OBJEKTNO-ORIJENTISANO PROGRAMIRANJE

2016/17

PRIRODNO-MATEMATIČKI FAKULTET, UNIVERZITET U KRAGUJEVCU

OOP 2016/17

Predavanja

- ponedeljkom 14.15-17.00

Vežbe

- Ponedeljkom 17.15-20.00 I grupa
- Utorkom 17.15-20.00 II grupa

OOP 2016/17

Poeni

50 + 50 (30+10+10)

2 kolokvijuma + završni (usmeni + seminarski)

I kolokvijum	sredina aprila
I kolokvijum POPRAVNI	kraj aprila/početak maja
II kolokvijum	druga polovina maja
II kolokvijum POPRAVNI	kraj maja/početak juna

Literatura

D. Poo, D. King, S. Ashok, *Object-oriented programming in Java*, Springer-verlag, 2008.

I. Horton, *Java2 - JDK 1.5*, CET, Beograd, 2006.

UVOD

APSTRAKCIJA PODATAKA

MOTIVACIJA

- **Problemi u razvoju softvera:**
  - Zahtevi korisnika su složeni i stalno se povećavaju. Softverski sistemi su složeni
  - Uvek je potrebno povećati produktivnost proizvodnje softvera. Kako? Povećanjem broja programera u timu? Problemi – interakcija između delova softvera!
  - Način povećanja produktivnosti – ponovna upotreba softvera (software reuse). Kako obezbediti?
  - Problemi održavanja softvera: ispravljanje grešaka, promena zahteva i dodavanje zahteva. Kako postići?

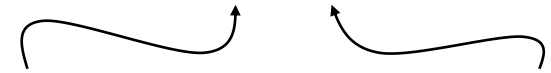
- **Kako odgovoriti na izazove? Unapređenjem koncepata!**

SAGA OOP

Sta je to tip podataka, a sta struktura?

Tip podataka je odredjen (konacnim) **skupom vrednosti** i nekim **paketom operacija i relacija** nad elementima tog skupa.
Pr. ({-32768, ..., -1, 0, 1, ..., 32767}, +, -, * div, mod, =, <)

Struktura podataka je konglomerat podataka. Ona se formira od drugih (jednostavnijih) struktura i od tipova. Jedine operacije koje se nad strukturama mogu obavljati su **operacije selekcije**.
Pr. var **a**: **array** [1 .. 100] of real;



Metod struktuiranja ima sopstvene operacije selekcije.
Pr. **array** indeksiranje (a[i]), a **record** projektovanje (odabir polja sloga, a.ime) - PASCAL

Pascal:
eksplicitni metodi struktuiranja (array, record, record-case)
implicitni (preko pokazivaca).

SAGA OOP

Zasto je važno praviti nove tipove?

Za uspešan rad u velikim timovima i na velikim projektima bitno pisati apstraktan kood.

NPR. Za projekat u kome je potrebno raditi nesto sa matricama: napraviti tip MATRIX (struktura MATRIX snabdevena operacijama za rad sa matricama), pa kada dodje do rešavanja konkretnog problema, samo pozivati gotove (i testirane) procedure.

```
C := Inv(A) * Transpose(B) + Det(Y) * Adj(X)
```

```
Inv(A, A1);
Transpose(B, B1);
MatMul(A1, B1, P);
Adj(X, X1);
ScalMul(Det(Y), Q);
MatAdd(P, Q, C);
```

```
for i := 1 to n do
        for j := 1 to n do
            (* rutina koja invertuje matricu A u A1 *);
    for i := 1 to n do
        for j := 1 to n do
            B1[i,j] := B[j, i];
    for i := 1 to n do
        for j := 1 to n do
            (* rutina koja mnozi A1 i B1 *);
    (* itd *)
```

SAGA OOP

Dakle, dizajnirati odgovarajuce strukture podataka, napisati procedure za manipulaciju tim strukturama podataka.

Tako dolazimo do jednog viseg nivoa apstrakcije u programima. U trenutku kada je potrebno primeniti neke operacije na nekim strukturama, ne interesuje nas *kako* procedure rade, vec *sta* rade.

Upotreba tipa

```
var a, b: MATRIX;
begin
   NewMatrix(a); NewMatrix(b);
   ReadMatrix(a);
   Transpose(a, b);
   WriteMatrix(b);
   DisposeMatrix(a);
   DisposeMatrix(b)
end;
```

Implementacija

```
type MATRIX = array [1 .. X, 1 .. Y] of real;
```

```
type MATRIX = ^MatrixEntry;
     MatrixEntry =  record
                      i, j: integer;
                    entry: real;
                    right, down:
  MATRIX
                    end;
```

NIJE BITNA IMPLEMENTACIJA, VEĆ MANIFESTACIJE (TJ. OSOBINE) OPERACIJA DATOG TIPA PODATAKA.

SAGA OOP

Osnovna ideja OOP-a - Abstract data type - ADT

Struktura podataka → Tip podatka data abstraction
ADT → Apstakcija + Skrivanje podataka information hiding

Apstraktni tip podataka je tip podataka čiju implementaciju ne znamo (primer: ne znamo da li su matrice predstavljene sa array ili preko pokazivaca), ali znamo kako se ponašaju operacije nad vrednostima tog tipa, tako da pišemo program koristeci samo osobine operacija.
Dakle, apstrahovana je jedna dimenzija problema: implementacija tipa (apstrahovati znaci izbaciti iz posmatranja ono sto u tom trenutku nije bitno).

Za pojam ADT neraskidivo vezan pojam sakrivanja informacije (information hiding). Programeru se da ime tipa i paket procedura. Implementacija tipa se SAKRIJE da je on ne vidi.

OBJEKTI

UVOD

SLOŽENI TIPOVI I PRIMERCI

```
var m: MATRIX;
...
m.init(9,10);
```

- MATRIX – složeni tip
- m - objekat /primerak tipa

podaci = **stanje objekta**

opisuju stanje

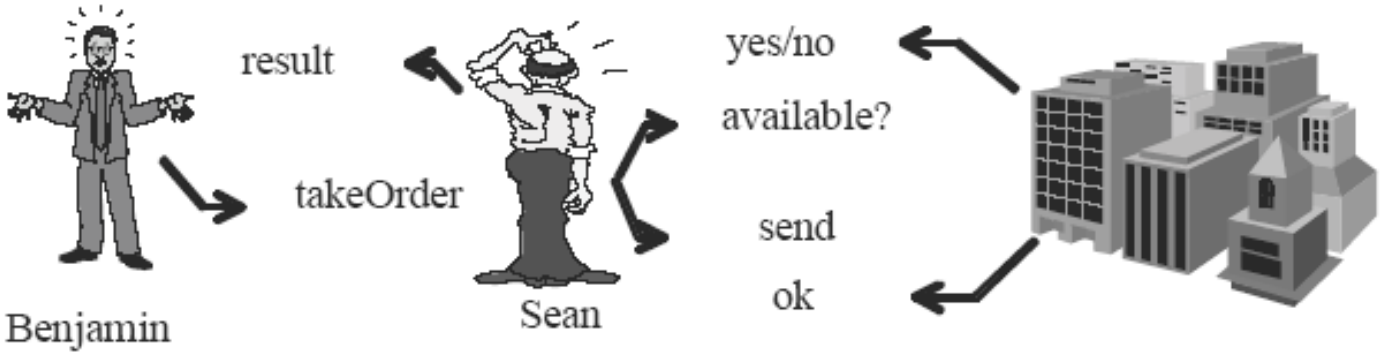
+

procedure = **metodi**

} **objekat**

- **pripadaju objektu**
- **menjaju stanje objekta**
- **njima se opisuje tzv. ponašanje objekta**

PRIMER – OBJEKTI U REALNOM SVETU



Sean as an Object

Attributes:

name = Sean

Methods:

takeOrder() {

check with warehouse on stock availability

check with warehouse on delivery schedule

if ok

then {instruct warehouse to deliver stock(address, date)

return ok}

else return not ok

}

Benjamin as an Object

Attributes:

name = Benjamin

address = 1, Robinson Road

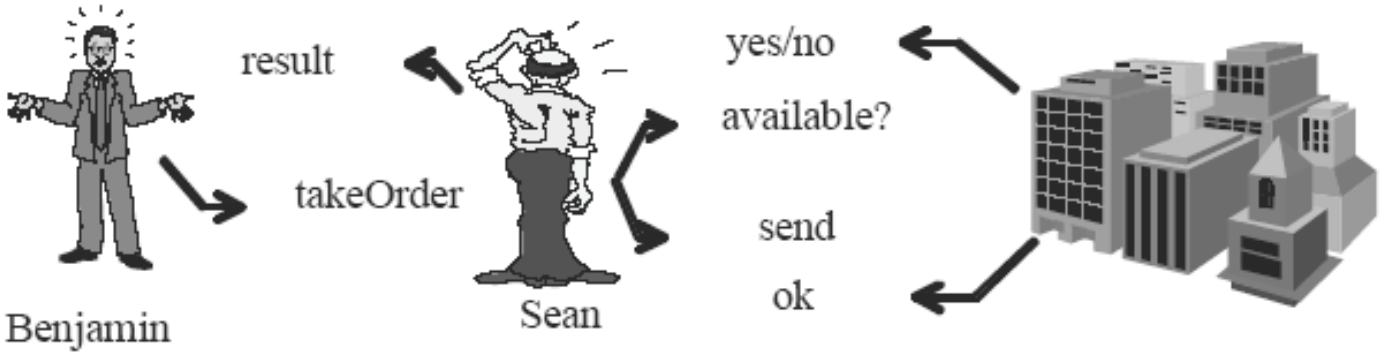
budget = 2000

Methods:

purchase() {send a purchase request to a salesperson}

getBudget() {return budget}

PRIMER – OBJEKTI U REALNOM SVETU



Sean as an Object

Attributes:

name = Sean

Methods:

takeOrder() {

check with warehouse on stock availability

check with warehouse on delivery schedule

if ok

then {instruct warehouse to deliver stock(address, date)

return ok}

else return not ok

}

Benjamin as an Object

Attributes:

name = Benjamin

address = 1, Robinson Road

budget = 2000

Methods:

purchase() {Sean.takeOrder("Benjamin", "sofa", "1, Robinson Road","12 November")}

getBudget() {return budget}

PRIMER – OBJEKTI U REALNOM SVETU

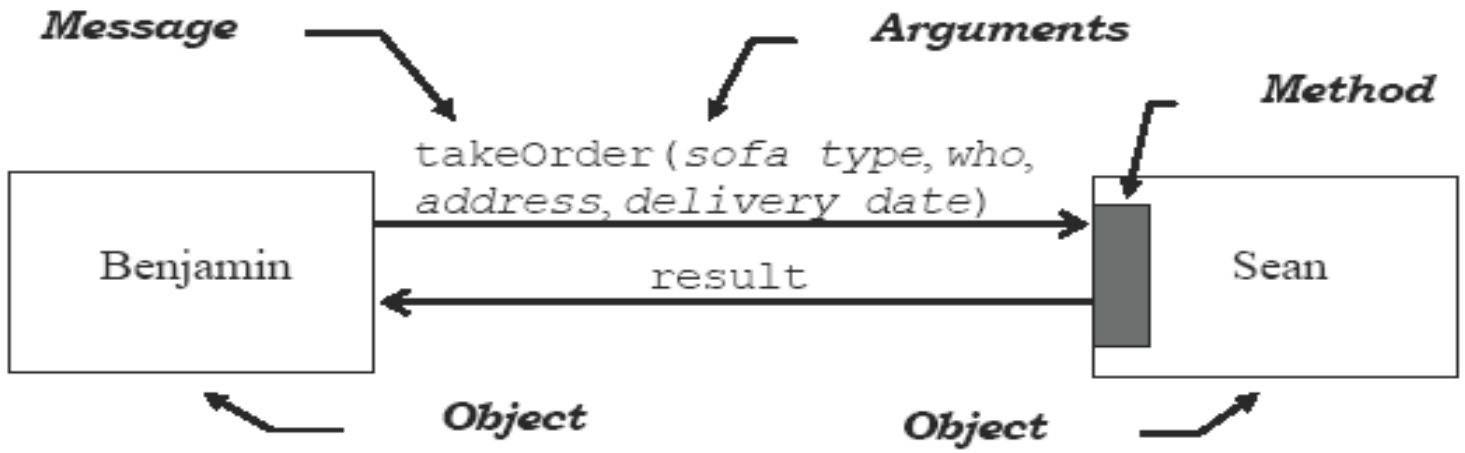
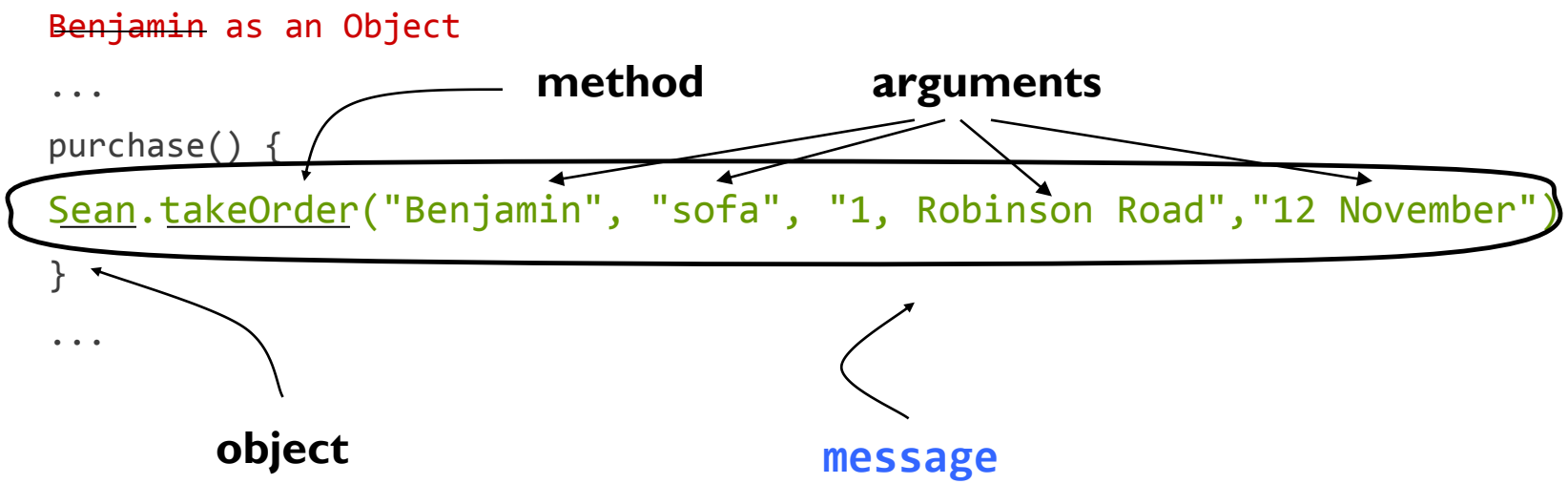


Figure 1-2: Object interactions in object-oriented programming terms.

PRIMER – OBJEKTI U REALNOM SVETU

message = object + method + arguments



OBJEKTNO V.S. STRUKTURNO/PROCEDURALNO

UVOD

OO VS STRUKTURNO MODELOVANJE

- **Strukturno modelovanje**
  - Modelovanje se vrši analizom problema i razbija na manje celine;
  - Podaci se smeštaju u strukture koje često nisu slika realnog sveta.

- **Objektno-orijentisano modelovanje**
  - Modelovanje se vrši tako što se razmatra kompletan sistem u kome obavlja posao;
  - Uočavaju se učesnici i aktivnosti koje oni znaju da obavljaju sa dostupnim podacima (sakrivanje podataka);
  - Program se formira određivanjem redosleda kojim se poziva izvršavanje aktivnosti.

PRIMER: OO VS STRUKTURNO MODELOVANJE

- U jednoj stambenoj zgradi na svakom od *m* spratova (*m*<10) ima po n stanova (*n*<10). Napisati program za pomoć u radu Kućnog saveta na sledeći način:
  - Za svaki stan je poznat broj i površina stana, ime vlasnika, starost vlasnika i njegov radni status (nezaposlen, zaposlen, penzioner);
  - Napisati potprogram koji učitava podatke o stanovima i njihovim vlsnicima, od prvog ka poslednjem spratu, i numeriše stanove tako da prva cifra predstavlja sprat na kome se stan nalazi, a druga redni broj stana na spratu;
  - Napisati potprogram koji vraća ime predsednika Kućnog saveta, ako se zna da se za predsednika bira najmlađi penzioner u zgradi;
  - Na sednici Kućnog saveta odlučeno je da pet najmlađih nezaposlenih stanara obavi krečenje zgrade. Napisati potprogram koji određuje stanare koji će učestvovati u krečenju;
  - Korišćenjem napisanih potprograma iz glavnog programa učitati podatke o stanovima i njihovim vlasnicima, numerisati stanove, odštampatti ime predsednika Kućnog saveta i imena stanara koji učestvuju u akciji krečenja.

STRUKTURNO MODELOVANJE

- Uočavaju se aktivnosti koje treba da se izvrše, a potom se na osnovu uočenih aktivnosti formiraju procedure za izvršavanje
  - Unos, Predsednik, Krečenje
- Podaci se čuvaju u strukturi koja čuva potrebne podatke

```
stan=record
     broj:string;
     povrsina:integer;
     ime:string[6];
     godiste:integer;
     posao:status;
end;
```

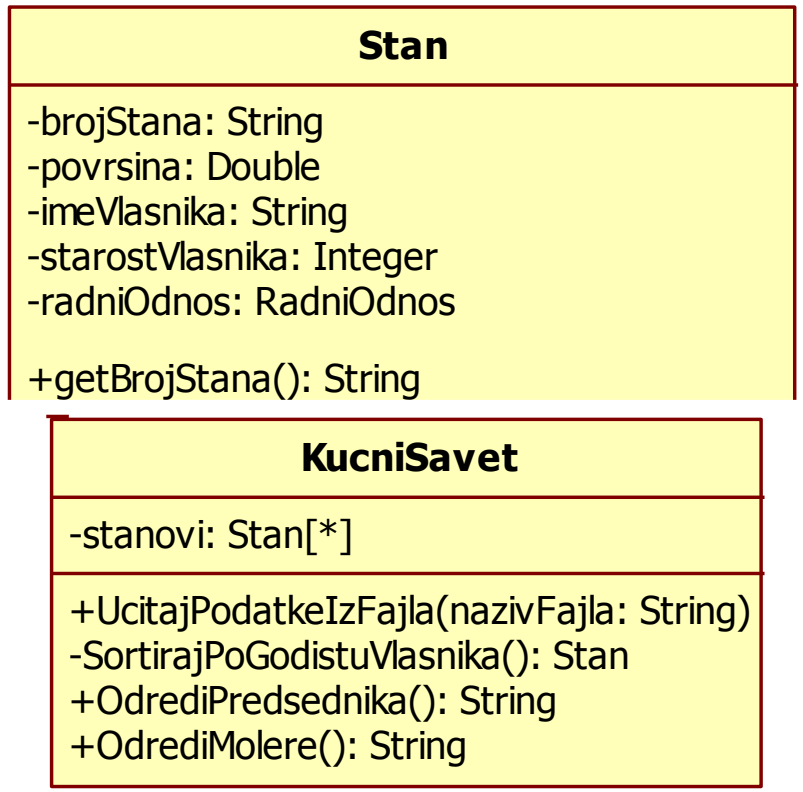
- U glavnom delu programa treba odrediti redosled pozivanja napisanih procedura

```
begin
  unos(a,m,n);
  sortg(a,b,k,m,n);
  predsednik(b,k);
  krecenje(b,k);
end.
```

OBJEKTNO-ORIJENTISANO MODELOVANJE

- Uočavaju se objekti koji učestvuju u rešavanju problema u realnom svetu i aktivnosti koje izvršava svaki od tih objekata

  - Kućni savet
    sadrži podatke o stanovima,
    od aktivnosti ima učitavanje podataka,
    izbor predsednika određivanje učesnika krečenja
  - Stan
    sadrži podatke o jednom stanu
    pruža uvid u te podatke



ZAŠTO OOP?

```
public class OOP_primer {
        public static void main(String[] args) {
        KucniSavet kucniSavet = new KucniSavet();
        KucniSavet.UcitajPodatkeIzFajla("UlazniPodaci.txt");
                System.out.println("Predsednik saveta je: " + kucniSavet.OdrediPredsednika());
                System.out.println("Moleri su:");
        String[] moleri = kucniSavet.OdrediMolere();
        for (int i = 0; i < moleri.length; i++) {
            System.out.println(moleri[i]);
        }
    }
}
```

OOP VS. PROCEDURALNO PROGRAMIRANJE

- Standardno, **proceduralno** programiranje (npr. C):

  Program započinje izvršavanjem funkcije main koja izvršava postavljeni zadatak pozivanjem drugih funkcija. Program završava kad se izvrše sve instrukcije funkcije main. Osnovni **gradivni blok programa je, dakle, funkcija**. Postavljeni zadatak se rešava tako da što se razbije na niz manjih zadataka od kojih se svaka može implementirati u jednoj funkciji, tako da je program niz funkcijskih poziva.

- U **objektno-orijentisanom** programiranju

  Osnovnu ulogu imaju **objekti koji sadrže i podatke i funkcije** (metode). Program se konstruiše kao **skup objekata koji međusobno komuniciraju**. Podaci koje objekat sadrži predstavljaju njegovo stanje, dok pomoću metoda on to stanje da može menja i komunicira sa drugim objektima.

OOP VS. PROCEDURALNO PROGRAMIRANJE

Centralni koncepti

- Nasleđivanje
- Učaurivanje (information hiding)

code reuse

- lakše uvezivanje postojećih rešenja
- lakša nadogradnja
- lakše lociranje grešaka

- vaše greške su samo vaše
- lepa posledica - vaše dvorište će biti vidljivo samo u meri u kojoj vi to budete dopustili, sa komšijom vam je zajednička samo ograda

jeftinije i jednostavnije

modelovanje je odvojeno od implementacije

JAVA

ZDRAVO, SVETE!

JAVA

Just Another Vague Acronim - JAVA

1991. tvorac Jave – James Gosling, Sun Microsystems

Stvorio jednostavan, platformski nezavistan jezik

Namenjen pokretanju elektronskih uređaja (Interaktivna TV,

inteligentne rerne, telefoni,..)

1994. Java se ugrađuje u web browser WebRunner

1995. Java se lansira na SunWorld-u,

obavljuje se kod i dokumentacija Jave na Internetu

1996. Sun razvija JDK 1.0

1997. Pojavljuje se JDK 1.1

1999. Pojavljuje se JDK 1.2

Java 2 SDK - Software Development Kit

2000. Pojavljuje se JDK 1.3.

2002. Pojavljuje se JDK 1.4.

2004. Pojavljuje se JDK 1.5.

2006. Sun objavljuje veliki deo Jave kao slobodan i otvoren kod pod GPL licencom

- 2010. Java postaje vlasništvo Oracle-a

OSNOVNE KARAKTERISTIKE JEZIKA

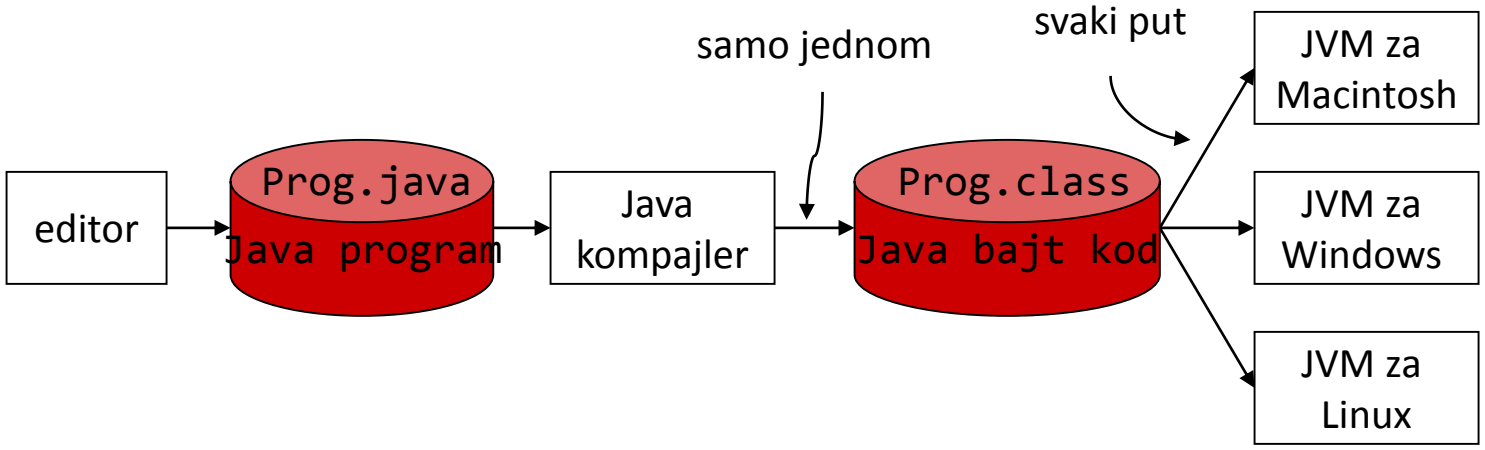
- **Objektna orijentacija**
  - podržava sve koncepte objektno orijentisanog programiranja
  - sintaksa slična C++, OO model jednostavniji
- **Prenosivost**
  - Java programi se prevode u byte kod koji nije mašinski jezik nijednog konkretnog računara, već se izvršava na JVM (Java Virtuelna Mašina)
- **Sigurnost**
  - JVM pruža zaštitu od virusa koji bi se prenosili kroz izvršni kod
- **Robusnost**
  - Stroga provera tipova, proveravani izuzeci, sakupljanje đubreta
- **Efikasnost**
  - JIT (Just In Time) prevodioci

KOMPAJLERI I INTERPERTERI

- Program pisan u nekom od viših programskih jezika potrebno je prevesti na mašinski jezik, ne bi li bio izvršen. To prevođenje vrši **kompajler** (compiler) odgovarjućeg programskog jezika. Nakon što je program jednom preveden, program u mašinskom jeziku se može izvršiti neograničen broj puta, ali, naravno, samo na određenoj vrsti računara.
- Postoji alternativa. Umesto kompajlera, koji odjednom prevodi čitav program, moguće je koristiti **interpreter**, koji prevodi naredbu po naredbu prema potrebi. Interpreter je program koji se ponaša kao CPU s nekom vrstom dobavi-i-izvrši ciklusa. Da bi izvršio program, interpreter radi u petlji u kojoj uzastopno čita naredbe iz programa, odlučuje šta je potrebno za izvršavanje te naredbe, i onda je izvršava (oni se mogu koristiti za izvršavanje mašinskog programa pisanog za jednu vrstu računara na sasvim različitom računaru).

JAVA KOMPAJLER I INTERPERTER

- Projektanti Jave su se odlučili za upotrebu kombinacije **kompajliranja** i interpretiranja. Programi pisani u Javi se prevode u mašinski jezik virtuelnog računara, tzv. **Java Virtual Machine**.
- Mašinski jezika za Java Virtual Machine se zove **Java bytecode**.
- Sve što je računaru potrebno da bi izvršio Java bajt kod jeste **interpreter**. Takav interpreter oponaša Java virtual machine i izvršava program.



HELLO, WORLD!

```
// HelloWorld.java
public class HelloWorld {
    public static void main(String[ ] args) {
        System.out.println("Hello, World");
    } // kraj main metode
 } // kraj klase HelloWorld
```

- svaka Java aplikacija mora sadržati barem jednu klasu s metodom
  - **`main(String[] args)`**
  - počinje svoje izvršavanje pozivom metoda **main**
  - ovako napisan program se prevodi izvršavajući
    - **`javac HelloWorld.java`**
  - Ako nema grešaka prevodilac javac kreira datoteku **`HelloWorld.class`** koja sadrži bytecode instrukcije za JVM.
  - A pokreće se pozivom JVM uz prosledjivanje bajtkoda
    - **`java HelloWorld`**