Iteratora mora podržati i operaciju `remove()` čijim se pozivom iz tablice briše trenutni element (onaj koji je vraćen posljednjim pozivom metode `next()`). Uočite da je tu metodu dozvoljeno pozvati samo jednom nakon poziva metode `next()`. Pogledajte dokumentaciju:

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html#remove--

Obratite također pažnju da korisnik *ne mora* uopće pozivati metodu `hasNext()` pri iteriranju. Umjesto toga, može pozivati samo `next()` i čekati na `NoSuchElementException` koji označava da je iteriranje gotovo (takav se način koristi kao uobičajeni u Pythonu).

**Modifikacije kolekcije dok traje iteriranje**

S obzirom da je efikasan algoritam iteriranja dosta teško (a ponekad i nemoguće) ostvariti ako se dozvoli da korisnik izvana modificira kolekciju, iteratore najčešće pišemo tako da svoj posao obavljaju sve dok ne uoče da je kolekcija modificirana; kada to utvrde, svi pozivi metoda iteratora bacaju iznimku `ConcurrentModificationException` čime odbijaju daljnje iteriranje.

U sljedećem primjeru, iz kolekcije se uklanja ocjena za Ivanu na korektan način (nema iznimke).

```
Iterator<SimpleHashtable.TableEntry> iter = examMarks.iterator();
while(iter.hasNext()) {
	SimpleHashtable.TableEntry pair = iter.next();
	if(pair.getKey().equals("Ivana")) {
		iter.remove();
	}
}
```

Sljedeći kod bacio bi `IllegalStateException` jer se uklanjanje poziva više od jednom za trenutni par nad kojim je iterator (to bi bacio drugi poziv metode `remove`):

```
Iterator<SimpleHashtable.TableEntry> iter = examMarks.iterator();
while(iter.hasNext()) {
	SimpleHashtable.TableEntry pair = iter.next();
	if(pair.getKey().equals("Ivana")) {
		iter.remove();
		iter.remove();
	}
}
```

Sljedeći kod bacio bi `ConcurrentModificationException` jer se uklanjanje poziva "izvana" (direktno nad kolekcijom a ne kroz iterator koji to može obaviti kontrolirano i ažurirati svoje interne podatke). Iznimku bi bacila metoda `hasNext` jer je ona prva koja se u prikazanom primjeru poziva nakon brisanja.

```
Iterator<SimpleHashtable.TableEntry> iter = examMarks.iterator();
while(iter.hasNext()) {
	SimpleHashtable.TableEntry pair = iter.next();
	if(pair.getKey().equals("Ivana")) {
		examMarks.remove("Ivana");
	}
}
```

Kako implementirati ovakvo ponašanje? Evo ideje. Opremite razred `SimpleHashtable` još jednom privatnom članskom varijablom: `modificationCount`. Svaka metoda razreda `SimpleHashtable` koja na bilo koji način mijenja sadržaj kolekcije (dodavanje, brisanje, promjena veličine tablice) treba ovaj brojač povećati za jedan. Svaki iterator pri stvaranju mora zapamtiti koju je vrijednost imao taj brojač. Pri svakom pozivu bilo koje od metoda iteratora, iterator najprije uspoređuje zapamćenu vrijednost brojača s trenutnom vrijednosti brojača; ako utvrdi da se vrijednosti ne podudaraju, baca iznimku. Prilikom implementacije metode `remove` u iteratoru pripazite da ažurirate zapamćenu vrijednost (čak i kada iterator radi brisanje, on mora povećati `modificationCount`).

## *Problem 4.*

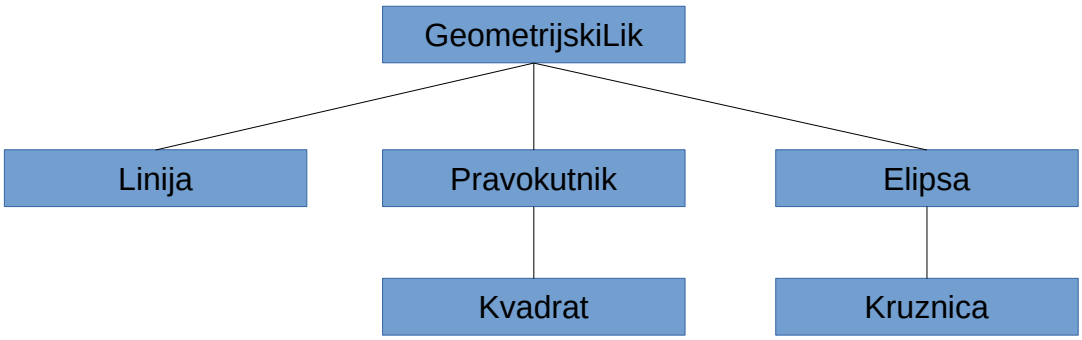
Rješenje ovog zadatka smjestite u paket `hr.fer.zemris.java.tecaj.hw4.grafika`. Svi razredi u ovom zadatku definirani su na hrvatskom, pa stoga tako ostvarite i rješenje (imena varijabli; dokumentacija).

Uzmite s Ferka arhivu prikaz.jar u kojoj se nalazi razred `Slika` koji predstavlja implementaciju crno-bijele rasterske slike, kao i razred koji sliku prikazuje u zasebnom prozoru.

Krećemo od osnovnog *apstraktnog* razreda `GeometrijskiLik` koji ima sljedeće metode:

```
public abstract boolean sadrziTocku(int x, int y);
public void popuniLik(Slika slika);
```

Napravite sljedeće stablo geometrijskih likova.




Linija je zadana početnom i završnom točkom. Pravokutnik je zadan koordinatama gornjeg-lijevog ugla, širinom i visinom. Kvadrat je zadan koordinatama gornjeg-lijevog ugla te duljinom stranice. Elipsa je zadana koordinatama centra, vodoravnim te okomitim radijusom. Kružnica je zadana koordinatama centra te radijusom. Za svaki od likova (gdje to ima smisla) definirajte efikasniju metodu popunjavanja lika na slici od one koja je dana u apstraktnom razredu GeometrijskiLik.

Definirajte sučelje `StvarateljLika` s jednom metodom:

```
String nazivLika();
GeometrijskiLik stvoriIzStringa(String parametri);
```

Opremite svaki od razreda konkretnih likova (ne u apstraktnim razredima) privatnim statičkim ugniježđenim razredom koji implementira ovo sučelje.

Primjerice, u razredu `Linija` takav će ugniježđeni razred imati metodu nazivLika koja će vraćati "`LINIJA`" i metodu stvoriIzStringa koja će primiti "10 20 50 70" i stvoriti i vratiti novu liniju koja počinje na (10,20) a završava na (50,70). Redosljed ovih parametara opisan je na dnu prethodne stranice.

Opremite svaki od razreda konkretnih geometrijskih likova i **statičkom konstantom** imena `STVARATELJ` koja je referenca na jedan primjerak prethodno definiranog ugniježđenog razreda. Ovime ćete imati strukturu razreda koja je poprilici sljedeća.

```
class LikX extends Nadrazred {

   ...

   public static final StvarateljLika STVARATELJ = new LikXStvaratelj();

   private static class LikXStvaratelj implements StvarateljLika {
            @Override
            public String nazivLika() {
                  return "LIKX";
            }

            public GeometrijskiLik stvoriIzStringa(String parametri) {
                  return new LikX(isjeckani parametri pravog tipa);
            };
  }

}
```

Sada pripredite glavni program, razred `hr.fer.zemris.java.tecaj.hw4.grafika.demo.Crtalo`. Njegova struktura je prikazana u nastavku (potrebno prebaciti u Javu, dodati provjere pogrešaka, konverzije tipova argumenata, ...).

```
public class Crtalo {

      public static void main(String[] args) {
            SimpleHashtable stvaratelji = podesi(Linija.class, Pravokutnik.class);
            String[] definicije = Files.readAllLines(
                  Paths.get(args[0]), StandardCharsets.UTF_8).toArray(new String[0]);
            GeometrijskiLik[] likovi = new GeometrijskiLik[definicije.length];
            za svaku definiciju d {
                  String lik = ocitaj lik
                  String parametri = ostatak retka
                  StvarateljLika stvaratelj = (StvarateljLika)stvaratelji.get(lik);
                  likovi[index++] = stvaratelj.stvoriIzStringa(parametri);
            }

            Slika slika = new Slika(args[1], args[2]);
            za svaki lik : likovi {
                  nacrtaj lik na slici
            }
            prikazi sliku
      }
```

```
	private static SimpleHashtable podesi(Class<?> ... razredi) {
		SimpleHashtable stvaratelji = new SimpleHashtable();
		for(Class<?> razred : razredi) {
			try {
				Field field = razred.getDeclaredField("STVARATELJ");
				StvarateljLika stvaratelj = (StvarateljLika)field.get(null);
				stvaratelji.put(stvaratelj.nazivLika(), stvaratelj);
			} catch(Exception ex) {
				throw new RuntimeException(
					"Nije moguće doći do stvaratelja za razred "+
					razred.getName()+".", ex);
			}
		}
		return stvaratelji;
	}

}
```

Ideja rješenja je sljedeća. U glavni program dodat ćemo metodu tvornicu koja će znati "proizvoditi" geometrijske likove temeljem njihovog imena i parametara (ovo ćemo čitati iz datoteke). Stoga ćemo iskoristiti razred `SimpleHashtable` koji ste prethodno napisali kako bismo u njega pohranili uređene parove (naziv lika, stvaratelj lika) za sve likove koje podržavamo. Za popunjavanje ove kolekcije zadužena je metoda podesi, koja prima varijabilan broj argumenata: popis razreda iz kojih treba dohvatiti stvaratelje i upisati ih u kolekciju. Za svaki predani razred, koristeći Java Reflection API pitamo javin virtualni stroj da nam pronađe člansku varijablu `STVARATELJ` i potom dohvaćamo vrijednost koja je upisana u toj članskoj varijabli. Ako predani razred nema takve članske varijable ili ako se dogodi pogreška bilo kojeg drugog tipa, bit će izazvana iznimka.

U metodi `main` zatražimo stvaranje takve kolekcije (metodi podesi pošaljemo sve razrede koje podržavamo). Općenitije rješenje bi bilo da se dogovorimo o nekim konvencijama (tipa: svi razredi koji predstavljaju slike moraju biti u nekom paketu i imati ime nekog specifičnog formata), pa pri pokretanju automatski utvrdimo što sve zadovoljava te kriterije. Međutim, za potrebe ove zadaće dovoljno je imati ovakvu generičku metodu i poslati joj popis razreda koje treba istražiti.

Jednom kada imamo kolekciju stvaratelja, trčimo po retcima s likovima, vadimo ime lika, iz kolekcije dohvaćamo objekt koji je stvaratelj za taj lik i pozivamo njegovu metodu tvornicu s ostatkom parametara. Lik spremamo u pomoćno polje, i kad smo gotovi, crtamo sve likove i prikazujemo sliku.

Metoda `stvoriIzStringa(String parametri)` u slučaju da postoji problem s parametrima mora baciti `IllegalArgumentException`. Glavni program to mora uhvatiti, korisniku ispisati poruku o pogrešci (te redak koji je uzrokovao tu pogrešku), napomenu da slika neće biti iscrtana, i završiti s radom (bez prikaza slike).

Primjer pokretanja programa:

```
java -cp sveStoTreba hr.fer.zemris.java.tecaj.hw4.grafika.demo.Crtalo dat.txt 400 400
```

Sadržaj *ispravne* datoteke dat.txt:

```
KVADRAT 0 2 10 10
PRAVOKUTNIK 11 2 10 20
LINIJA 399 0 0 399
LINIJA 399 0 399 399
LINIJA 399 0 0 0
KVADRAT 390 390 20
KRUG 200 40 30
ELIPSA 200 300 180 80
```

Parametri programa su: ime datoteke, širina slike i visina slike. Program crta samo onaj dio svakog lika koji se doista vidi na slici.

U konačnoj verziji programa, metodi `podesi` morate predati sve konkretne razrede koji su zadani u ovom zadatku.

**Important notes**

Solve all of the problems in a single Eclipse 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 all of the classes developed in problems 1, 2 and 3.

You must equip your project with `build.xml` script so that the project can be build from the command line. The script must support the same targets as defined in previous homework assignment. If you don't tell the ant to use `prikaz.jar` in classpath while compiling sources with javac, compilation will fail since the compiler won't be able to obtain needed classes. Learn how this can be fixed. The `prikaz.jar` should be placed in `lib` subdirectory.

All of the classes in all problems 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. You are not allowed to use any of Java Collection Framework classes or its derivatives, except the classes and interfaces explicitly given in this homework (i.e. `Iterable`, `Iterator` are allowed; you can additionally use class `Arrays` if you need 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) `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 11th 2015. at 7 AM.