INFORMATIČKI KLUB

FUTURA

Od ideje do aplikacije

Dubrovnik, ožujak-svibanj 2014.

Radionice se organiziraju u suradnji sa **Sveučilištem u Dubrovniku** i **RIT Croatia**, a uz potporu **Zajednice tehničke kulture Dubrovačko - neretvanske županije** i **Grada Dubrovnika**

SVEUČILIŠTE U DUBROVNIKU
UNIVERSITAS STUDIORUM RAGUSINA

R·I·T Croatia

zajednica tehničke kulture
ZTK
Dubrovnik

FUTURA

Uvod u objektno orijentirano programiranje

Krunoslav Žubrinić, Informatički klub FUTURA

Dubrovnik, ožujak 2014.

Creative Commons

slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **remiksirati** — prerađivati djelo

pod slijedećim uvjetima:

- **imenovanje**. Morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- **nekomercijalno**. Ovo djelo ne smijete koristiti u komercijalne svrhe.
- **dijeli pod istim uvjetima**. Ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela. Najbolji način da to učinite je linkom na ovu internetsku stranicu.
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.
Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s http://creativecommons.org/.

1. DIO: Uvod u programiranje

- Računalni program
- Varijable
- Osnovni operatori
- Programske strukture
  - Slijed instrukcija
  - Grananje
  - Petlja
  - Funkcije

Primjeri programa u programskim jezicima **C**, **Java** i **PHP**

Računalni program

Računalni program je skup uputa računalu što treba učiniti i kako to izvesti

Analogija s receptom koji sadrži:

- Popis sastojaka (**varijable**)
- Popis uputa za obradu sastojaka (**instrukcije**)

Palačinke

Sastojci:
- 3 jaja
- 250g brašna
- 350 ml mlijeka
- 1 dcl mineralne vode
- malo soli
- 2 žličice
- pola žličice ruma

Priprema:
Pjenasto umutite cijela jaja, izmiješajte dok ne dobijete gustu homogenu smjesu, ulijte ulje pa mlijeko i izmiješajte. Potom uz neprestano miješanje pjenjačom sipajte brašno dok ne potrošite zadanu mjericu, dodajte prstohvat soli i malo ruma tek toliko da smjesa dobije aromu, a još važnije da spriječite upijanje ulja. Ukoliko vam se smjesa čini gusta razrijedite je mineralnom vodom.

```
// C program za izračun duljine hipotenuze
//   pravokutnog trokuta
#include <stdio.h>
#include <math.h>
int main(){
  int a, b;
  float c;
  printf("Unesi duljine stranica:");
  scanf("%d %d", &a, &b);
  c = sqrt(a*a + b*b);
  printf("Duljina hipotenuze je ");
  printf(" %.2f\n", c);
  return 0;
}
```

Računalni program

Varijabla

- memorijska lokacija čiji se sadržaj može mijenjati tokom izvođenja - određena je nazivom i tipom podatka.

```
int a;  // memorijska lokacija za spremanje cijelog broja
```

Instrukcija

- Naredba računalu da izvrši određenu operaciju.

```
c = a + b;  // zbroj dva broja pridruži trećem
```

Funkcija

- samostalna programska struktura.
- Sastoji se od niza instrukcija, a na kraju izvršavanja može vratiti rezultat određenog tipa.
- Određena je nazivom, parametrima i povratnom vrijednošću.

```
int opsegTrokuta(int a, int b, int c){ … }
// Funkcija računa opseg trokuta i vraća izračunatu vrijednost
```

Varijabla

- Dio memorije u kojem se čuva neki podatak.
- Određena je **nazivom,** a u nekim jezicima i **tipom**.
- Pomoću **naziva** se pristupa memorijskoj lokaciji.
- **Tip** varijable određuje kakav tip podataka se čuva na toj memorijskoj lokaciji.

- Neki programski jezici zahtijevaju da se varijabla prije prvog korištenja **deklarira** (da joj se odredi tip i pridruži naziv).

Varijabla

Naziv varijable je jedna riječ koja mora započeti **slovom** (ili znakom _), a može sadržavati **slova**, **brojeve** i znak **podcrte** _

Ispravni nazivi	Neispravni nazivi
duljina_3	1kut
AdresaIKucniBroj	Stranica trokuta
_x	ime-i-prezime

- Varijabli se vrijednost pridružuje pomoću operatora pridruživanja =
  - Vrijednost se može pridružiti:
    - Odmah pri deklaraciji

```
float pi = 3.141;
int x = 7;
```

    - Naknadno u programu

```
int x, y;
...
x = 5;
...
y = x * 2;
```

Aritmetički operatori

Operator	Naziv	Sintaksa korištenja	Primjer
+	**zbrajanje**	**a + b**	x = 2 + 3 // x = 5
-	**oduzimanje**	**a - b**	x = 7 – 4 // x = 3
*	**množenje**	**a * b**	x = 3 * 5 // x = 15
/	**dijeljenje**	**a / b**	x = 12 / 3 // x = 4
%	**modulo** *(ostatak dijeljenja)*	**a % b**	x = 5 % 3 // x = 2
++	**inkrementiranje (unarni operator)**	**a++ ili ++a**	x=0; x++ // x =1
--	**dekrementiranje (unarni operator)**	**a-- ili --a**	x=7; x-- // x =6
=	**pridruživanje**	**a = 5**	*Vrijednost se desne strane izraza se pridružuje varijabli koja stoji s lijeve strane*

Logički operatori

Operator	Naziv	Sintaksa korištenja	Primjer
&&	**logičko I**	**a && b**	if((a>3) **&&** (a%2!=0))
||	**logičko ILI**	**a || b**	if((a<10) **||** (a%2==0))
!	**negacija** *(unarni operator)*	**!a**	if (**!**(a%2==0))
and	**Logičko I** **Samo PHP!**	**a and b**	if((a>3) **and** (a%2!=0))
or	Logičko ILI **Samo PHP!**	**a or b**	if((a<10) **or** (a%2==0))

Relacijski operatori

Operator	Naziv	Sintaksa korištenja	Primjer
<	manje od	a < b	if (a < 10)
>	veće od	a > b	if (a > 0)
==	Operator uspoređivanja	a == b	if (a == 100)
<=	manje ili jednako	a <= b	if (a <= 10)
>=	veće ili jednako	a >= b	if (a >= 0)
!=	različito	a != b	if (a != 5)

PAZI!

`==` uspoređivanje (jesu li izrazi s lijeve i desne strane znaka jednaki?)

`if (a == b)` znači provjeru: **Jesu li a i b jednaki?**

`=` pridruživanje (izraz s desne strane se pridružuje izrazu s lijeve strane)

`a = b` (vrijednost iz **b** se pridružuje **a**. **Nakon ovoga a i b su jednaki**!

Primjeri korištenja varijabli

C

```
  int a = 0, r, c = 0;
  float pi = 3.141, o = 0;
…
  c = 4 * a;
  o = 2 * r * pi;
…
```

SPECIFIČNOST PHP-a!

- Nazivi varijabli u **PHP-u** počinju znakom **$**
- Varijable u PHP-u **ne treba deklarirati** prije korištenja.
- PHP automatski određuje tip varijable na temelju njezinog sadržaja.

```
$b=2;  // cijeli broj
$c=1.127;  // realni broj
$ime="Pero";  // niz znakova
```

PHP

```
  $a = 0;
  $pi = 3.141;
…
  $c = 4 * $a;
  $o = 2 * $r * $pi;
…
```

Varijable u **C-u** i **Javi** prije korištenja **treba deklarirati** (odrediti im tip i naziv)

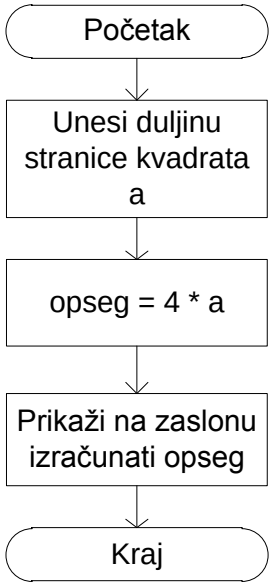
Java

```
  int a, r, c = 0;
  float pi = 3.141, o = 0;
…
  c = 4 * a;
  o = 2 * r * pi;
…
```

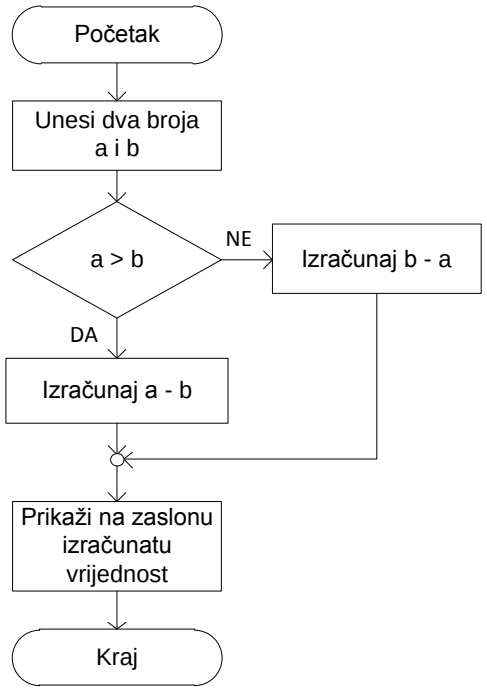
Programska struktura

- Način i redosljed izvršavanja manjih zadataka.
- Tri osnovne vrste programskih struktura:

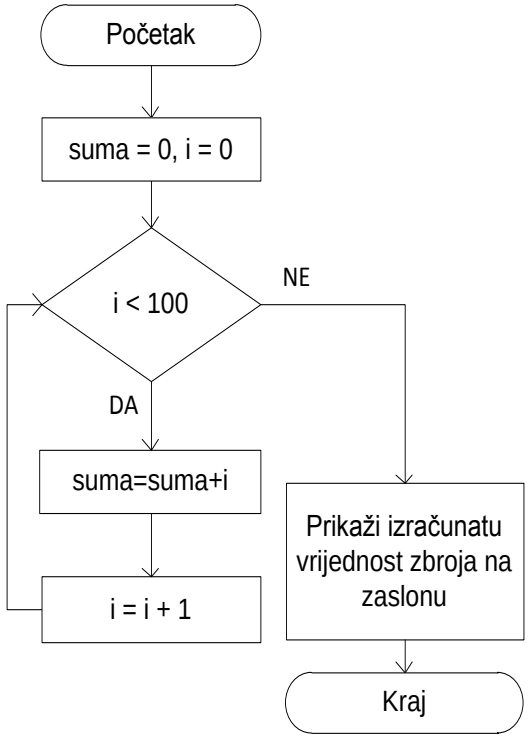
1. Slijed instrukcija



2. Grananje



3. Petlja



Primjeri slijeda instrukcija

Instrukcije se navode **jedna iza druge** i **izvršavaju u tom redoslijedu**.

- Primjer: opseg kvadrata

SPECIFIČNOST PHP-a!
Ispis u prozoru web preglednika!

C

```c
#include<stdio.h>
int main(){
  int a = 5, o = 0;
  o = 4 * a;
  printf("Opseg je %d\n", o);
  return 0;
}
```

Datoteka: **Opseg.c**

PHP

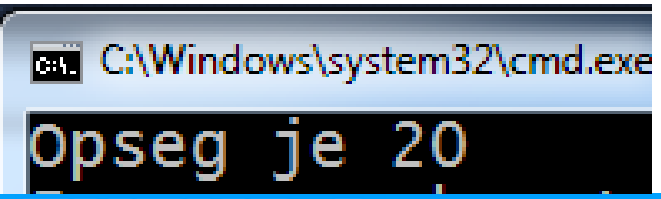
```php
<?php
  $a = 5;
  $o = 0;
  $o = 4 * $a;
  echo "Opseg je ", $o;
?>
```

Datoteka: **Opseg.php**

Java

```java
public class Opseg{
  public static void main(String[] args){
    int a = 5, o = 0;
    o = 4 * a;
    System.out.println("Opseg je " + o);
  }
}
```

Datoteka: **Opseg.java**



Primjeri jednostavnog grananja

Na početku se provjerava **zadani uvjet**

- Ako je uvjet zadovoljen izvršava se **prvi blok** instrukcija.
- Ako nije, izvršava se **drugi blok**.
- Primjer: razlika dva broja

PHP

```php
<?php
  $a = 5; $b = 3; $c = 0;
  if ($a > $b)
    $c = $a - $b;
  else
    $c = $b - $a;
  echo "Razlika je ", $c;
?>
```

Datoteka: **Razlika.php**

C

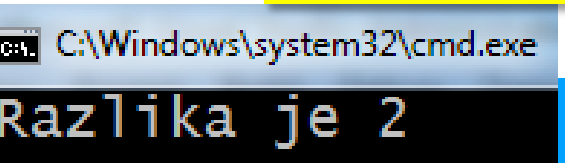
```c
#include<stdio.h>
int main(){
  int a = 5, b = 3, c = 0;
  if (a > b)
    c = a - b;
  else
    c = b - a;
  printf("Razlika je %d", c);
  return 0;
}
```

Datoteka: **Razlika.c**

Java

```java
public class Razlika{
  public static void main(String[] args){
    int a = 5, b = 3, c = 0;
    if (a > b)
      c = a - b;
    else
      c = b - a;
    System.out.println("Razlika je "+c);
  }
}
```

Datoteka: **Razlika.java**



Grananje

Ovisno o uvjetu, program se ponaša drugačije

```
…
if (a > b) {
   // prvi blok
}
else {
   // drugi blok
}
```

Ako je izraz u zagradama iza **if** istinit (bilo koji izraz koji može biti istina ili laž), izvršava se ***prvi blok instrukcija***.

… inače se izvršava ***drugi blok instrukcija*** (iza **else**)

Za određivanje uvjeta kod if instrukcije se mogu koristiti svi logički i relacijski operatori.

Ako se provjerava jednakost PAZI!
b == 0 **uspoređuje** vrijednost **b** s nulom
b = 0 **pridružuje** nulu varijabli **b**

Grananje

- Instrukcije grananja se mogu **ulančavati** i međusobno **ugnježđivati**

```
…
if (c > max){
    med = max;
}
else
    if (c < min){
      med = min;
    }
    else{
      med = c;
    }
```

Ako je **c** veći od **max**, med=max, a obrada završava s tom instrukcijom.

Inače, ako **c NIJE** veći od **max**, ali je **c** manji od **min**, med=min, a obrada završava s ovom instrukcijom.

Ako niti jedan od prethodnih uvjeta nije istinit, to znači da **c** ima središnju vrijednost, pa je med=c, a izvodi se samo instrukcija unutar ovog else bloka.

`if (c < min) … else …` blok je ugnježđen unutar `else` bloka više razine Čitav blok je **jedna cjelina** pa ga ne treba uokvirivati vitičastim zagradama.

Primjeri višestrukog grananj

PHP višestruko grananje
`if()…`
`elseif()…`
`else…`

Primjer: pozdrav ovisno o dobu dana

C

```c
#include<stdio.h>
int main(){
  int sat = 13;
  if (sat < 12)
    printf("Dobro jutro!");
  else if ((sat<19)&&(sat>11))
    printf("Dobar dan!");
  else
    printf("Dobro veče!");
  return 0;
}
```

Datoteka: **Pozdrav.c**

PHP

```php
<?php
  $sat = 13;
  if ($sat < 12)
    echo "Dobro jutro!";
  elseif (($sat<19)&&($sat>11))
    echo "Dobar dan!";
  else
    echo "Dobro veče!";
?>
```

Datoteka: **Pozdrav.php**

Java

```java
public class Pozdrav{
  public static void main(String[] args){
    int sat = 13;
    if (sat < 12)
      System.out.println("Dobro jutro!");
    else if ((sat < 19) && (sat > 11))
      System.out.println("Dobar dan!");
    else
      System.out.println("Dobro veče!");
  }
}
```

Datoteka: **Pozdrav.java**

PAZI!

C i **Java** – dvije instrukcije **if** ugnježđene unutar **else** više razine

```
if…
else…
  if …
  else …
```

Primjeri petlje

Petlja se koristi kada istu obradu treba izvršiti više puta

- Obrada se ponavlja sve dok je zadovoljen zadani uvjet
- Primjer: zbroj brojeva u intervalu od 1 do 100

C

```c
#include<stdio.h>
int main(){
  int i, suma=0;
  for (i=1;i<=100;i++){
     suma=suma+i;
  }
  printf("Suma je %d", suma);
  return 0;
}
```

Datoteka: **Suma.c**

PHP

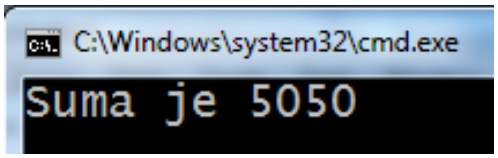
```php
<?php
  $suma=0;
  for ($i=1;$i<=100;$i++){
      $suma=$suma+$i;
  }
  echo "Suma je ", $suma;
?>
```

Datoteka: **Suma.php**

Java

```java
public class Suma{
  public static void main(String[] args){
    int i, suma=0;
    for (i=1;i<=100;i++){
      suma=suma+i;
    }
    System.out.println("Suma je " + suma);
  }
}
```

Datoteka: **Suma.java**



for petlja

Petlja se koristi kada istu obradu treba izvršiti više puta

```
for (početna_vrij;uvjet_završetka;korak_promjene){…}
```

Varijabli **brojača** prije prvog korištenja treba pridružiti početnu vrijednost. Inicijalizacija se odrađuje **samo jednom prije prvog prolaza kroz petlju**.

Uvjet koji mora biti zadovoljen da bi se izvršile instrukcije navedene u tijelu petlje. Provjera se odrađuje **na početku svakog prolaza**.

Na kraju svakog prolaza kroz petlju odrađuje se ovaj izraz (primjerice povećava se vrijednost brojača).

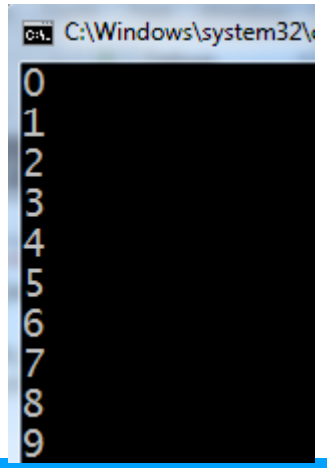
for petlja – 1. primjer

for (i = 0 ; i < 10 ; i++){ … }

- **i** je varijabla brojača
- Na početku se brojač postavlja na 0
- U svakom prolazu brojač se povećava za 1
- Petlja se vrti sve dok brojač ne dođe do 9
- Brojač poprima vrijednosti svih brojeva u intervalu od 0 do 9:
- u 1. prolazu i=0
- u 2. prolazu i=1
- …
- u 10. prolazu i=9

```
#include<stdio.h>
void petlja1(){
  int i;
  for (i=0;i<10;i++){
     printf("%d\n", i);
  }
}
```

Datoteka: **Petlja.c**
funkcija: **petlja1()**



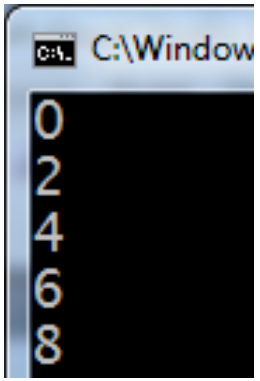
for petlja – 2. primjer

for (i = 0 ; i < 10 ; i = i + 2) {…}

- **i** je varijabla brojača
- Na početku se brojač postavlja na 0
- U svakom prolazu brojač se povećava za 2
- Petlja se vrti sve dok brojač ne dođe do 9
- Brojač poprima vrijednosti svih parnih brojeva u intervalu 0-9:
- u 1. prolazu i=0
- u 2. prolazu i=2
- …

```
#include<stdio.h>
void petlja2(){
  int i;
  for (i=0;i<10;i=i+2){
     printf("%d\n", i);
  }
}
```

Datoteka: **Petlja.c**
funkcija: **petlja2()**

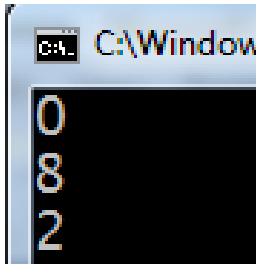


for petlja – 3. primjer

for (; sluc % 2 == 0 ;){…}

- Brojač nam nije potreban pa ga ne navodimo
- Petlja se vrti sve dok je varijabla **sluc** paran broj
- Nakon što varijabla **sluc** poprimi vrijednost neparnog broja, petlja završava.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void petlja3(){
  int sluc=0;
  srand(time(NULL));
  for (; sluc%2==0 ; ){
     printf("%d\n", sluc);
     sluc = rand() % 10;
  }
}
```



Datoteka: **Petlja.c**
funkcija: **petlja3()**

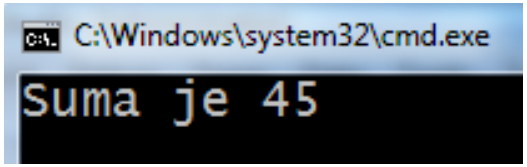
Izlazak iz petlje

Ako iz petlje treba izaći **odmah** "na silu" koristi se instrukcija **break**

- Izađi iz petlje nakon što brojač dođe do prvog broja djeljivog s 10 bez ostatka

C

```c
#include<stdio.h>
int main(){
  int i, suma=0;
  for (i = 1; i <= 100; i++){
    if (i % 10 == 0)
       break;
    suma = suma + i;
  }
  printf("Suma je %d", suma);
  return 0;
}
```



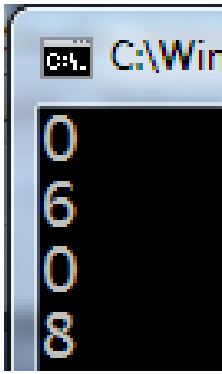
Datoteka: **Suma2.c**

for petlja – 4. primjer

for (; ;){...}

- Beskonačna petlja
- Iz petlje se može izaći samo "na silu" (primjerice pozivom naredbe break)
- Iz petlje se izlazi nakon što se generira neparan broj

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void petlja4(){
  int sluc=0;
  srand(time(NULL));
  for (; ;){
     printf("%d\n", sluc);
     sluc = rand() % 10;
     if (sluc % 2 == 1)
        break;
  }
}
```



Datoteka: **Petlja.c**
funkcija: **petlja4()**

Funkcija

Funkcija je samostalna programska struktura koja se sastoji od jedne ili više instrukcija, a na kraju izvršavanja može vratiti rezultat određenog tipa.

Koristi se:

- kada se složeni zadatak želi razbiti na manje i jednostavnije cjeline.
- kada isti zadatak treba izvršiti više puta.

Varijable deklarirane unutar funkcije (lokalne varijable) vidljive su samo unutar te funkcije.

Funkcije koje vraćaju rezultat čine to pomoću naredbe **return** iza koje se navodi vrijednost koja se vraća.

```
float vrati_pi() {
    float pi = 3.14; // lokalna varijabla
    return pi; // funkcija vraća podatak tipa float
}
```

Primjeri funkcija

Definicija funkcije sadrži programski kod koji se izvodi

- Primjer: opseg trokuta

C

```c
int opseg_trokuta(int a, int b, int c){
  int opseg;
  opseg = a + b + c;
  return opseg;
}
```

Datoteka: **OpsegT.c**

PAZI!

C i **Java**
Navodi se **tip** i **naziv parametara**.
Funkcija vraća vrijednost pozivom ključne riječi **return**
Funkcija koja ništa ne vraća je tipa **void**

Java

```java
public class Trokut{
  public int opsegTrokuta(int a, int b, int c){
    int opseg;
    opseg = a + b + c;
    return opseg;
  }
}
```

Datoteka: **OpsegT.java**

PAZI!

Specifičnost PHP funkcija
Funkcija je određena ključnom riječi **function**.
Svaka funkcija vraća neku vrijednost (pa čak i ako se ne koristi ključna riječ **return**).
Vrsta podatka koji se vraća određuje se automatski na temelju njegovog sadržaja.

PHP

```php
<?php
  function opseg_trokuta($a, $b, $c){
    $opseg = $a + $b + $c;
    return $opseg;
  }
?>
```

Datoteka: **OpsegT.php**

Funkcija

C i Java

```
int    opseg_trokuta  (int a, int b, int c){ … }
```

Tip povratne vrijednosti. **void** – funkcija ne vraća ništa

Naziv funkcije

Tip parametra

Naziv parametra

Implementacija funkcije. Instrukcije su navedene unutar vitičastih zagrada. Funkcija vraća povratnu vrijednost pomoću **return**

PHP

```
function opseg_trokuta  ($a, $b, $c)
```

```
{ … }
```

Ključna riječ **function** određuje da je ovo funkcija

PHP automatski određuje tip podatka na temelju sadržaja varijable pa se tip varijabli i tip povratne vrijednosti funkcije eksplicitno ne trebaju navoditi

Pozivanje funkcija

C

```c
#include<stdio.h>
int opseg_trokuta(int a, int b, int c){
  return a + b + c;
}
int main(){
  int a = 3, b = 4, c = 5, op;
  op = opseg_trokuta(a, b, c);
  printf("Opseg je %d", op);
  return 0;
}
```

Datoteka: **OpsegT.c**

PHP

```php
<?php
  function opseg_trokuta($a, $b, $c){
    return $a + $b + $c;
  }
  $a = 3; $b = 4; $c = 5;
  $op = opseg_trokuta($a, $b, $c);
  echo "Opseg je ", $op;
?>
```

Datoteka: **OpsegT.php**

Java

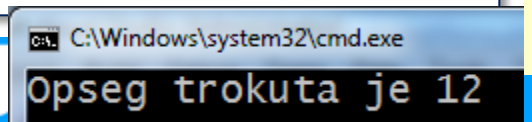
```java
public class OpsegT{
  public int opsegTrokuta(int a, int b, int c){
    return a + b + c;
  }
  public void izracun(){
    int a = 3, b = 4, c = 5, op;
    op = opsegTrokuta(a, b, c);
    System.out.println("Opseg je "+op);
  }
}
```

Datoteka: **OpsegT.java**

Funkcije se pozivaju navođenjem **naziva funkcije** i predavanjem parametara.

Broj (i tip) parametara mora odgovarati broju (i tipu) parametara navedenom pri deklaraciji funkcije.

Funkcija vraća vrijednost deklariranog tipa



2. DIO: Uvod u OO programiranje

- Objektno orijentirano programiranje
- Osnovni pojmovi
  - Objekt
  - Klasa
  - Varijable i metode
  - Konstruktor
- Stvaranje i korištenje objekata
- Pozivanje metoda
- Nasljeđivanje

Primjeri programa u programskim jezicima **Java**, **Objective-C** i **PHP**

Pristupi razvoju programa

Proceduralno programiranje

- Program se promatra kao niza programskih odsječaka (funkcija) koje sudjeluju u rješavanju zadatka.
- Podaci su neovisni o procedurama.

Objektno orijentirano programiranje

- Osnovna ideja je razbijanje problema na niz zaokruženih cjelina (objekata) koje objedinjavaju podatke i operacije.
- Operacije objekta djeluju nad podacima tog objekta.

Većina današnjih programskih jezika su

- Objektno orijentirani

  ili

- Omogućuju objektno orijentirani razvoj

http://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages

Pristupi razvoju programa

Primjer – rotiranje geometrijskog lika na zaslonu

Proceduralno	Objektno orijentirano
1. Listaj sve elemente 2. Odredi vrstu određenog elementa 3. Ovisno o vrsti lika pozovi funkciju koja rotira lik određene vrste. 4. Ovisno o vrsti lika ažuriraj koordinate.	1. Listaj sve objekte 2. Zatraži određeni objekt da se zarotira.

Objektno orijentirano programiranje

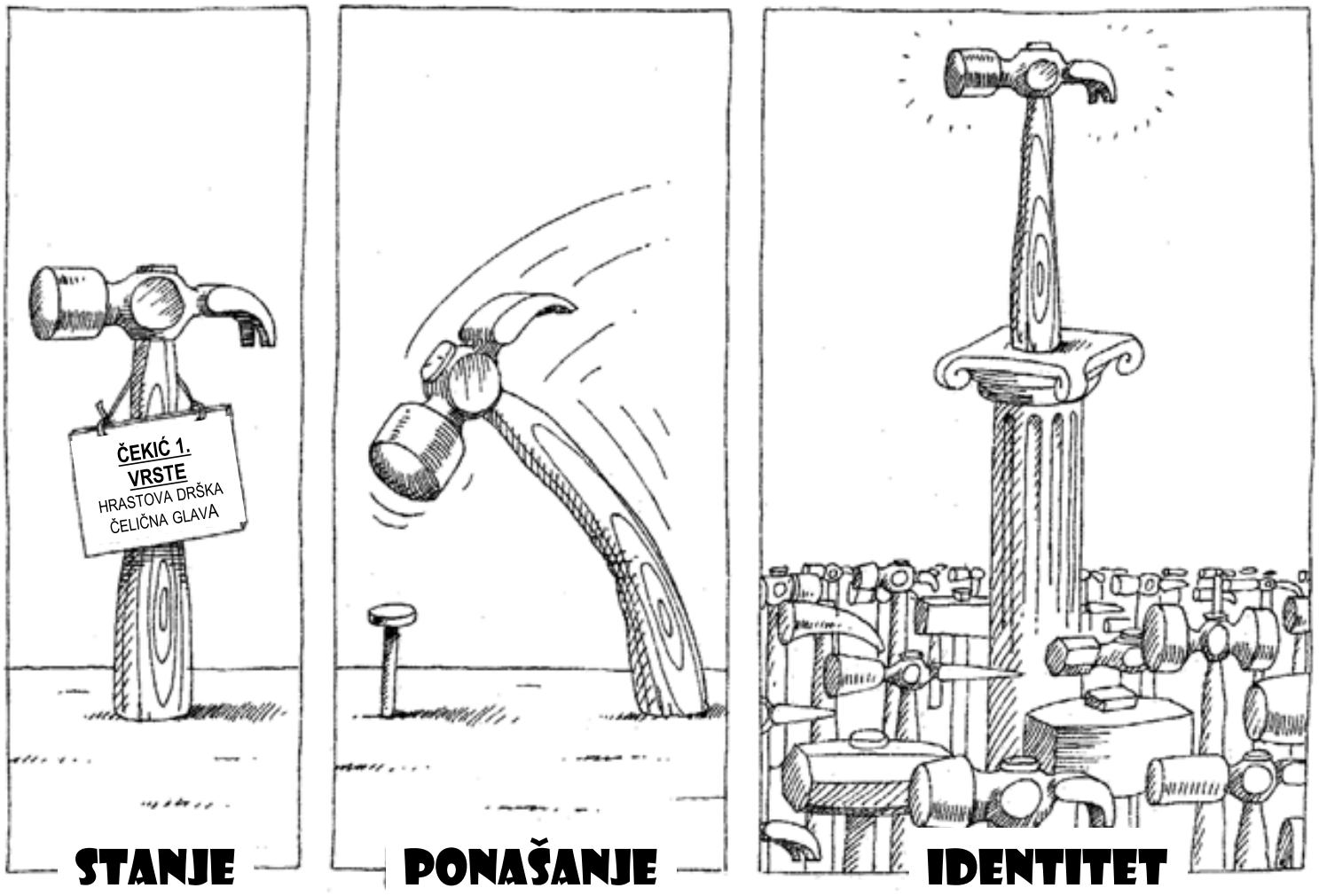
Osnovni koncept OOP-a je objedinjavanje podataka (varijabli) i operacija (funkcija) koje manipuliraju podacima u jednu cijelinu koja se naziva **objekt**.

- Program se sastoji od skupa objekata koji međusobnom komunikacijom riješavaju problem.

Ovaj način razvoja je pogodan za razvoj složenih programskih sustava.

- Razvoj je brži jer se omogućuje jednostavno korištenje postojećeg programskog koda.
- Posebno je pogodan za razvoj aplikacija s grafičkim korisničkim sučeljem.

Što je objekt?



Objekt

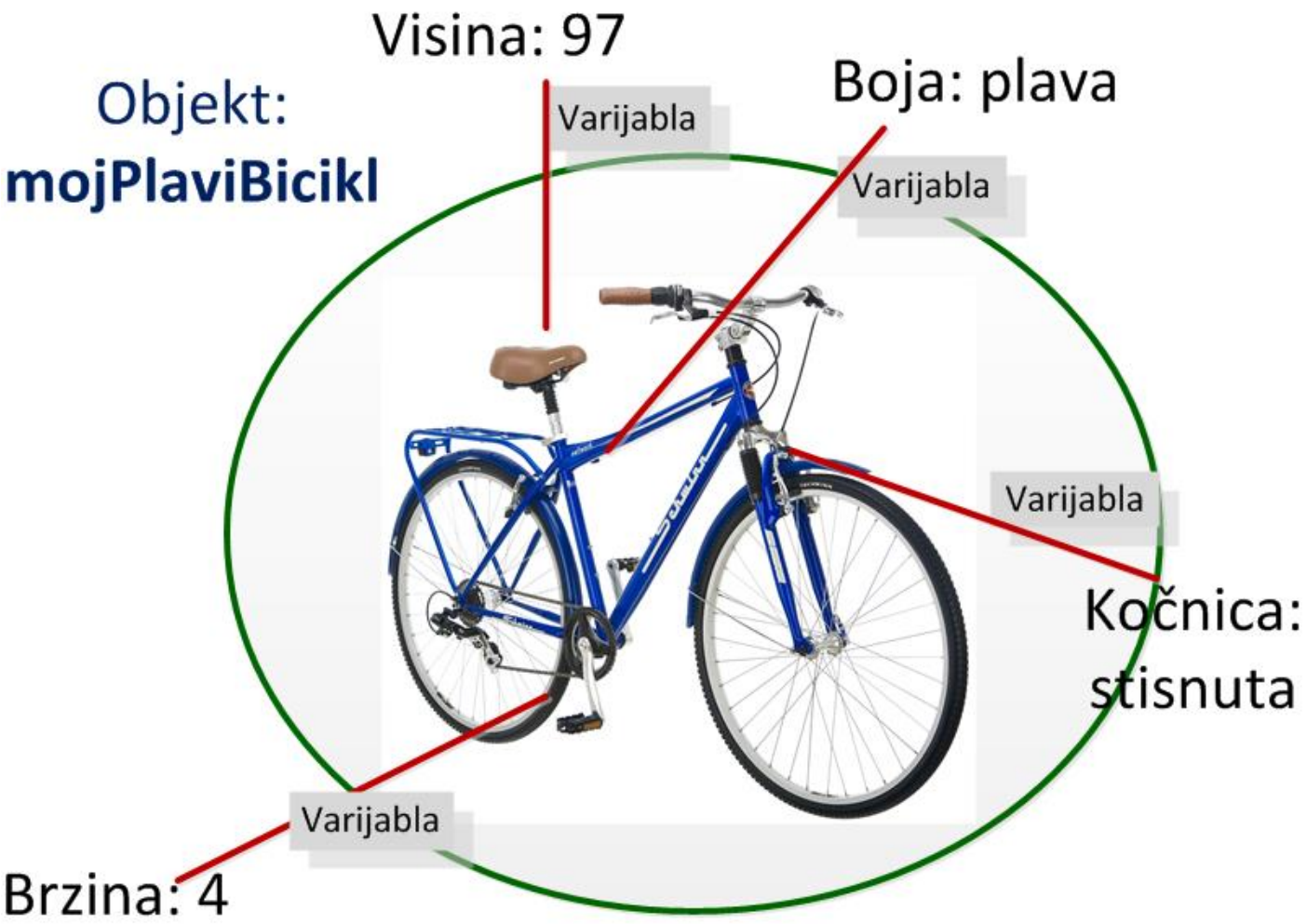
Objekt može biti bilo koji predmet ili pojam iz stvarnog ili apstraktnog svijeta.

- npr. televizor, računalo, knjiga, stolica,...

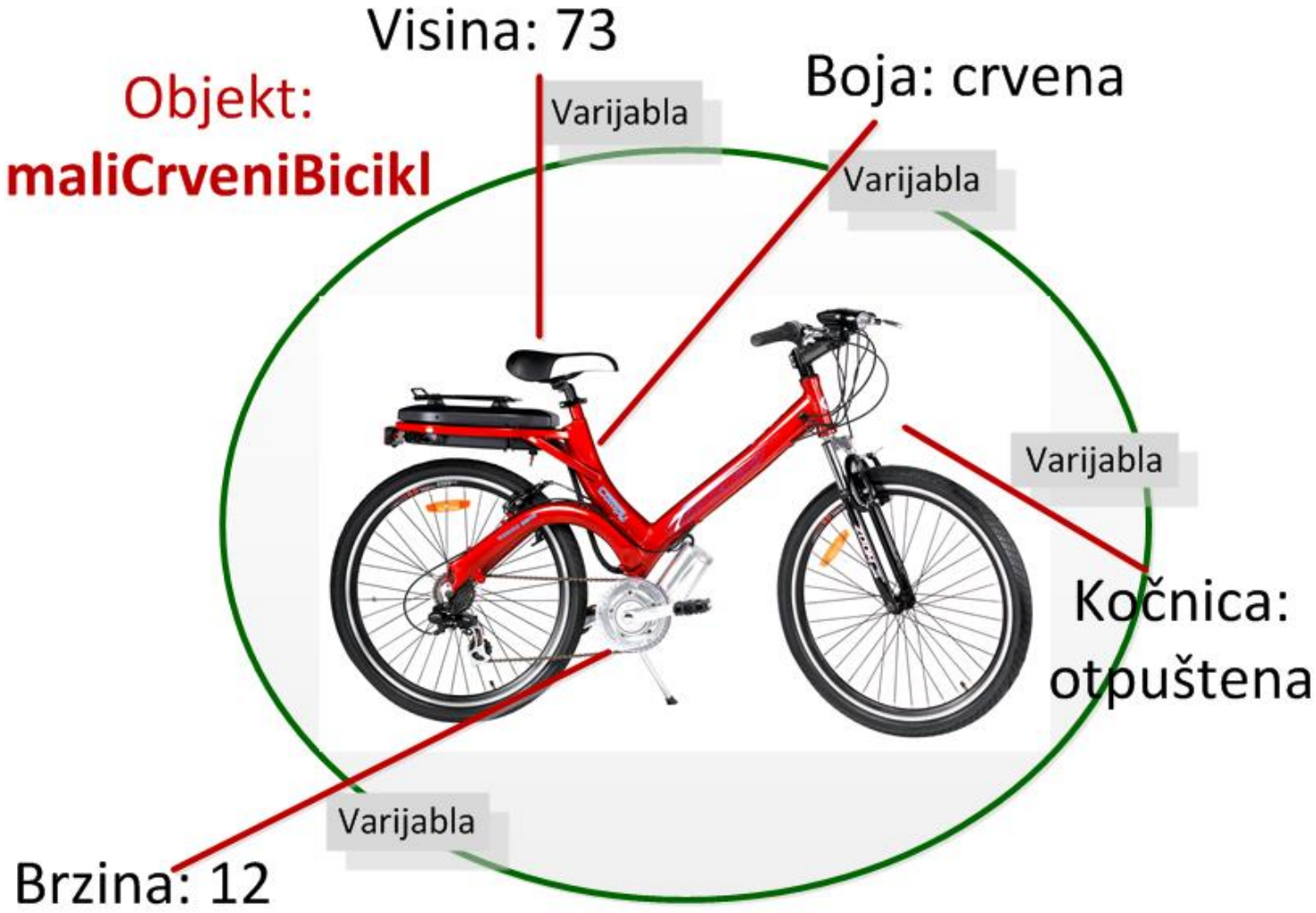
Znamo kako objekti u stvarnom svijetu izgledaju i kako se "ponašaju" (što se s njima događa nakon neke akcije)

- Stanje objekta je opisano konkretnim vrijednostima varijabli
- Ponašanje objekta definiraju funkcije koje rade s varijablama objekta (metode)
  - Stanje objekta mijenja se uslijed ponašanja (npr. komunikacijom s drugim objektima)
- Identitet predstavljaju osobine objekta po kojima se on razlikuje od drugih objekata

Objekt


Visina: 97
Objekt:
mojPlaviBicikl
Varijabla
Boja: plava
Varijabla
Varijabla
Kočnica:
stisnuta
Varijabla
Brzina: 4

Objekt


Visina: 73
Varijabla
Boja: crvena
Varijabla
Objekt:
maliCrveniBicikl
Varijabla
Kočnica:
otpuštena
Varijabla
Brzina: 12

Objekti

Identitet

stariCrt:Televizor
- velicinaEkrana=55 - proizvodjac="Sony" - boja=GRAY
+ upali() : void + ugasi() : void + promijeniProgram(int) : void + promijeniGlasnocu(int) : void

plazma:Televizor
- velicinaEkrana=110 - proizvodjac="Panasonic" - boja=BLACK
+ upali() : void + ugasi() : void + promijeniProgram(int) : void + promijeniGlasnocu(int) : void

Stanje

prijenosni:Televizor
- velicinaEkrana=5 - proizvodjac="LG" - boja=NAVY
+ upali() : void + ugasi() : void + promijeniProgram(int) : void + promijeniGlasnocu(int) : void

Ponašanje

Što je klasa?

Klasa

Klasa je obrazac (ili nacrt) koji definira varijable i metode zajedničke skupini objekata

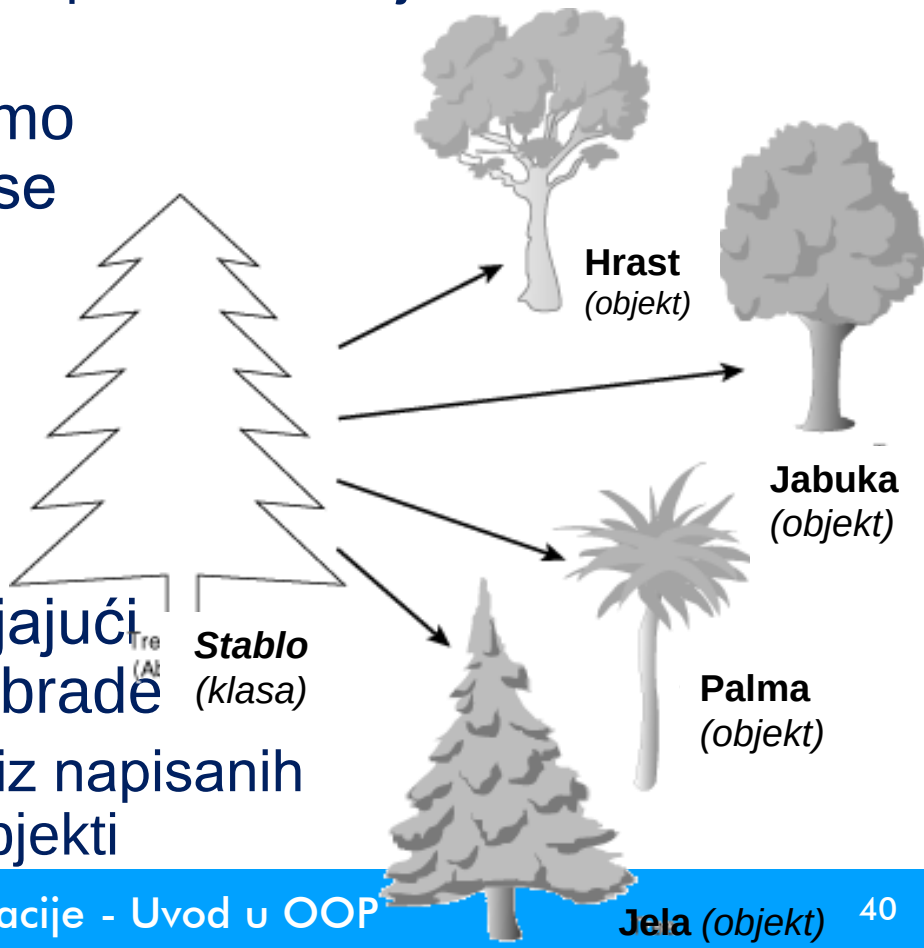
- Klasa nije konkretan objekt već predložak koji koristimo za stvaranje objekata

Kada dizajniramo klasu, moramo razmišljati o objektima koji će se kreirati iz te klase

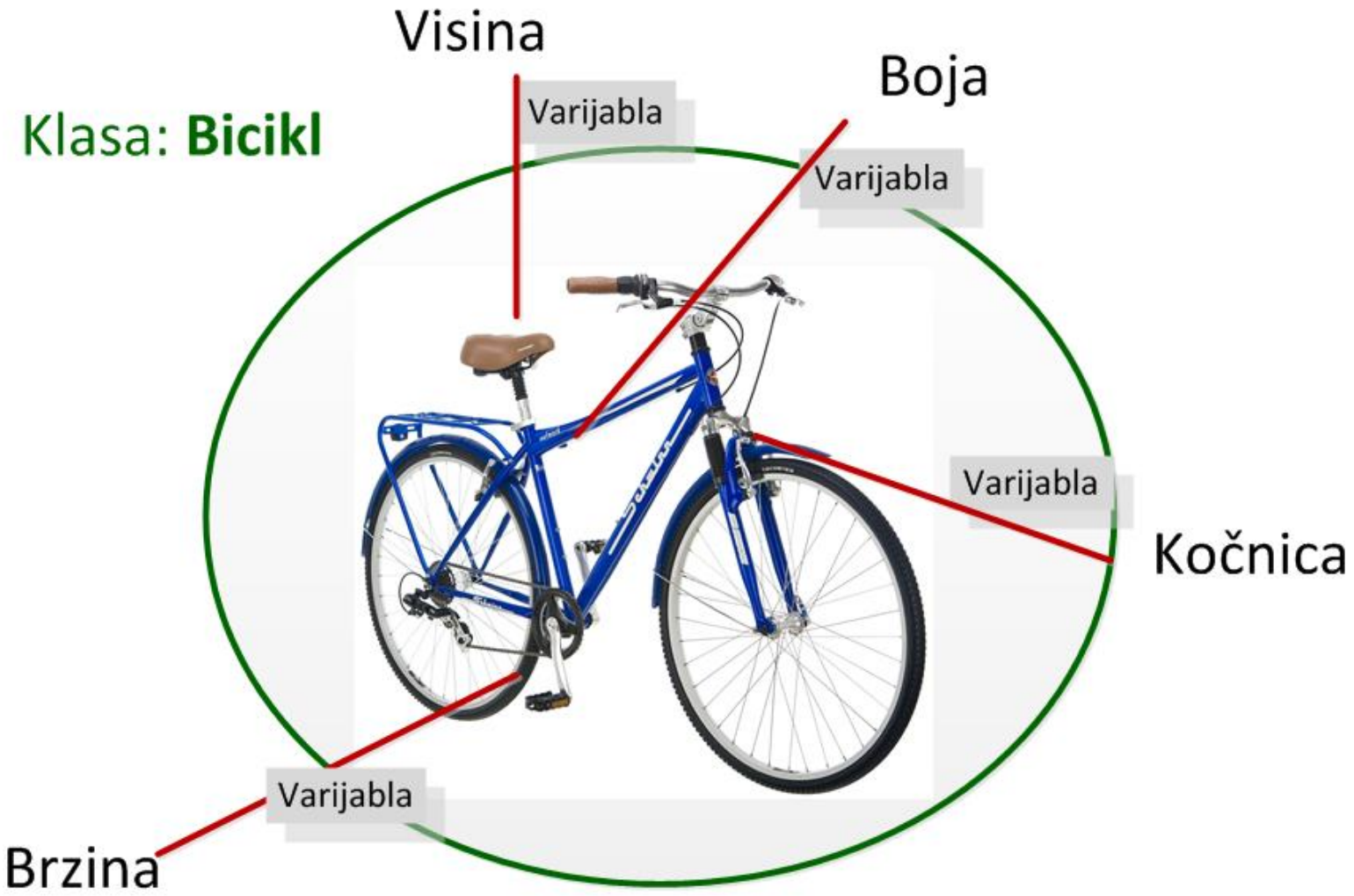
- Kako izgledaju ti objekti? → **varijable**
- Kako se ponašaju ti objekti? → **metode**

Programeri pišu **klase** razmišljajući o **objektima** koji su predmet obrade

- Tijekom izvođenja programa iz napisanih klasa se stvaraju konkretni objekti

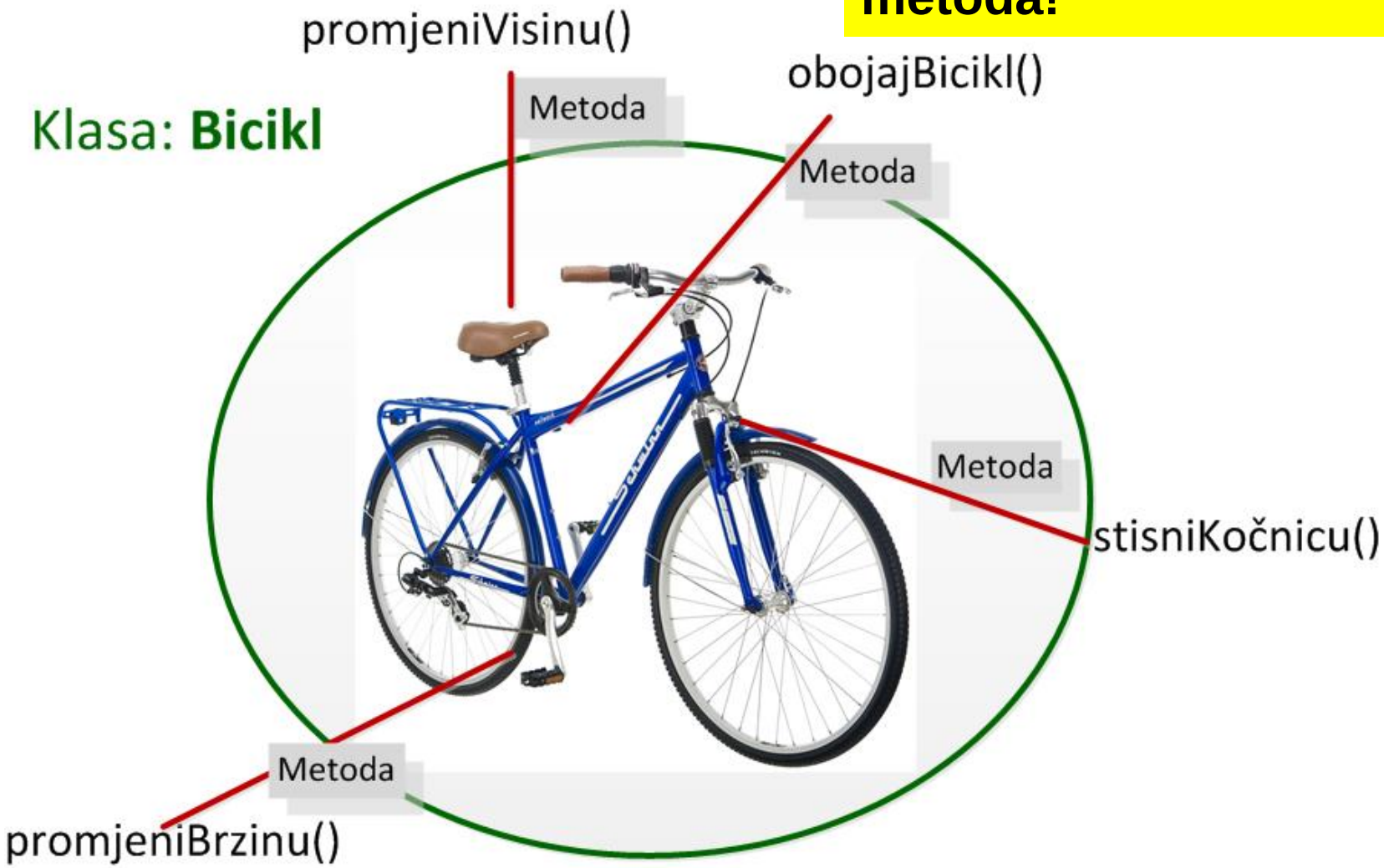


Klasa - varijable



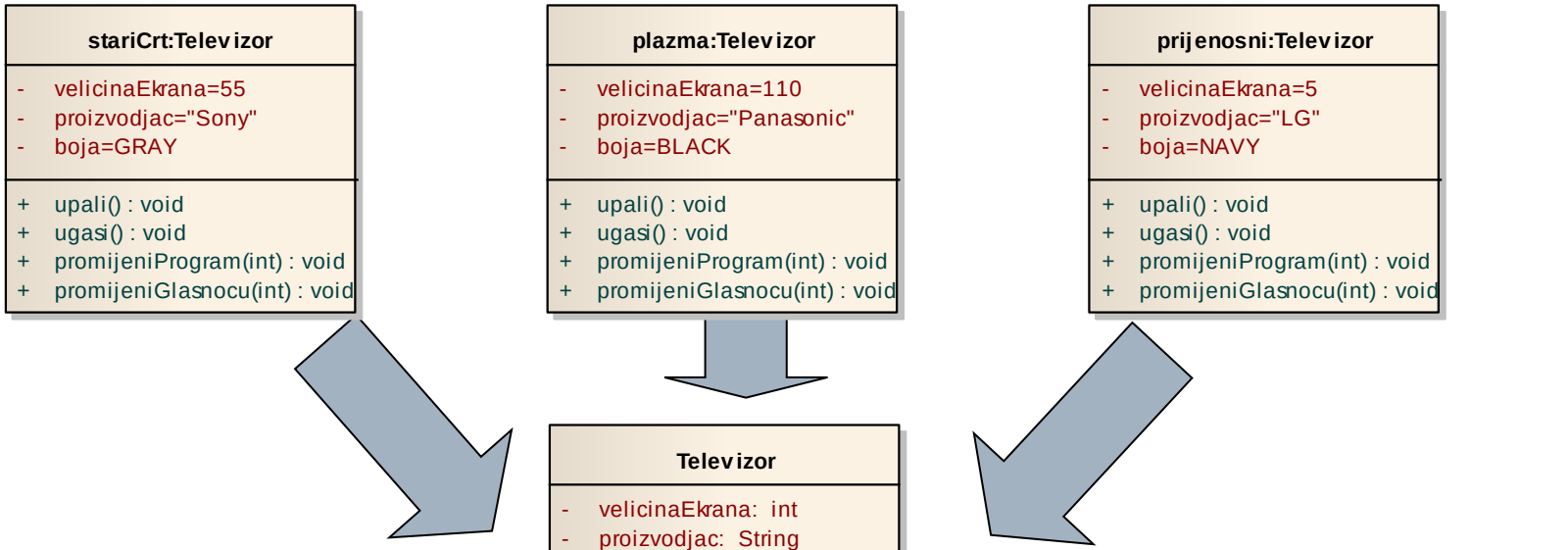
Klasa - metode

Sadržaj varijabli određenog objekta se mijenja pomoću metoda!



Objekti ⇨ Klasa

Objekti



Klasa

Objekti ⇨ Klasa

Objekti se stvaraju (*instanciraju*) na temelju opisa klase.

- Stvoreni objekt je **instanca** odgovarajuće klase.

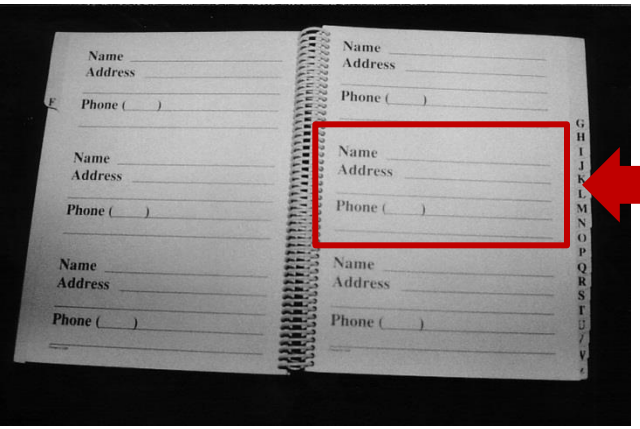
Npr. imamo prazan adresar s kontakt podacima:

- ime i prezime,
- adresa,
- telefonski broj.

Kontakt podaci su klasa koja sadrži atribute (ime i prezime, adresu i telefonski broj osobe)

- Kada u adresar upisujemo podatke nove osobe, stvaramo novi objekt klase **Kontakt**
- Metode su akcije koje možemo raditi s podacima adresara
  - dodati novu osobu,
  - promijeniti podatke izabrane osobe,
  - tražiti podatke određene osobe,...

Objekti ⇨ Klasa



kontakt podaci = **klasa**

Ime i prezime:
Adresa:
Telefon:

operacije s kontaktima = **metode**

Ime i prezime: Ana Anić
Adresa: I. Vojnovića 2, Dubrovnik
Telefon: 020/277-110

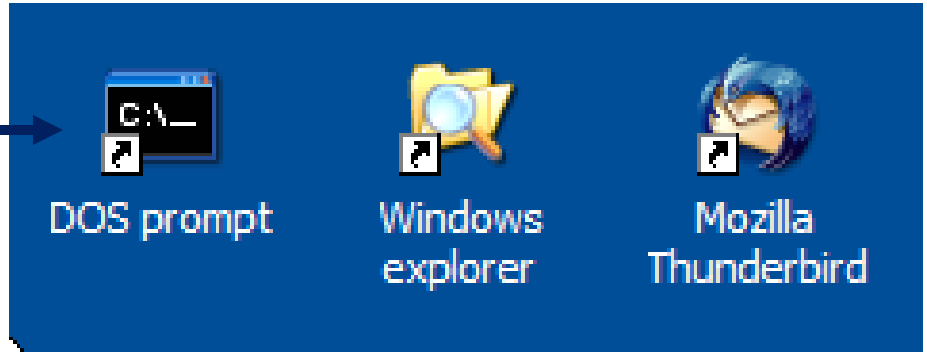
Ime i prezime: Pero Perić
Adresa: Vukovarska 67 Dubrovnik
Telefon: 020/221-112

instance klase = **objekti**

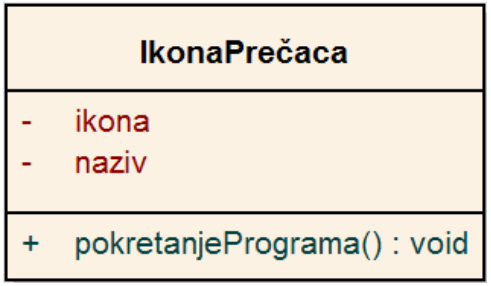
Objekti ⇨ Klasa

Različite ikone prečaca (*svaka ikona predstavlja 1 objekt*)

Objekti



Klasa

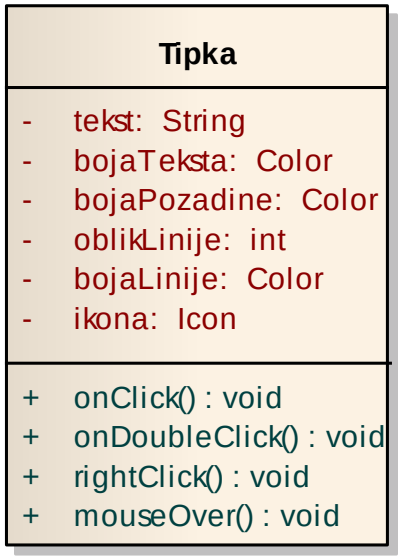


U praksi ne postoji nešto što se zove "*ikona prečaca*", međutim svaka konkretna ikona ima neke zajedničke elemente.

Objekti ⇨ Klasa

Elementi grafičkog sučelja

Objekti



Klasa

Elementi tipke i akcije koje su dozvoljene s tipkom

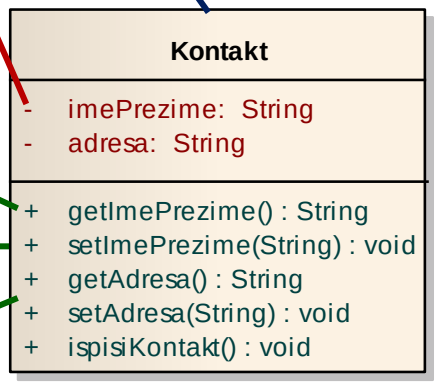
Klasa: Java

```
public class Kontakt {
    private String imePrezime;
    private String adresa;
    public void setImePrezime(String ip){
        imePrezime = ip;
    }
    public void setAdresa(String ad){
        adresa = ad;
    }
    public String getImePrezime(){
        return imePrezime;
    }
    public String getAdresa(){
        return adresa;
    }
    public void ispisiKontakt(){
        System.out.println("Ime i prezime: "+imePrezime);
        System.out.println("Adresa: "+adresa);
    }
}
```

Naziv klase

Varijable

Metode



UML opis klase

Varijable i metode: Java

Deklaracija varijable

```
private   int   x  ;
```

Razina pristupa
public – "svatko"
private – samo metode te klase

Tip varijable

Naziv varijable

Metoda

```
public  int   zbroj  (int a, int b)   {…}
```

Tip povratne vrijednosti
void – metoda ne vraća ništa

Naziv metode

Tip parametra

Naziv parametra

Implementacija metode. Naredbe su navedene unutar vitičastih zagrada.

Klasa: Java

```
public class Kontakt {
    private String imePrezime;
    private String adresa;
    public void setImePrezime(String ip){
        imePrezime = ip;
    }
    public void setAdresa(String ad){
        adresa = ad;
    }
    public String getImePrezime(){
        return imePrezime;
    }
    public String getAdresa(){
        return adresa;
    }
    public void ispisiKontakt(){
        System.out.println("Ime i prezime: " + imePrezime);
        System.out.println("Adresa: " + adresa);
    }
}
```

Datoteka: **Kontakt.java**

Klasa: Objective-C

Nastao proširenjem standardnog C-a s elementima objektno orijentiranog pristupa

Posljedica (jedna od) je da kada se piše klasa, njezin programski kod prvo treba deklarirati (opisati varijable i metode/funkcije), a nakon toga implementirati (napisati konkretan programski kod).

Deklaracija klase se navodi između:

```
@interface
…
@end
```

Implementacija klase se navodi između:

```
@implementation
…
@end
```

Klasa: Objective-C (deklaracija)

```
#import <Foundation/NSObject.h>
#import <Foundation/NSString.h>
@interface Kontakt: NSObject {
    @private
    NSString * imePrezime;
    NSString * adresa;
  }
  -(void) setImePrezime: (NSString *) ip;
  -(void) setAdresa: (NSString *) adr;
  -(NSString *) getImePrezime;
  -(NSString *) getAdresa;
  -(void) ispisiKontakt;
@end
```

PAZI!

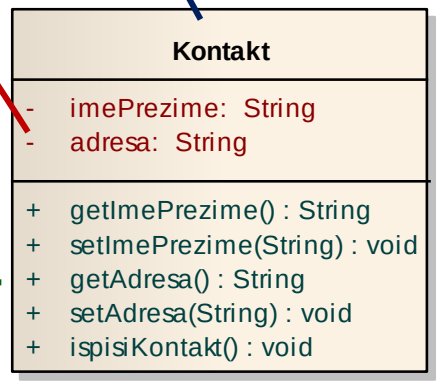
C `#include<…>`

Objective-C `#import<…>`

Naziv klase

Varijable

Metode



UML opis klase

Klasa: Objective-C (implementacija)

```
#import "Kontakt.h"
@implementation Kontakt
  -(NSString*) getImePrezime {
    return imePrezime;
  }
  -(NSString*) getAdresa {
    return adresa;
  }
  -(void) setImePrezime: (NSString*) ip {
    imePrezime = ip;
  }
  -(void) setAdresa: (NSString*) ad {
    adresa = ad;
  }
  -(void) ispisiKontakt {
    NSLog(@"Ime i prezime: %@\n", imePrezime);
    NSLog(@"Adresa: %@\n", adresa);
  }
@end
```

Datoteka u kojoj je navedena deklaracija klase

Varijable i metode: Objective-C

```
int x ;
```

Tip varijable — Naziv varijable

```
Trokut * t ;
```

Tip varijable * označava da se radi o adresi objekta — Naziv varijable

```
- (int) zbroj : (int) a {…}
```

- Vrsta metode:
  - metoda objekta
  + metoda klase
- Tip povratne vrijednosti
  void – metoda ne vraća ništa
- Naziv metode
- Oznaka da metoda prima parametre
- Tip parametra
- Naziv parametra
- Implementacija metode. Naredbe su navedene unutar vitičastih zagrada.

Klasa: Objective-C (deklaracija)

```
#import <Foundation/NSObject.h>
#import <Foundation/NSString.h>
@interface Kontakt: NSObject {
    @private
    NSString * imePrezime;
    NSString * adresa;
  }
  -(void) setImePrezime: (NSString *) ip;
  -(void) setAdresa: (NSString *) adr;
  -(NSString *) getImePrezime;
  -(NSString *) getAdresa;
  -(void) ispisiKontakt;
@end
```

Datoteka: **Kontakt.h**

Klasa: Objective-C (implementacija)

```
#import "Kontakt.h"
@implementation Kontakt
  -(NSString*) getImePrezime {
    return imePrezime;
  }
  -(NSString*) getAdresa {
    return adresa;
  }
  -(void) setImePrezime: (NSString*) ip {
    imePrezime = ip;
  }
  -(void) setAdresa: (NSString*) ad {
    adresa = ad;
  }
  -(void) ispisiKontakt {
    NSLog(@"Ime i prezime: %@\n", imePrezime);
    NSLog(@"Adresa: %@\n", adresa);
  }
@end
```

Datoteka: **Kontakt.m**

Klasa: PHP

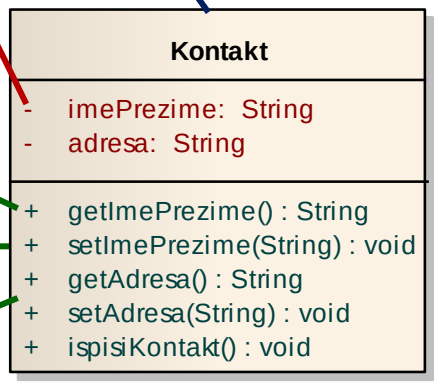
Varijablama objekta (deklariranima van metoda) u PHP-u se pristupa pomoću **`$this->varijabla`**

Naziv klase

Varijable

Metode

```
class Kontakt {
    private $imePrezime;
    private $adresa;
    public function setImePrezime($ip){
        $this->imePrezime = $ip;
    }
    public function setAdresa($ad){
        $this->adresa = $ad;
    }
    public function getImePrezime(){
        return $this->imePrezime;
    }
    public function getAdresa(){
        return $this->adresa;
    }
    public function ispisiKontakt(){
        echo "Ime i prezime: ", $this->imePrezime ;
        echo "Adresa: ", $this->adresa ;
    }
}
```



UML opis klase

Varijable i metode: PHP

Za razliku od lokalnih varijabli (unutar funkcije/metode), **varijable objekta (instance) u PHP-u morate deklarirati**.

Deklaracija varijable

```
private      $x  ;
```

Razina pristupa
public – "svatko"
private – samo metode te klase

Naziv varijable

Metoda

```
public  function  zbroj  ($a)  {…}
```

Oznaka da se radi o funkciji/metodi

Naziv metode

Naziv parametra

Implementacija metode. Naredbe su navedene unutar vitičastih zagrada.

Klasa: PHP

```
class Kontakt {
    private $imePrezime;
    private $adresa;
    public function setImePrezime($ip){
        $this->imePrezime = $ip;
    }
    public function setAdresa($ad){
        $this->adresa = $ad;
    }
    public function getImePrezime(){
        return $this->imePrezime;
    }
    public function getAdresa(){
        return $this->adresa;
    }
    public function ispisiKontakt(){
        echo "Ime i prezime: ", $this->imePrezime;
        echo "Adresa: ", $this->adresa;
    }
}
```

Datoteka: **Kontakt.php**

Konstruktor: Java

Konstruktor u Javi je dio programskog koda koji se **automatski izvodi** tijekom stvaranja i inicijalizacije objekta.

- Naziv konstruktora u Javi je **jednak nazivu klase**.
- Sintaksa konstruktora u Javi je jednaka sintaksi metode, **osim što nema povratnu vrijednost**.

```
class Kontakt {
   String imePrezime;
   public Kontakt(String ip){
      imePrezime = ip;
   }
   …
}
```

Datoteka: **Kontakt2.java**

Konstruktor: Objective-C

- U Objective-C jeziku ulogu konstruktora ima **metoda *init*** koja se **mora eksplicitno pozvati pri inicijalizaciji objekta**.
- Metoda ***init*** vraća objekt bilo kojeg tipa (tip podatka **id**)

```
@implementation Kontakt
   - (id)init:(id)ip{
     imePrezime = ip;
     return self;
   }
   …
}
```

Datoteka: **Kontakt2.m**

Konstruktor: PHP

- Konstruktor u PHP-u je dio programskog koda koji se **automatski izvodi** tijekom stvaranja i inicijalizacije objekta.
- Naziv konstruktora u PHP-u je **__construct**
- Sintaksa konstruktora u PHP-u je jednaka sintaksi metode, osim što nema povratnu vrijednost.

```
class Kontakt {
  private $imePrezime;
  public function __construct($ip){
    $this->imePrezime = $ip;
  }
  …
}
```

Datoteka: **Kontakt2.php**

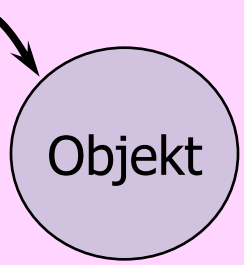
Stvaranje objekta

1. deklariranje varijable kojom će se pristupati objektu. (referenca/pokazivač)

```
// Java
Kontakt susjed = new Kontakt();
// Objective-C
Kontakt * susjed = [[Kontakt alloc] init];
// PHP
$susjed = new Kontakt();
```

U memoriji se osigurava prostor za **varijablu kojom će se pristupati objektu**.

2. Stvaranje i inicijalizacija objekta




```
// Java
Kontakt susjed = new Kontakt();
// Objective-C
Kontakt * susjed = [[Kontakt alloc] init];
// PHP
$susjed = new Kontakt();
```

- Osigurava se **prostor za objekt** u memoriji.
- Inicijaliziraju se varijable objekta.
- Omogućuje se pristup objektu pomoću varijable reference.

Stvaranje i inicijalizacija objekta

Java: ključna riječ new

```
Kontakt susjed = new Kontakt();
```

Datoteka: **Adresar.java**

Objective-C: slanje poruka za alokaciju memorije (alloc) i inicijalizaciju objekta (init)

```
Kontakt * susjed = [[Kontakt alloc] init];
```

Datoteka: **Adresar.m**

PHP: ključna riječ new

```
$susjed = new Kontakt();
```

Datoteka: **Adresar.php**

Pozivanje metode na objektu

Java: operator točka

```
objekt.nazivMetode(vrijednostParametra,…);
susjed.setImePrezime("Pero Perić");
```

Datoteka: **Adresar.java**

Objective-C: slanje poruka objektu

```
[objekt nazivMetode: vrijednostParametra …];
[susjed setImePrezime: @"Pero Perić"];
```

Datoteka: **Adresar.m**

PHP: operator strelica

```
objekt->nazivMetode(vrijednostParametra,…);
$susjed->setImePrezime("Pero Perić");
```

Datoteka: **Adresar.php**

3. DIO: Dodatak

Za samostalni rad u opisanim programskim jezicima pogledajte linkove na sljedećim slajdovima.

Na njima ćete naći

- Adrese za download programskih jezika, alata i razvojnih okruženja.
- Upute za instalaciju.
- Tutoriale za samostalno proširivanje znanja.

Java

Download JDK

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html/

Upute za instalaciju JDK

- **Windows OS**
  http://docs.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html
- **MAC OS X**
  http://docs.oracle.com/javase/7/docs/webnotes/install/mac/mac-jdk.html
- **Linux**
  http://docs.oracle.com/javase/7/docs/webnotes/install/linux/linux-jdk.html

Prva Java aplikacija

http://docs.oracle.com/javase/tutorial/getStarted/cupojava/index.html

Java tutoriali

http://docs.oracle.com/javase/tutorial/

Objective-C

Započnite razvoj aplikacija na Mac platformi

https://developer.apple.com/library/mac/referencelibrary/GettingStarted/RoadMapOSX/chapters/01_Introduction.html

Objective-C na Linux platformi (download i upute)

http://www.gnustep.org/

Objective-C na Windows platformi (download i upute)

http://www.gnustep.org/experience/Windows.html

PHP

XAMPP okruženje za razvoj web aplikacija na Windows, Linux i Mac OS platformama (download i upute)

http://www.apachefriends.org/index.html

Tutoriali za učenje PHP-a (od osnova programiranja do objektnog programiranja)

http://www.codecademy.com/tracks/php

Za one koji žele znati više

- Nasljeđivanje (*inheritance*)
- Učahurivanje (enkapsulacija)
- Overloading
- Polimorfizam

Nasljeđivanje (*inheritance*)

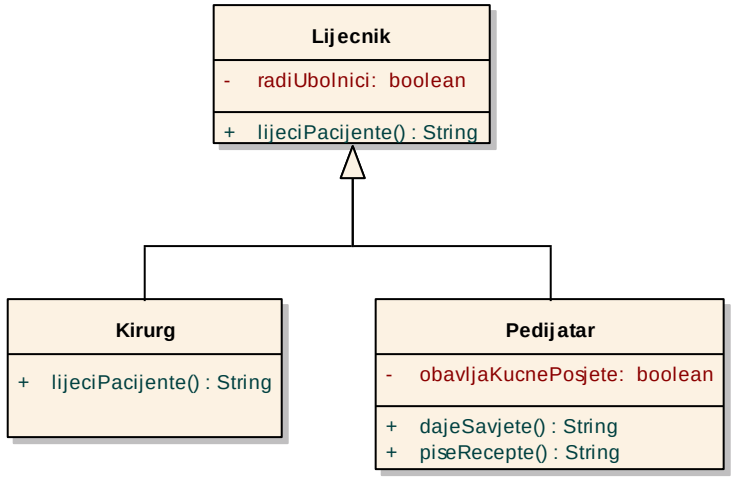
- Prilikom nasljeđivanja kreira se "opća" ili nadređena klasa (*superclass*).
- Podređenim klasama (*subclass*) je ta "opća" klasa nadređena i one nasljeđuju njezina svojstva.
- Strukture i ponašanje zajedničko za više klasa prelazi u nadređenu klasu.
  - Izbjegava se dupliciranje koda u podređenim klasama.
  - Jednosmjerno (podređene klase nasljeđuju ponašanje i svojstva nadređenih).
  - Podređene klase proširuju funkcionalnost nadređene.

Nasljeđivanje (*inheritance*): Java

```
public class Lijecnik {
     boolean radiUBolnici;
     public String lijeciPacijente() {
          //pregledava pacijente
     }
}
```

```
public class Pedijatar extends Lijecnik {
    boolean obavljaKucnePosjete;
    public String dajeSavjete() {
          //daje savjete
    }
    public String piseRecepte() {
          //piše recepte pacijentima
    }
}
```

```
public class Kirurg extends Lijecnik {
    public String lijeciPacijente() {
          //obavlja kirurške operacije
    }
}
```



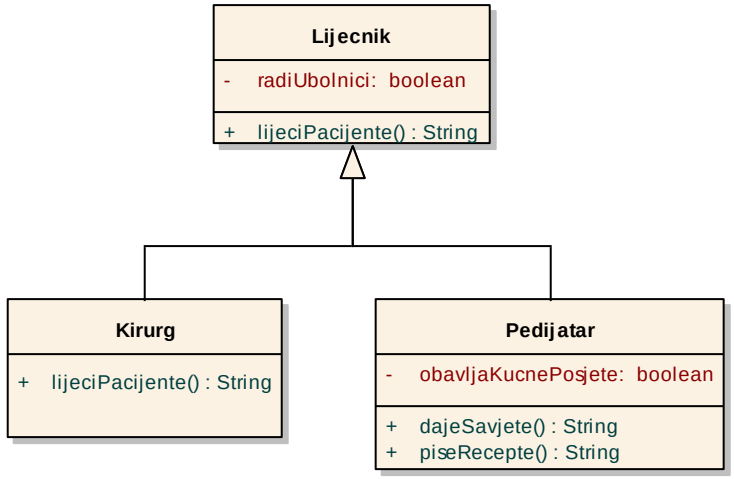

```
public class Pacijent {
  public static void main(String[] a) {
    Pedijatar prvi=new Pedijatar();
    Kirurg drugi=new Kirurg();
    prvi.piseRecepte(); //piše recepte
    prvi.lijeciPacijente(); //pregledava
    drugi.lijeciPacijente(); //kirurški
    }
}
```

Nasljeđivanje (*inheritance*): Objective-C

```
@interface Lijecnik: NSObject {
    BOOL radiUBolnici;
  }
  -(NSString *) lijeciPacijente;
@end
```

```
@interface Pedijatar : Lijecnik {
    BOOL obavljaKucnePosjete;
  }
  -(NSString *) dajeSavjete;
  -(NSString *) piseRecepte;
@end
```

```
@interface Kirurg : Lijecnik
  -(NSString *) lijeciPacijente;
@end
```



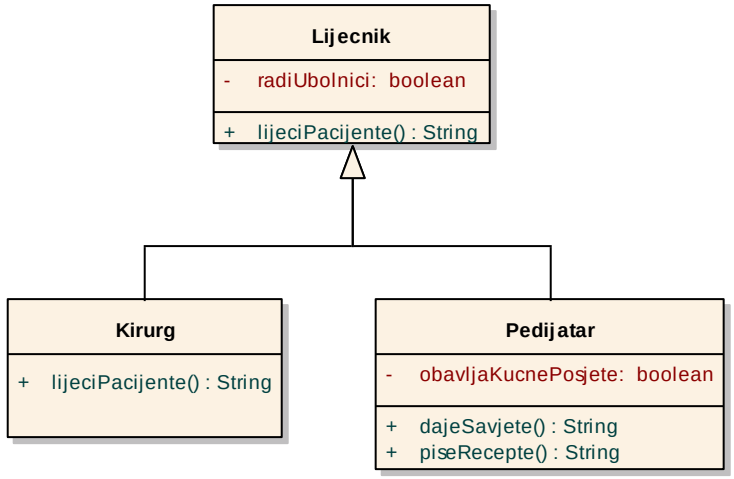

```
int main(int argc, const char * argv[]){
  Pedijatar *prvi = [[Pedijatar alloc] init];
  Kirurg *drugi = [[Kirurg alloc] init];
  [prvi piseRecepte]; //piše recepte
  [prvi lijeciPacijente]; //pregledava
  [drugi lijeciPacijente]; //kirurški
}
```

Nasljeđivanje (*inheritance*): PHP

```php
public class Lijecnik {
     public $radiUBolnici;
     public function lijeciPacijente() {
          //pregledava pacijente
     }
}
```

```php
public class Pedijatar extends Lijecnik {
    public $obavljaKucnePosjete;
    public function dajeSavjete() {
          //daje savjete
    }
    public function piseRecepte() {
          //piše recepte pacijentima
    }
}
```

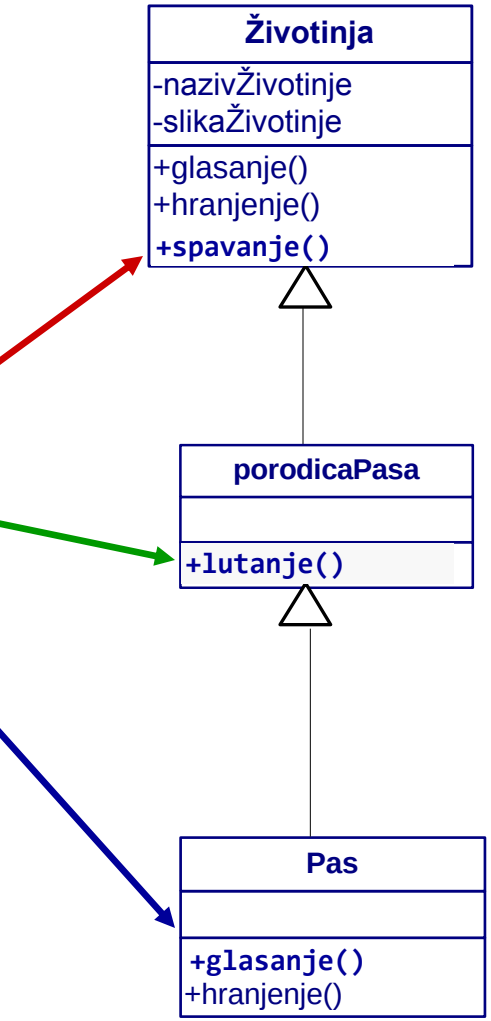
```php
public class Kirurg extends Lijecnik {
    public function lijeciPacijente() {
          //obavlja kirurške operacije
    }
}
```




```php
$prvi = new Pedijatar();
$drugi = new Kirurg();
$prvi->piseRecepte(); //piše recepte
$prvi->lijeciPacijente(); //pregledava
$drugi->lijeciPacijente(); //kirurški
```

Koja metoda se poziva?

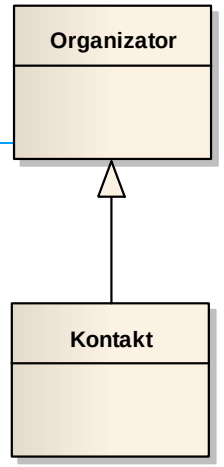
```
Pas p = new Pas();

p.glasanje();

p.lutanje();

p.spavanje();
```



Uvijek se poziva metoda najnižeg stupnja u stablu nasljeđivanja!

Nasljeđivanje (*inheritance*)

UML



Prirodan jezik

Klasa **Kontakt** proširuje klasu **Organizator**

Java

```
class Kontakt extends Organizator {…}
```

Objective-C

```
@interface Kontakt : Organizator … @end
```

PHP

```
class Kontakt extends Organizator {…}
```

Poziv metode nadređene klase

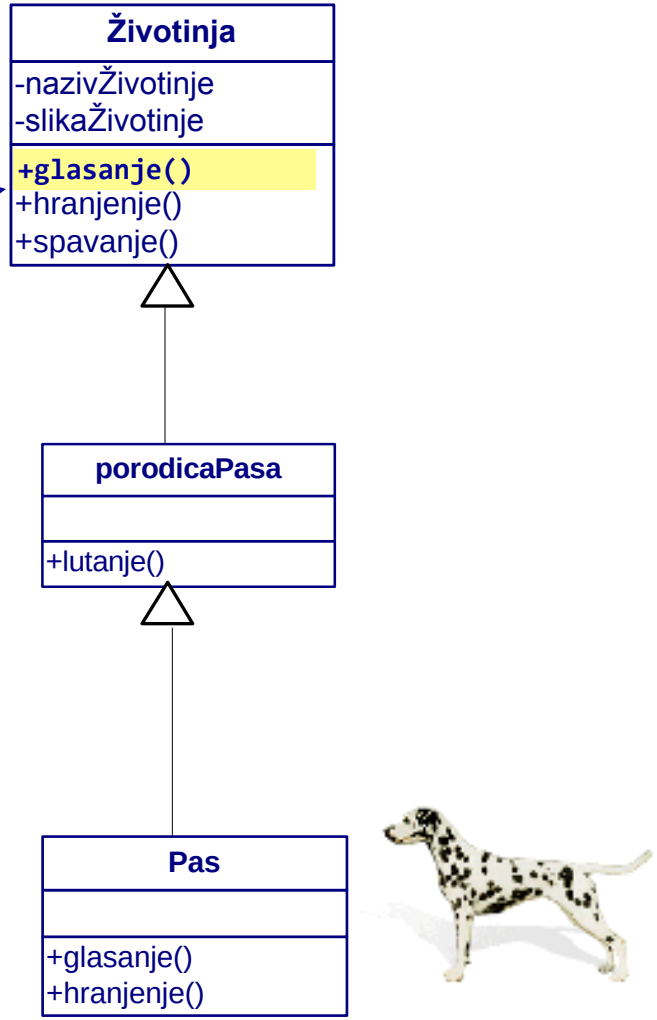
Ključna riječ **super** poziva metodu ili konstruktor **nadređene** klase

```
super.glasanje();
```

```
super();
```

poziva **konstruktor** nadređene klase

- Ako se koristi, super mora biti prva naredba u konstruktoru.
- Obavezno se koristi u konstruktoru ako konstruktor nadređene klase prima parametre



Učahurivanje (enkapsulacija)

Skrivanje detalja implementacije klase

- okolina vidi samo onaj dio sustava koji je bitan za suradnju s njim
- omogućuje izmjenu programske logike klase, a da okolina ništa ne primijeti

Kada kreiramo neku klasu štitimo varijable za koje ne želimo da im "netko drugi" izvana izravno pristupa i mijenja ih

- problem je što nam je ponekad u drugoj klasi potrebna vrijednost tih varijabli
- potrebno je moći postaviti, promijeniti ili pročitati njihovu vrijednost
- ako je varijabla "*private*", to nije izravno moguće

Učahurivanje (enkapsulacija)

```
class BillsCat {
  public int height=27;
}
```

"Sadly, Bill forgot to encapsulate his Cat class and ended up with a flat cat."

```
class Flat {
   BillsCat Flat = new BillsCat();
   Flat.height=0;
}
```

Učahurivanje (enkapsulacija)

1. **varijablama** stavljamo nivo pristupa **private**
   - mogu im pristupiti samo metode iste klase
2. izrađujemo metode za pristup "izvana":
   - postavljanje odgovarajuće vrijednosti varijable (*mutator* ili "**setter**" metoda);
   - čitanje vrijednosti varijable (*accessor* ili "**getter**" metoda).
3. tim **metodama** postavljamo nivo pristupa **public**

Overloading

Korištenje iste metode ili konstruktora sa različitim parametrima

- Npr. u JAVI imamo ugrađenu metodu System.out.print koja na takav način ispisuje podatke bilo kojeg tipa (integer, String, ...)
- bez toga svojstva trebalo bi izraditi složenu metodu koja bi vršila konverziju tipova ili bi trebalo biti po jedna metoda za svaki tip podatka

```
void ispis(int niz) {
        //ispisuje podatke tipa integer
}
void ispis(String niz) {
   // ispisuje podatke tipa String
}
```

Višeobličje (polimorfizam)

Objekt se može promatrati u više oblika

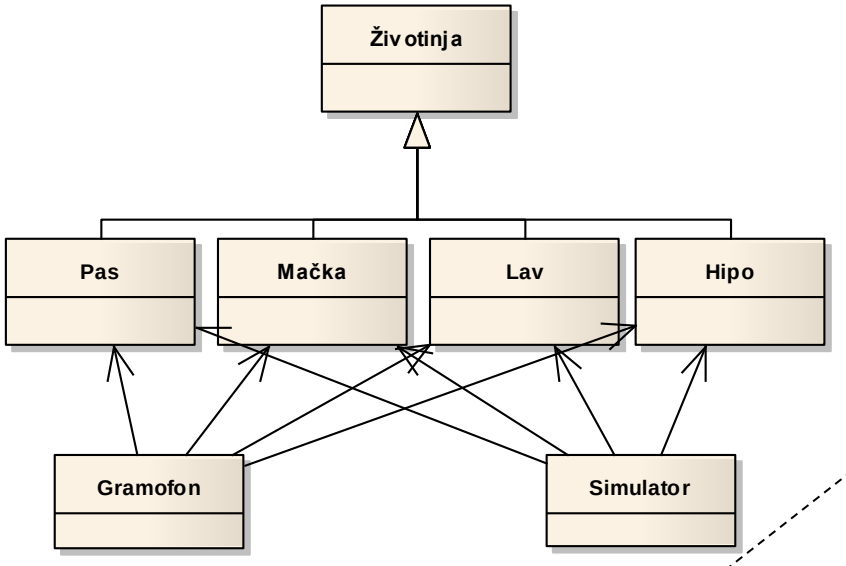
- objekt tipa **Pas** kao **Psa**, ali i kao **Životinju**
- objekt tipa **String** može se promatrati kao **String** ali i kao **Object** ...

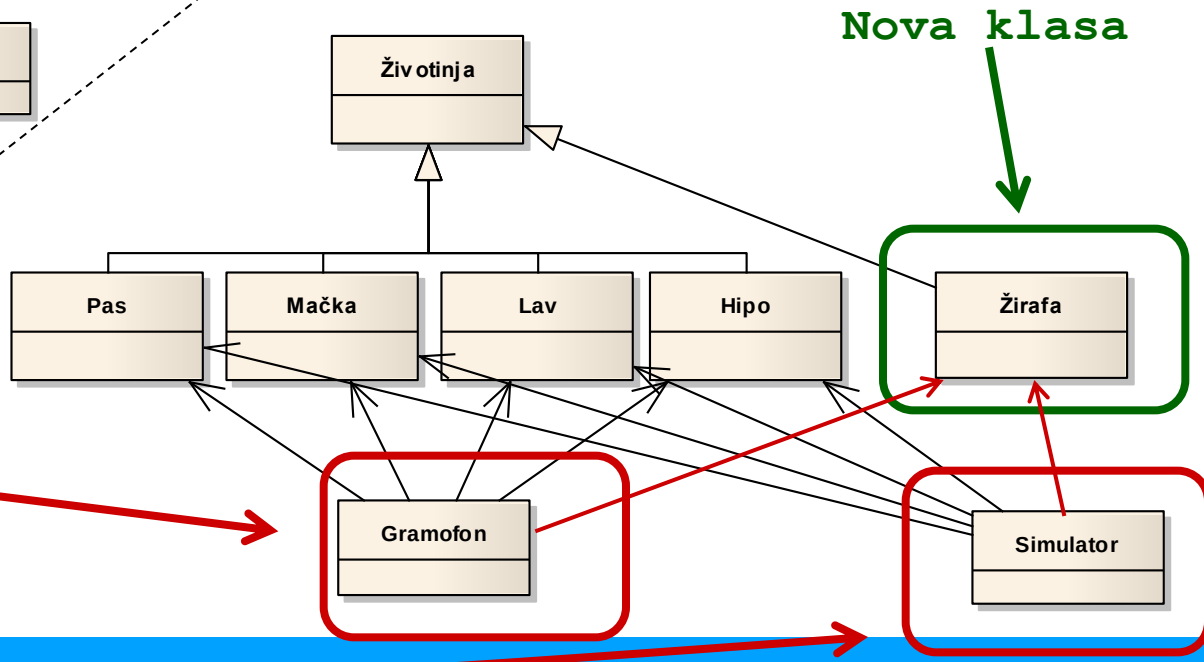
Referenca koja pokazuje na objekt nadređenog tipa može pokazivati i na sve objekte podređenih tipova

Uobičajan način	Polimorfizam
`Pas Kiki = new Pas();`	`Životinja Kiki = new Pas();`
`Mačka Nera = new Mačka();`	`Životinja Nera = new Mačka();`

- varijabla referenca je nadređenog tipa, a kreirani objekt podređenog
- varijabla referenca nadređenog tipa može pokazivati na objekte bilo kojeg podređenog tipa

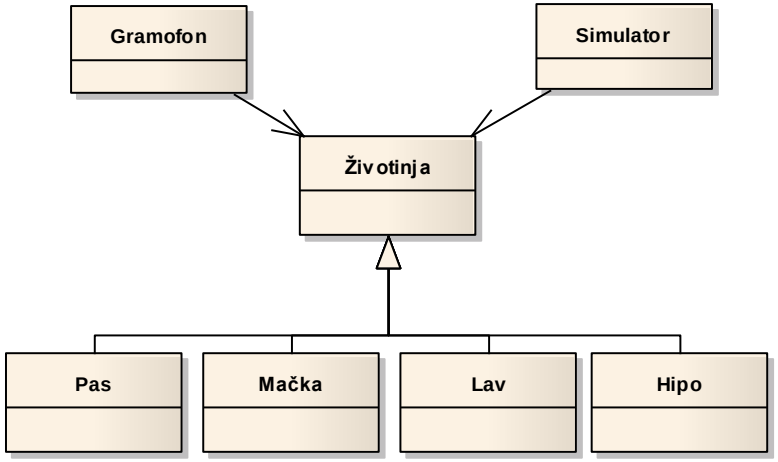
Bez korištenja polimorfizma



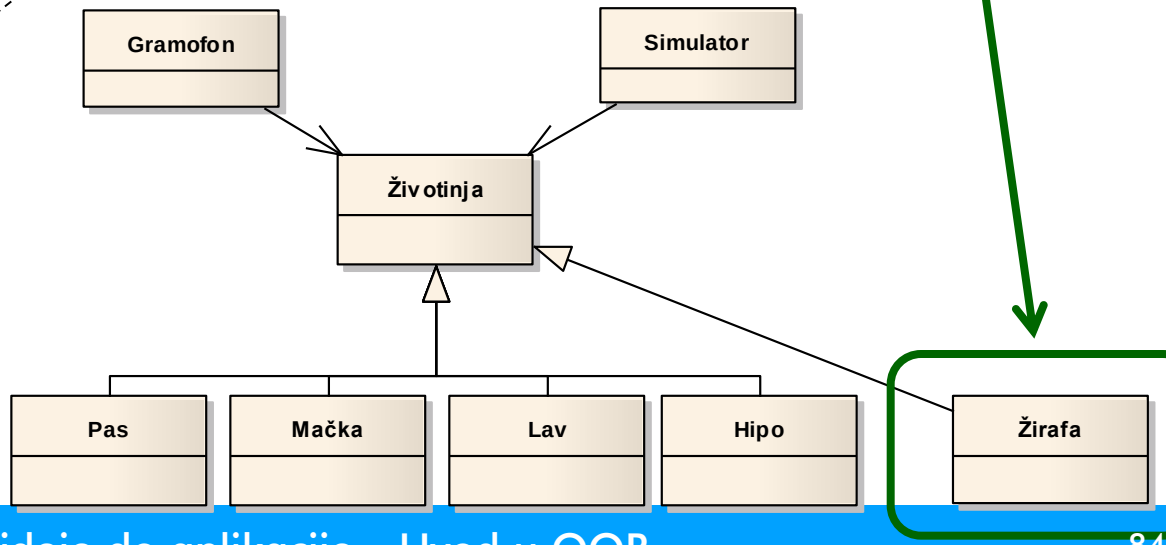


Kada se kreira nova klasa, izmjenu je potrebno vršiti u svim klasama koje koriste novo kreiranu klasu.

Uz korištenja polimorfizma



Nova klasa



Kada se kreira nova klasa, nije potrebno raditi nikakve izmjene.