ᘐᘁᘁᘃᘐᘂᘓᘃᘐᘐ ᑲᓂᓭᘐᘐᘐᓂᓂᓂᘐᘃᘃᘃᘐ

ᘐᘐ ᒪᑲᘐᘐᓭᑲᓂᓂᓭᑲᓂᘐᘐᘃᘐᘓ ᘔ ᘃᘐᓂᓂᘃᘐ

ᗰᓂᑲᘔᘐ ᘓᘓᓂᓭᓂᑲ

Otvoreno računarstvo

OO programiranje i Java

Mario Žagar

Za domaću zadaću:

- Proučiti jezik Javu što više možete:
- Linkovi (korisni i potrebni za lab. vježbe):
  - **Java Tutorial** u HTMLHelp (.chm) formatu:
    http://www.allimant.org/javadoc/index.php
  - **J2SE 6 dokumentacija** u istom formatu (ista stranica):
    http://www.allimant.org/javadoc/index.php
- Linkovi (korisni):
  - Razvojni alat JDK1.6.0 (i JRE1.6.0)
    http://java.sun.com
  - Knjiga "Thinking in Java" (u eletroničkom obliku):
    http://www.mindview.net/Books/TIJ/
  - I sve ostalo što dođe pod ruku :-)

Strukture

- u C-u, ako trebamo složeni tip podatka, koristimo strukture:

```
typedef struct _pravokutnik
{
        int a;
        int b;
        char *opis;
} pravokutnik;
```

- Problem: svatko može promijeniti vrijednost člana strukture (npr. postaviti negativnu duljinu stranice)
  - Rješenje: napisati posebnu funkciju za postavljanje vrijednosti, koja će prvo provjeriti vrijednost
    - Kako osigurati da svi koriste tu funkciju, tj. ne pristupaju članu strukture direktno?

Strukture

- Problem: kako specijalizirati strukturu
  - npr. kvadrat je poseban slučaj pravokutnika, s dvije iste stranice?
- Struktura ista pravokutniku + dodatno svojstvo
  - kako to ostvariti strukturama?

Objektno orijentirano programiranje

- Drugačiji način razmišljanja

  strukture podataka i algoritmi <-> objekti

- **OBJEKT** – skup varijabli i s njima povezanih funkcija
  - oponaša objekt iz stvarnog svijeta
  - pamti stanje i modelira ponašanje
- **RAZRED, KLASA** – nacrt po kojem se stvara objekt
  - ima članske varijable i svojstvene funkcije (metode), postoji kontrola pristupa varijablama i funkcijama
- OBJEKT = primjerak, instanca razreda

Objektno orijentirano programiranje

- Principi objektno orijentiranog programiranja (OOP):
  - apstrakcija (*abstraction*),
  - enkapsulacija (*encapsulation*), (učahurivanje, zatvaranje)
  - nasljeđivanje (*inheritance*),
  - polimorfizam (*polymorphism*) (višeobličje)

Apstrakcija

- Primjer automobila:
  - država - kupovina, carina, PDV, registracija, brojevi, vlasništvo, porezi,..
  - vlasnik - potrošnja, servisi, pranje,..
  - mehaničar - vlasnik, dijelovi, kilometri, cijena rada,..
- sve su to primjeri **apstrakcije** podataka

Apstrakcija

- **apstrakcija** - proces odbacivanja nepotrebnih detalja o objektu, a dodavanje (zadržavanje) onih podataka koji ga prikladno opisuju
- u klasičnim pristupima (imperativni), je to opisano strukturama

Enkapsulacija

- Korak dalje od apstrakcije je prepoznavanje važnosti operacija nad podacima
- Podaci i operacije nad njima postaju cjelina
- Cilj je skrivanje implementacije (korisniku se daje samo sučelje)
- I u klasičnim jezicima prevodilac se buni ako npr. mora rotirati broj definiran kao *float* (pomični zarez)

Enkapsulacija

- Određene operacije nad određenim podacima
- Podaci i operacije (metode, postupci - *methods*) zajedno čine razred, klasu (*class*)
- Varijablu tipa određenog razreda zovemo objekt (Golf - auto) ili instanca (**utjelovljenje, pojava**) (*instance*)

Enkapsulacija

- Slično i u klasici (*int x; x=5;*) osim što su funkcije (metode) vezane uz podatke
- Metode = funkcije koje pripadaju razredu (ime iz *Smalltalk-72*) (plan, put, shema, **postupak**)
- OOP bi se trebalo zvati **Programiranje temeljeno na razredima** (*Class-Based Programming - CBS*)
- primjer (tko zna što ovdje piše?):

  DbrVmdn!a a ao

  Dobar Vam dan!

Nasljeđivanje

- Nasljeđivanje je proširivanje razreda (klase) koji je već definiran,
- Preuzimanje iz “prošlosti”
  - npr: Automobil ima upravljač. Golf je automobil s četiri kotača. (Golf naravno ima upravljač iako to nije posebno naglašeno)
- Pogodno za dodavanje svojstava (specifičnosti, detalja, novih inačica)
- Java i C# ne dozvoljavaju višestruko nasljeđivanje, umjesto toga implementiraju se sučelja (*interface*)

Polimorfizam

- Višeobličje, dijeljenje imena
- Isto ime za različite postupke (metode)
- Dva tipa - *overloading* i *overriding*
- ***Overloading* - preopterećivanje**
  - statičko rješavanje
  - višeznačnost - unutar razreda isto ime za različite postupke uz različiti broj parametara

Polimorfizam

- ***Overriding* - nadjačavanje**
- Pravi polimorfizam
- Dinamičko rješavanje
  - metoda unutar podklase s istim imenom i "**potpisom**" važnija je od metode iz nadređene klase
    - limun iz porodice voće, guli se drugačije od ostalog voća!!
  - ne može se predvidjeti u fazi prevođenja već izvođenja

Zadatak

- Napisati program koji računa površine raznih geometrijskih likova
- Zajednička karakteristika svih geometrijskih likova je da sami "znaju" izračunati svoju površinu
- Likovi sadrže podatke koji ih opisuju i karakteristični su za svaki lik posebno.

Zadatak

- Program treba iz datoteke pročitati oznaku tipa lika i stvoriti objekt odgovarajućeg razreda (klase) te mu prenijeti parametre lika zapisane u liniji iza oznake tipa lika.
- Nakon što se svi podaci iz datoteke pročitaju i stvore objekti odgovarajućeg razreda, redom se ispisuju opisi likova (tip i parametri), njihove površine te zbroj površina koji zauzimaju svi stvoreni likovi.

Zadatak - ulaz

Ulazna datoteka:

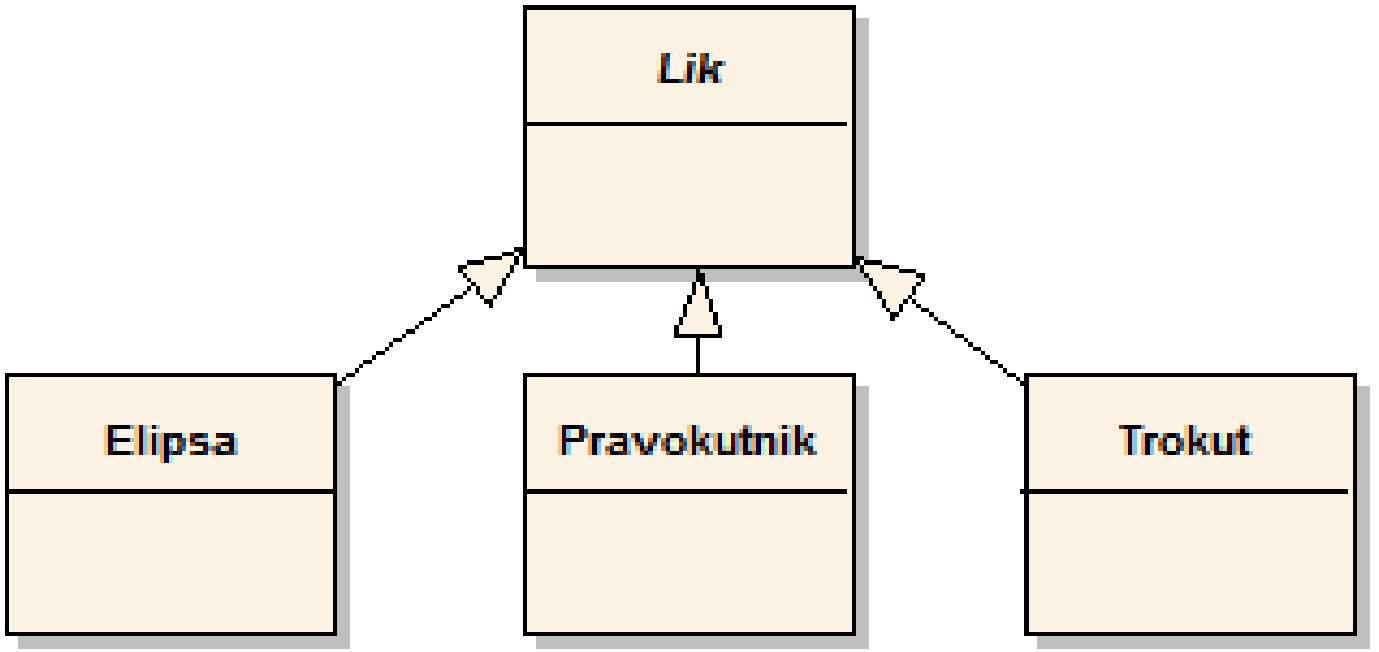
```
K
20
P
10 40
C
40
E
50 30
T
30 40 50
J
20
K
20
C
10
P
5 8
K
5
```

Objekt	Oznaka vrste
pravokutnik	P
kvadrat	K
trokut	T
jednakostraničan trokut	J
elipsa	E
krug	C

Objekt	Podaci u datoteci	Primjer
pravokutnik	stranice a i b	10 20
kvadrat	stranica a	10
trokut	stranice a, b, c	10 20 30
jednakostraničan trokut	stranica a	10
elipsa	radijusi r1 i r2	10 20
krug	radijus r	10

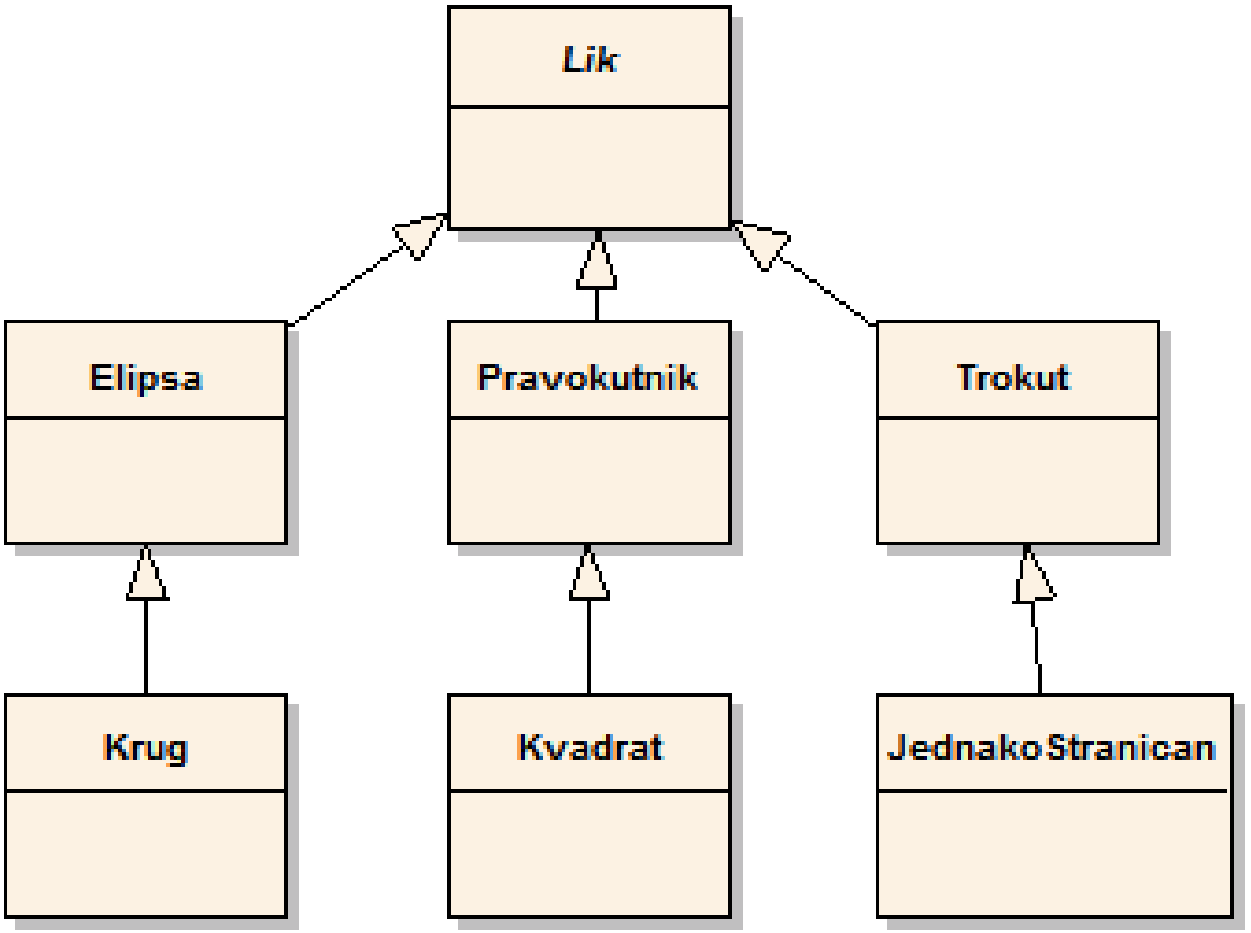
Prvo …

- Definirajmo osnovni apstraktni razred Lik
- Nasljeđuju ga objekti koji predstavljaju geometrijske likove (Elipsa, Pravokutnik, Trokut):



Nasljeđivanje

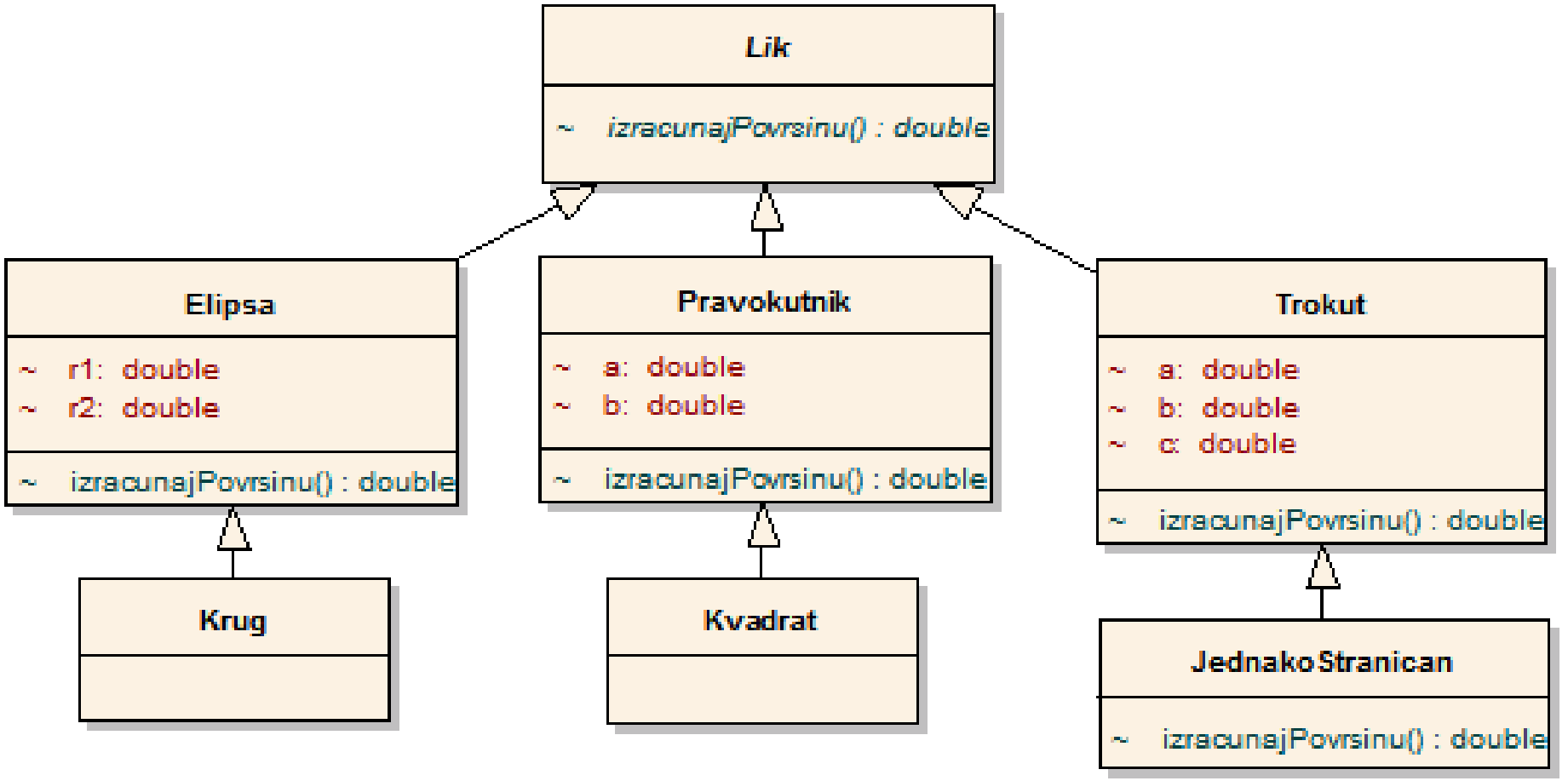
- Posebni slučajevi (kvadrat je posebni slučaj pravokutnika, jednakostraničan trokut trokuta, krug elipse):



Enkapsulacija

- Svaki lik mora znati izračunati vlastitu površinu
- Svaka površina se računa drugačije
- -> računanje površine je zajedničko ponašanje -> mora biti opisano osnovnim razredom -> metodom *Lik.izracunajPovrsinu()*

Geometrijski lik	Formula
Pravokutnik	a*b
Kvadrat	a*a
Trokut	sqrt[s(s-a)(s-b)(s-c)], s = (a+b+c)/2 (Heronova formula)
Jednakostraničan trokut	a*a*sqrt(3)/4
Elipsa	r1*r2*pi
Krug	r*r*pi


Lik
~ izracunajPovrsinu() : double
Elipsa
~ r1: double
~ r2: double
~ izracunajPovrsinu() : double
Pravokutnik
~ a: double
~ b: double
~ izracunajPovrsinu() : double
Trokut
~ a: double
~ b: double
~ c: double
~ izracunajPovrsinu() : double
Krug
Kvadrat
JednakoStranican
~ izracunajPovrsinu() : double

Opis razreda

- *"redom se ispisuju opisi likova (tip i parametri),"*
- osim računanja površine, svaki lik mora imati i opis
- Java – osnovni razred (*java.lang.Object*) ima metodu *toString()*, koju treba nadjačati:

toString

```
public String toString()
```

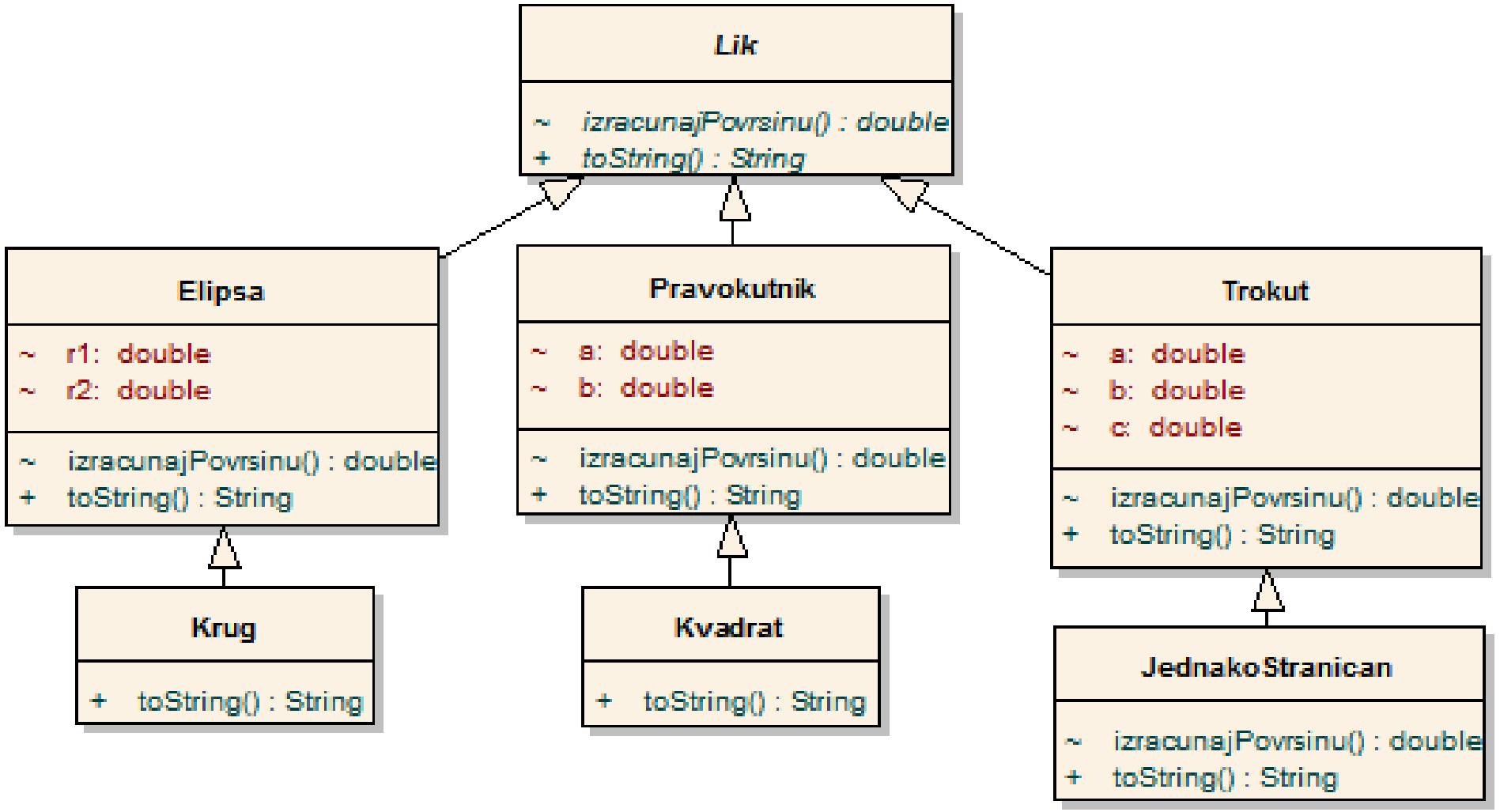
Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The `toString` method for class `Object` returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of:

```
getClass().getName() + '@' + Integer.toHexString(hashCode())
```

Returns:

a string representation of the object.


Lik
~ izracunajPovrsinu() : double
+ toString() : String
Elipsa
~ r1: double
~ r2: double
~ izracunajPovrsinu() : double
+ toString() : String
Pravokutnik
~ a: double
~ b: double
~ izracunajPovrsinu() : double
+ toString() : String
Trokut
~ a: double
~ b: double
~ c: double
~ izracunajPovrsinu() : double
+ toString() : String
Krug
+ toString() : String
Kvadrat
+ toString() : String
JednakoStranican
~ izracunajPovrsinu() : double
+ toString() : String

Kôd, pokažite kôd ...

Lik.java

```
import java.util.*;
import java.io.*;
public abstract class Lik
{
        abstract double izracunajPovrsinu();
        ...
}
```

Pravokutnik.java

```
import java.util.*;
public class Pravokutnik extends Lik
{
        double a,b;
        ...
        double izracunajPovrsinu()
        {
                return a*b;
        }
        public String toString()
        {
                return "Pravokutnik, a="+a+", b="+b+"\t");
        }
}
```

Trokut.java

```
import java.util.*;
import java.lang.Math;
public class Trokut extends Lik{
        double a,b,c;
        double izracunajPovrsinu(){
                double s=(a+b+c)/2;
                return Math.sqrt(s*(s-a)*(s-b)*(s-c));
        }
        public String toString(){
                return "Trokut, a="+a+",b="+b+",c="+c+ "\t";
        }
}
```

Elipsa.java

```
import java.lang.Math;
import java.util.*;
public class Elipsa extends Lik{
        double r1,r2;
        double izracunajPovrsinu(){
                return r1*r2*Math.PI;
        }
        public String toString(){
                return "Elipsa, r1="+r1+", r2="+r2+"\t";
        }
}
```

Stvaranje objekata

- Konstruktor:
  - posebna metoda za stvaranje objekta
  - istog naziva kao i klasa
  - nema povratnog tipa
- Svaki Javin razred implicitno nasljeđuje razred *java.lang.Object*
- *java.lang.Object* definira konstruktor bez parametara
  - svaki Javin razred ima podrazumijevani konstruktor, bez parametara
  - nije ga potrebno navoditi

Konstruktor

- Svaki Javin razred može definirati dodatne konstruktore, s različitim potpisom:

```
public class Pravokutnik extends Lik
{
        double a,b;
        public Pravokutnik()
        {
        //podrazumijevani konstruktor (samo za primjer)
              a=1; b=2;
        }
        public Pravokutnik(String redak)
        {
            Vector args=new Vector();
            args=parsiraj(redak,2);
            this.a=((Double)(args.get(0))).doubleValue();
            this.b=((Double)(args.get(1))).doubleValue();
        }
        ...
}
```

Konstruktor

- Poziv konstruktora:

```
Pravokutnik pk1=new Pravokutnik();
Pravokutnik pk2=new Pravokutnik("3 4");
```

- Operator *new:*
  - alocira memorijski prostor za novi objekt
  - poziva konstruktor
  - vraća referencu na novostvoreni objekt

Zadatak, dodatno

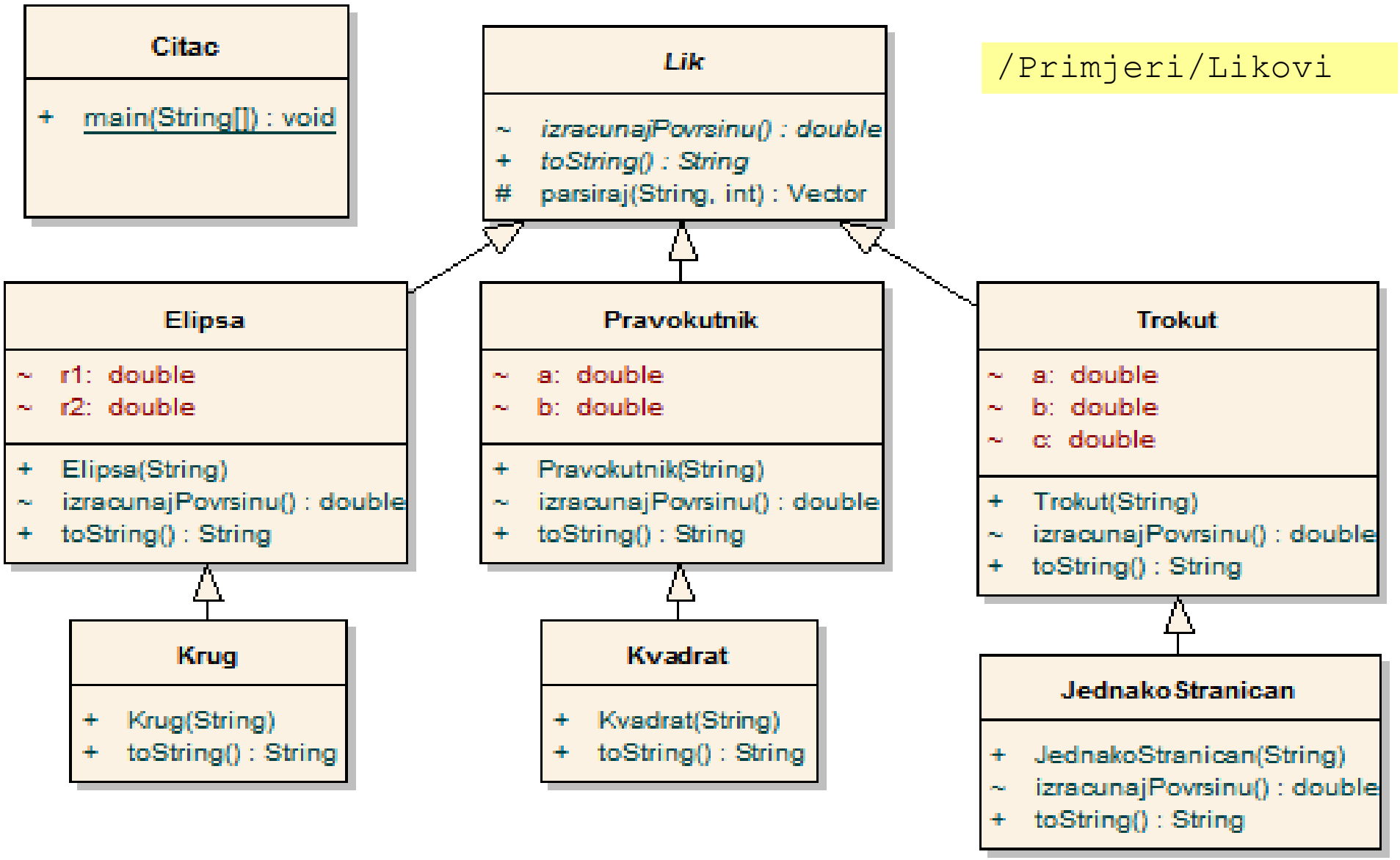
- Konstruktor svakog razreda mora omogućiti unos parametara objekta preko varijable tipa *java.lang.String*
- Unose se podaci učitani iz datoteke – duljine stranica odvojene razmakom
  - npr. konstruktor razreda Trokut prima niz od tri broja odvojena prazninama, stranice a, b, c: 10 20 30

Zadatak, dodatno

- Glavna metoda programa treba otvoriti datoteku čije je ime zadano kao parametar komandne linije, stvoriti vektor (koristiti razred *java.util.Vector*) koji će sadržavati sve likove i popuniti ga objektima koji reprezentiraju geometrijske likove.
- Za svaki lik pročitati redak sa zapisom vrste lika i redak s parametrima, stvoriti odgovarajući objekt i prenijeti mu varijablu tipa *java.lang.String* s parametrima lika.

Zadatak, dodatno

- Kad su svi podaci uneseni, ispisuje se opis i površina svakog pojedinog lika te ukupna površina svih likova u datoteci. Ove zadatke obaviti slijednom obradom svih objekata pohranjenih u vektoru objekata


Citac
+ main(String[]) : void
Lik
~ izracunajPovrsinu() : double
+ toString() : String
# parsiraj(String, int) : Vector
/Primjeri/Likovi
Elipsa
~ r1: double
~ r2: double
+ Elipsa(String)
~ izracunajPovrsinu() : double
+ toString() : String
Pravokutnik
~ a: double
~ b: double
+ Pravokutnik(String)
~ izracunajPovrsinu() : double
+ toString() : String
Trokut
~ a: double
~ b: double
~ c: double
+ Trokut(String)
~ izracunajPovrsinu() : double
+ toString() : String
Krug
+ Krug(String)
+ toString() : String
Kvadrat
+ Kvadrat(String)
+ toString() : String
JednakoStranican
+ JednakoStranican(String)
~ izracunajPovrsinu() : double
+ toString() : String

Metoda *main*

- Definirat ćemo novi razred, koji će sadržavati samo metodu main:

```
public class Citac
{
    public static void main(String [] args)
    {
        ...
    }
}
```

- Metoda main:
  - poziva se prilikom pokretanja programa
  - prima argumente iz komandne linije preko polja objekata tipa String (*args* u gornjem primjeru)

Još kôda

- Učitavamo podatke i instanciramo objekte:

```
public class Citac {
    public static void main(String [] args) {
        ...
        String tmp=null, linija=null;
        Vector<Lik> kontejner= new Vector<Lik>();
        while(true){
            tmp = breader.readLine();
            linija = breader.readLine();
            switch(tmp.charAt(0)){
                case 'P':
                    kontejner.add(new Pravokutnik(linija));
                    break;
                case 'K':
                    kontejner.add(new Kvadrat(linija));
                    break;
                case 'T':
                    //itd ...
                    // -> nastavak
```

NE programirati ovako!!! [error handling izostavljen zbog manjka prostora!]

poziv konstruktora

- Ispisujemo površine i ukupnu površinu:

```
        int ukupna=0
        for(int i = 0; i < kontejner.size(); i++)
        {
            povrsina=kontejner.get(i).izracunajPovrsinu();
            ukupna+=povrsina;
            System.out.print(kontejner.get(i));
            System.out.println(", povrsina=[" + povrsina + "]cm^2" );
        }
        System.out.println("Ukupna povrsina = [" +ukupna + "]cm^2");
    }
}
```

Pokretanje

- Svaki razred u posebnoj datoteci, istog imena kao i ime razreda (s nastavkom *.java*).
- Prevoditelj se pokreće naredbom:
  `'javac <datoteke sa izvornim kodom>'`
- nakon čega nastaju *.class* datoteke koje sadrže prevedeni Java program.
- Program se pokreće naredbom
  `'java <razred s metodom main()> <parametri>'`

I sve ovo zbog: