OBJEKTNO PROGRAMIRANJE 2

Oznaka predmeta: OP2

Predavanje broj: 01

Nastavna jedinica: JAVA

Nastavne teme:

Uvod u programski jezik JAVA, JVM, osnovni koncepti, tipovi podataka, klase i objekti, prevođenje i pokretanje programa, reference na objekte, operatori, kontrola toka programa, inicijalizacija objekata, uništavanje objekata, metode i njihovi parametri, ključna reč final, ključna reč static, nizovi, višedimenzionalni nizovi .

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Eckel B., *Thinking in Java*, 2nd edition, Prentice-Hall, New Jersey 2000.

Cay S. Horstmann and Gary Cornell*: "Core Java, Advanced Features", Vol. 2, Prantice Hall, 2013.*

The Java Tutorial, Sun Microsystems 2001. *http://java.sun.com*

Branko Milosavljević, Vidaković M, *Java i Internet programiranje*, GInT, Novi Sad 2002.

JVM – Java virtualna mašina

- Specifikacija Jave obuhvata dve celine:
  - specifikaciju programskog jezika Java;
  - specifikaciju Java virtuelne mašine (JVM).
- Specifikacija programskog jezika Java se ne razlikuje mnogo od sličnih specifikacija za druge jezike slične namene.
- JVM specifikacija predstavlja novinu u odnosu na druge objekt-orijentisane programske jezike opšte namene.
- JVM specifikacija predstavlja specifikaciju platforme za izvršavanje Java programa u čijoj osnovi se nalazi programski model izmišljenog procesora. Programi napisani u programskom jeziku Java se prevode za ovakvu platformu za izvršavanje.
- Prevedeni programi se ne mogu pokretati direktno na nekoj konkretnoj raunarskoj platformi.
- Potreban je poseban softver koji će takav prevedeni program da prilagodi konkretnoj mašini i operativnom sistemu.
- Potreban je odgovarajući interpreter.

JVM – Java virtualna mašina

- Kompanija koja je vlasnik jezika Java, *Sun Microsystems*, je stavila u javno vlasništvo JVM interpreter, kompajler i skup drugih razvojnih alata grupisanih u paket pod nazivom *Java Development Kit* (JDK).
- U pitanju su alati koji se pokreću iz komandne linije i nude samo osnovni set funkcija za razvoj softvera.
- Sun je izdao JDK paket za nekoliko različitih platformi: *Windows*, *Solaris/ SPARC*, *Solaris/Intel* i *Linux/Intel*.
- Kako je Java specifikacija (jezik i JVM) javno dostupna, drugi proizvođači su proizveli svoje implementacije Jave za različite platforme.
- Na primer, IBM nudi svoje verzije implementacije za većinu svojih hardversko/softverskih platformi, ali i za *Linux* na *Intel* mašinama.
- Iako se najčešće programski jezik Java i Java virtuelna mašina pominju u paru, kao dve komplementarne specifikacije, nema prepreka da se Java kod prevodi i za izvršavanje na nekoj drugoj platformi (*TowerJ*).

JVM – Java virtualna mašina

- Java je kombinacija programskog jezika i platforme za izvršavanje programa koja ima nekoliko važnih osobina:
  - Projektovana je tako da što manje zavisi od karakteristika konkretnog računarskog sistema na kome se izvršava;
  - Jednom napisan i *preveden* program se može pokretati na bilo kojoj platformi za koju postoji odgovarajući JVM interpreter. Dakle, prenosivost programa je garantovana na nivou izvršnog (prevedenog) koda;
  - Java je interpreterski jezik, što ima odgovarajući efekat na brzinu izvršavanja programa.

- Proizvod prevođenja izvornog Java koda je program predviđen za rad u okviru JVM, koji se često naziva bajt-kod (*byte-code*).

Programski jezik JAVA

- Iako je JVM sastavni deo specifikacije, o njoj se govori veoma retko, praktično je koriste samo autori kompajlera i JVM interpretera za konkretne računarske platforme.
- Sa druge strane, većina Java programera govori o drugom delu Java specifikacije, samom programskom jeziku Java.
- Može se reći da je Java objekt-orijentisani programski jezik opšte namene, posebno pogodan za pisanje konkurentnih, mrežnih i distribuiranih programa.
- Referentna dokumentacija za Javu nalazi se na sajtu firme *JavaSoft* (ogranak firme *Sun Microsystems*) *http://java.sun.com*
- Dosta knjiga o Javi npr. *Thinking in Java*, (autor Bruce Eckel) dostupna je u elektronskom obliku koji je besplatan na *http://www.bruceeckel.com*

Osnovni koncepti, tipovi podataka

- Sintaksa Jave slična je sintaksi jezika C++ (sintaksna pravila biće prikazana u samim programskim primerima).
- Java operiše sa dve vrste tipova podataka:
  - primitivnim tipovima;
  - objektima.
- U Javi je definisano *osam* primitivnih tipova podataka (tabela 1.1):
  - celi brojevi – ova grupa obuhvata *byte*, *short*, *int* i *long*;
  - brojevi u pokretnom zarezu – obuhvata *float* i *double* koji su namenjeni realnim vrednostima;
  - znakovi – tip *char* koji je namenjen simbolima u skupu znakova;
  - logičke vrednosti – tip *boolean*, za predstavljanje vrednosti tačno/netačno (*true*/*false*).

- ***Tabela 1.1. Primitivni tipovi***

Primitivni tip	Veličina	Minimum	Maksimum
boolean	1-bit	-	-
char	16-bit	Unicode 0	Unicode $2^{16}-1$
byte	8-bit	-128	+127
short	16-bit	-2^{15}	$+2^{15}-1$
int	32-bit	-2^{31}	$+2^{31}-1$
long	64-bit	-2^{63}	$+2^{63}-1$
double	64-bit	1,7e-308 IEEE754	1,7e+308 IEEE754
float	32-bit	3,4e-038 IEEE754	3,4e+038 IEEE754
void	-	-	-

Primitivni tipovi

- Iz tabele 1.1 se vidi da Java raspolaže primitivnim tipovima koji su na isti način definisani i u drugim programskim jezicima.
- Obratiti pažnju na tip char, koji zauzima dva bajta, umesto uobičajenog jednog bajta.
  - Radi se o tome da se tipom char može predstaviti svaki karakter definisan Unicode standardom koji definiše kodni raspored koji obuhvata sve današnje zvanične jezike;
  - To znači da su Java programi u startu osposobljeni da rade sa višejezičnim tekstom, ili u našim uslovima, ravnopravno sa srpskom latinicom i srpskom ćirilicom.
- Takođe, trebalo bi obratiti pažnju da je tip boolean 1-nobitni.
- String, kao često korišćen tip podatka, nema odgovarajući primitivni tip u Javi.

Klase i objekti

- Druga vrsta podataka sa kojima operiše Java program su objekti.
- Objekti predstavljaju osnovni koncept objektno-orijentisanog razmišljanja pri modelovanju sistema.
  - Svaki objekat realnog sistema koga posmatramo predstavljamo odgovarajućim objektom koji je sastavni deo modela sistema.
  - Objekte koji zajednike osobine (ne moraju imati iste vrednosti tih osobina) možemo da opišemo klasom.
  - Objekat je jedna instanca (primerak) svoje klase.
  - Klasa predstavlja model objekta, koji obuhvata atribute i metode.
- Primer Java klase:

```
class Automobil {
  boolean radi;
  void upali() {
        radi = true;
  }
  void ugasi() {
        radi = false;
  }
}
```

- Plavom bojom su navedene ključne reči jezika. Klasa ima naziv *Automobil*, definiše jedan atribut koji se zove *radi* i logičkog je tipa (*boolean*). Definiše dve metode koje se mogu pozvati nad objektima te klase, metode *upali* i *ugasi*.

Klase i objekti

- Kreiranje objekata koji predstavljaju instance (primerke) ove klase može se obaviti na sledeći nain:

```
Automobil a = new Automobil();
Automobil b = new Automobil();
```

- Ovim su kreirana dva objekta klase *Automobil*, koji su nazvani *a* i *b*. Atributu *radi* objekta *a* može se pristupiti pomoću:

```
a.radi
```

a poziv metoda *upali* i *ugasi* mogao bi da izgleda kao u sledećem primeru:

```
a.upali();
b.ugasi();
```

- Neke od osobina Jave koje je bitno razlikuju u odnosu na C++ su:
  - Nije moguće definisati promenljive i funkcije izvan neke klase. Samim tim, nije moguće definisati globalne promenljive, niti globalne funkcije ili procedure.
  - Ne postoje odvojene deklaracija i definicija klase. Java poznaje samo definiciju klase. Prema tome, ne postoje posebni _header_ fajlovi koji sadrže deklaraciju klase.

Klase i objekti

- Kako Java ne dopušta postojanje bilo čega što bi postojalo izvan neke klase, postavlja se pitanje odakle počinje izvršavanje Java programa.
- Java koristi funkciju *main*, samo što i ta funkcija mora biti metoda neke klase. C i C++ koriste funkciju *main* kao osnovnu funkciju od koje počinje izvršavanje programa.
- Izgled jedne klase koja sadrži metodu main, i predstavlja primer jednog izvršivog Java programa dat je u sledećem primeru:

```
class Hello {
   public static void main(String args[]) {
      System.out.println("Hello world!");
   }
}
```

- Kompletan tekst ove klase smešten je u datoteku *Hello.java*. Dakle, njena ekstenzija je obavezno *.java*, a ime mora biti jednako imenu klase, uključujući i razliku između velikih i malih slova.
- Standardna preporuka je da se svaka klasa programa smešta u posebnu datoteku. Dakle, svakoj Java klasi odgovara jedan fajl sa identičnim nazivom i ekstenzijom *.java*.

Prevođenje i pokretanje programa

- Svaka Java klasa se može prevesti nezavisno od ostalih elemenata programa.
- Komanda kojom se klasa *Hello* korišćenjem alata iz standardnog JDK paketa iz prethodnog primera prevodi je:

```
javac Hello.java
```

- Dakle, kompajler se poziva komandom javac, a kao parametri navode se imena onih datoteka koje želimo da prevedemo (može ih biti više, i mogu se koristiti džoker-znaci).
  - Navođenje ekstenzije datoteke je obavezno, iako je ekstenzija uvek *.java*.
- Prevođenjem datoteke *Hello.java* dobija se datoteka *Hello.class*, koja sadrži JVM bajt-kod koji pripada klasi *Hello*. Rezultat prevođenja daje odgovarajući *.class* fajl.
- Prevedeni program pokrećemo komandom:

```
java Hello
```

- Ovog puta nije dozvoljeno navođenje ekstenzije *.class* prilikom pokretanja.

Prevođenje i pokretanje programa

- Primer programa koji se sastoji iz dve klase:

Automobil.java

```
class Automobil {
        boolean radi;
        void upali() {
            radi = true;
        }
        void ugasi() {
           radi = false;
        }
}
```

Test.java

```
class Test {
  public static void main(String args[])
  {
    Automobil a;
    a = new Automobil();
    a.upali();
  }
}
```

- Ove dve klase su smeštene u odgovarajućim datotekama *Automobil.java* i *Test.java*.
- Njihovim prevođenjem dobijaju se dva *.class* fajla, *Automobil.class* i *Test.class*.
- Program se pokreće tako što se navodi ime one klase koja sadrži metodu main, u ovom primeru bilo bi:

```
java Test
```

- Više klasa koje čine program mogu posedovati metodu *main*. Početak izvršavanja programa određuje se prilikom pokretanja programa, tako što se navede ime one klase čiju metodu *main* želimo da pokrenemo.

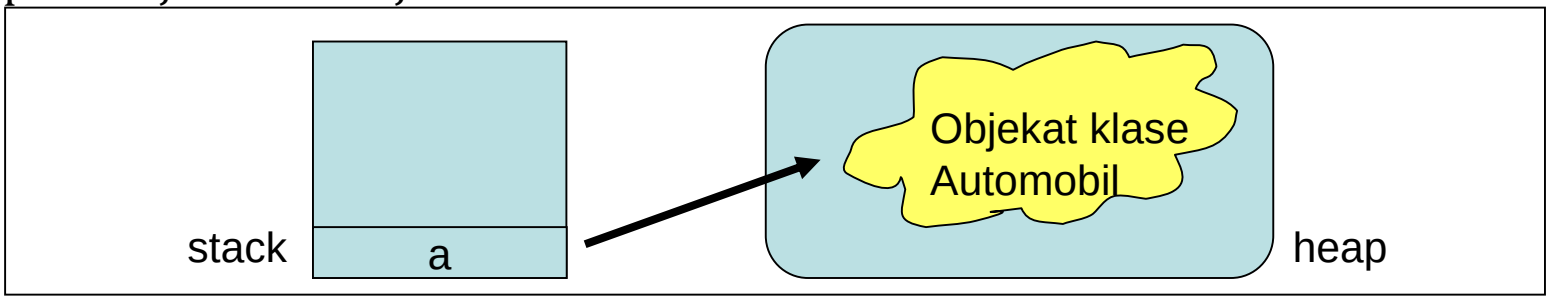
Reference na objekte

- U prethodnom primeru, u metodi *main*, napisano je:

```
Automobil a;
a = new Automobil();
```

deklarisali smo promenljivu *a* tipa *Automobil*, a zatim kreirali objekat klase *Automobil* i vezali ga za tu promenljivu *a*.

Promenljiva *a* predstavlja referencu na objekat klase *Automobil*. Promenljiva *a* je lokalna promenljiva u metodi *main*, tako da se smešta na stek, dok se memorija za objekat klase *Automobil* zauzima na *heap*-u programa. Slika 1.1 prikazuje tu situaciju.



Slika 1.1. Referenca koja ukazuje na objekat

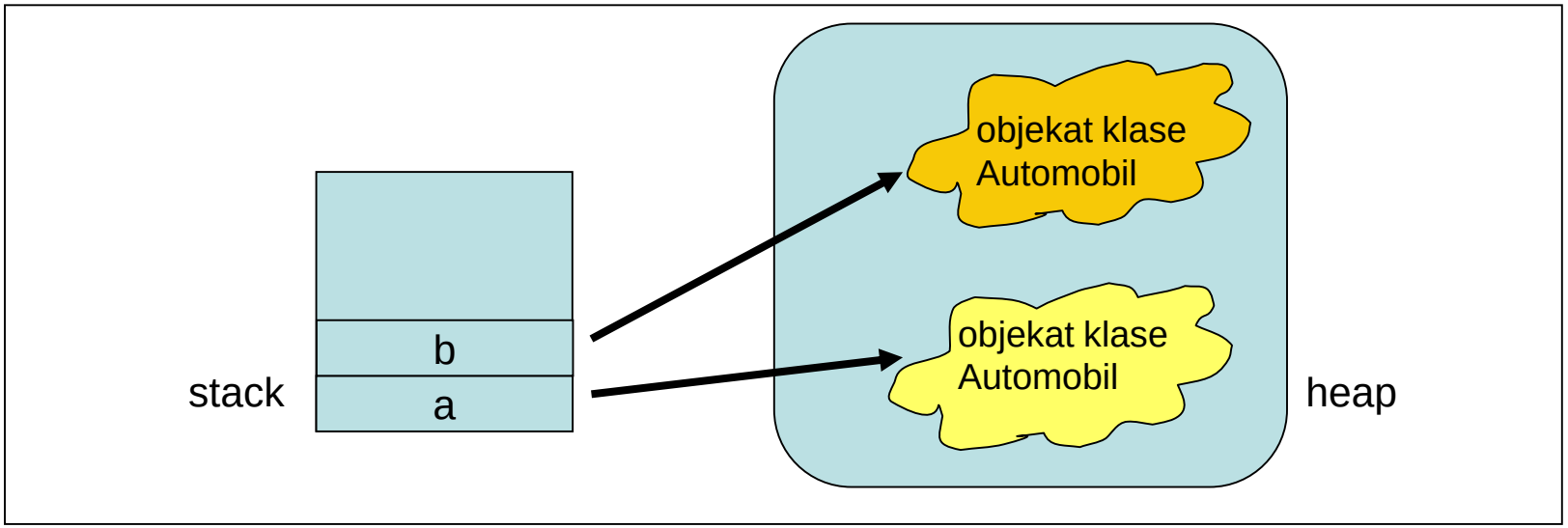
- Kaže se da je *a* referenca na objekat klase *Automobil*. Promenljiva *a* nije pointer u smislu kako ga definiše C++, jer nije dopuštena nikakva aritmetika sa ovakvim promenljivim, niti dodeljivanje proizvoljnih vrednosti. Jedina vrednost koju referenca može da sadrži je adresa pravilno inicijalizovanog objekta na koga ukazuje ili null.

Reference na objekte

- Sledeći primer prikazuje kreiranje dva objekta klase *Automobil* i inicijalizaciju referenci tako da ukazuju na odgovarajuće objekte. Reference se nalaze na steku programa, dok su objekti smešteni na *heap*.

```
Automobil a = new Automobil();
Automobil b = new Automobil();
```

- Situacija koja se nakon ovoga nalazi u memoriji je prikazana na slici 1.2.



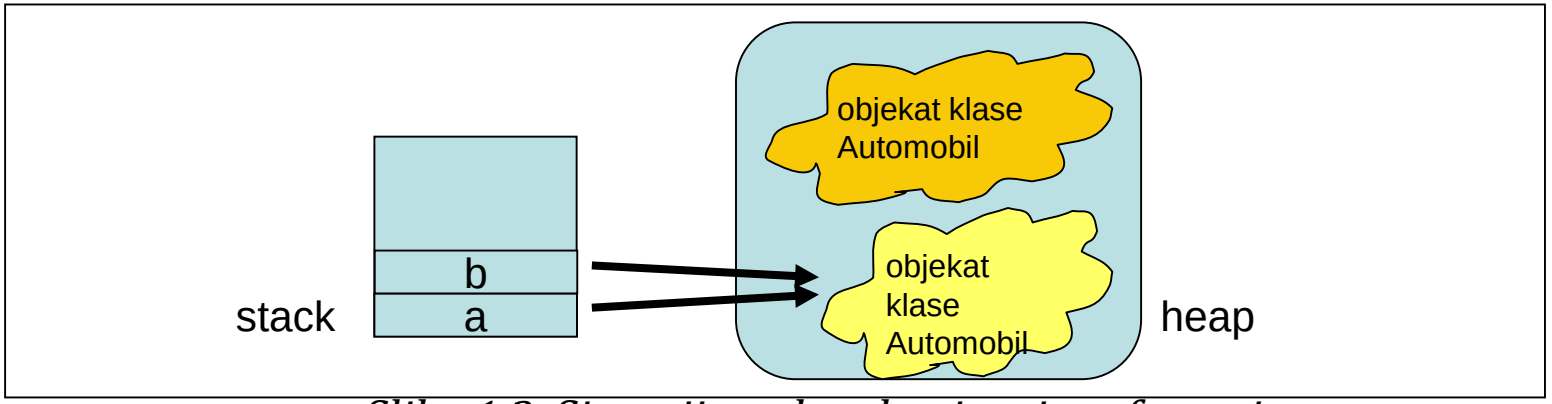
Slika 1.2. Dve reference koje ukazuju na dva objekta

Reference na objekte

- Ako se sada izvrši naredba

```
b = a;
```

desiće se situacija kao što je prikazano na slici 1.3.



Slika 1.3. Situacija nakon kopiranja referenci

- Objekat na koga je ukazivala referenca *b* više nije dostupan ni na jedan način, jer je jedina mogućnost da se nekoj referenci dodeli vrednost dodela vrednosti postojeće reference (tipa b = a) ili dodela vrednosti reference na novokreiran objekat (tipa a = new ...) ili null.
- Kako objekat više nije dostupan, trebalo bi ga ukloniti iz memorije kako bi se izbeglo curenje memorije.
- Java ne poseduje posebnu jezičku konstrukciju kojom se memorija dealocira (poput operatora delete u jeziku C++). Za dealokaciju memorije zadužen je poseban pozadinski proces programa koji se naziva *garbage collector* (skupljač đubreta).

Operatori

- Operatori koji služe za gradnju Java izraza su operatori koji su preuzeti iz jezika C++.
- Grupisani su u nekoliko grupa:
  - aritmetički operatori (+, -, *, /, %, ++, +=, -=, --, itd.);
  - relacioni operatori (==, <, >, =, !=, >=, <=) ;
  - logički operatori (&&, ||, !);
  - bit-operatori (&, |, ~, ^, >>, >>>, <<, &=, |=, ~=, ^=, >>=, >>>=, <<=) ;
  - operator dodele (=);
  - operator uslovne dodele (?:).
- Bitna razlika u odnosu na C++ je postojanje primitivnog tipa boolean te vrednost logičkih izraza mora biti ovog tipa.
- To znači da su vrednosti u if ili while konstrukcijama logičkog tipa, pa nije moguće pisati

```
while (1)       //error
if (a = 1)      //error
```

Kontrola toka programa

- Za kontrolu toka programa na raspolaganju su standardne konstrukcije, preuzete iz jezika C++.
- Dostupne konstrukcije su sledeće:
  - uslovne naredbe

```
if (uslov) {naredba/e}
if (uslov) {naredba/e 1} else {naredba/e 2}
switch (izraz) {
      case vrednost 1: naredba/e 1; break;
deafult: naredba/e 2;
}
```

  - iterativne

```
for(inicijalizacija; uslov ; iteracija) {naredbe}
for(tip promenljiva:kolekcija) {naredbe}
while(uslov){naredbe}
do {naredbe}while(uslov);
```

  - naredbe skoka

```
break;          //izlazak iz prvog okružujućeg bloka
continue;       /* skok na kraj tekuće iteracije */
```

Inicijalizacija objekata

- Prilikom konstruisanja novog objekta, nakon alokacije potrebne memorije za smeštaj objekta, biće pozvana specijalna metoda namenjena za inicijalizaciju objekta nazvana *konstruktor*.
- Konstruktor obavezno ima naziv jednak nazivu klase, i nema nikakav povratni tip, čak ni void.
- Primer konstruktora

```
class A {
   A() {
     System.out.println("konstruktor");
   }
}
```

- U ovom primeru, konstruktor će biti pozvan prilikom kreiranja objekta klase *A*. Na primer:

```
A varA = new A();
```

je deklaracija reference *varA* koja ukazuje na objekat klase *A*, pri čemu se vrši i inicijalizacija ove reference na novokreirani objekat. U trenutku kada se izvrši ovaj red (kreiranje objekta pomoću new A()), na konzoli će se ispisati reč konstruktor.

- Ukoliko se unutar definicije klase ne navede nijedan konstruktor, kompajler će sam generisati tzv. podrazumevani konstruktor koji nema parametre i telo mu je prazno.

Uništavanje objekata

- Pomenut je problem uklanjanja nedostupnih objekata iz memorije. Za taj posao zadužen je poseban pozadinski proces koji se naziva *garbage collector* (GC).
- Ovaj proces radi nezavisno od pokrenutog programa, u smislu da sam odlučuje u kom trenutku će iz memorije osloboditi koji nedostupni objekat. Pored automatske dealokacije memorije, GC je zadužen i za automatsku defragmentaciju memorije.
- Za razliku od jezika C++, Java klase ne mogu imati destruktore. Destruktori su specijalne metode koje se pozivaju neposredno pre uklanjanja objekta iz memorije.
- Svu potrebnu dealokaciju memorije u Javi obavlja GC proces. U trenutku dealokacije podrazumeva se da je Java objekat oslobodio ostale resurse koje je koristio (otvorene datoteke, mrežne konekcije, itd).
- Ukoliko je neophodno obaviti neku operaciju neposredno pre nego što GC uništi objekat, ta operacija se može implementirati u okviru specijalne metode *finalize*.
- Metoda *finalize* se poziva neposredno pre uništavanja objekta od strane GC-a.
  - Ovu metodu ne treba koristiti za oslobađanje zauzetih resursa, jer se metoda *finalize* ne mora pozvati!
  - GC sam određuje kada će ukloniti objekat iz memorije, i lako se može desiti da se to nikad ne dogodi.

Uništavanje objekata

- Primer: program je pokrenut, radio je neko vreme, računar raspolaže sa dovoljno radne memorije tako da GC nije počeo sa uklanjanjem objekata, i tako sve do završetka rada programa; GC nije uklonio objekat, samim tim nije pozvao metodu finalize, i eventualno oslobađanje zauzetih resursa se nije ni desilo.
- Primer klase koja implementira metodu *finalize* (pokretanjem ovog programa sa java A verovatno će se demonstrirati mogućnost da se GC nikad ne aktivira):

```
class A {
   A() {
      System.out.println("Konstruktor");
   }
   protected void finalize() throws Throwable {
      System.out.println("finalized");
   }
   public static void main(String[] args) {
      A a = new A();
       a = null;
      System.out.println("main running...");
   }
}
```

Metode i njihovi parametri

- Sintaksa definisanja metoda (u smislu navođenja njihovog imena, liste parametara i tipa rezultata) liči na sintaksu u jeziku C++.
- Primer metode koja prima tri parametra, sa tipovima String, int i boolean, a vraća vrednost tipa void (tj. ne vraća rezultat).

```
void metoda(String name, int value, boolean test) { ... }
```

- Parametri metode mogu biti primitivni tipovi i reference na objekte;
- Tip rezultata metode može biti primitivni tip ili referenca na objekat (rekosmo, konstruktor nema povratni tip).
- Promena vrednosti parametra u okviru metode može da ima ili nema efekta na promenljivu koja je korišćena kao parametar nakon povratka iz metode.

  Posmatrajmo sledeću metodu:

```
static void test(Automobil a) {
        a.upali(); // a.radi = true;
}
```

- U pitanju je metoda koja u okviru svog tela vrši modifikaciju svog parametra *a* preko metode *upali*.

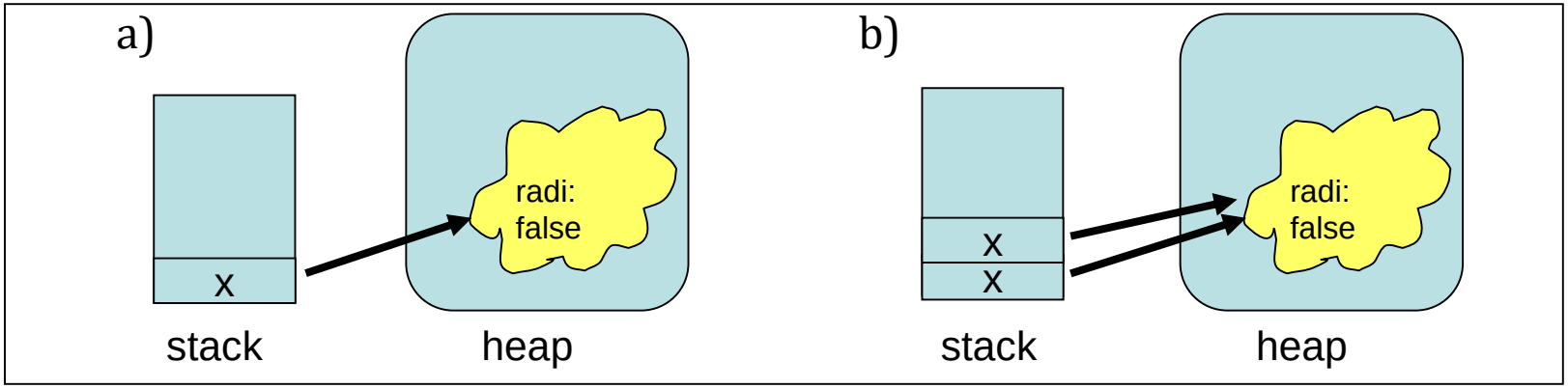
Metode i njihovi parametri

- U slučaju da se ova metoda pozove u sledećem segmentu koda:

```
Automobil x = new Automobil();
x.radi = false;
test(x);        // vrednost atributa radi objekta x biće true
   // uklonite static sa prethodnog slajda i podesite da radi:
    // Hm hm = new Hm(); hm.test(x) umesto test(x);
```

- Slika 1.4 ilustruje da je na steku kreirana referenca *x* na objekat klase *Automobil*. Zatim je vrednost atributa *radi* ovog objekta postavljena na false (slika a).

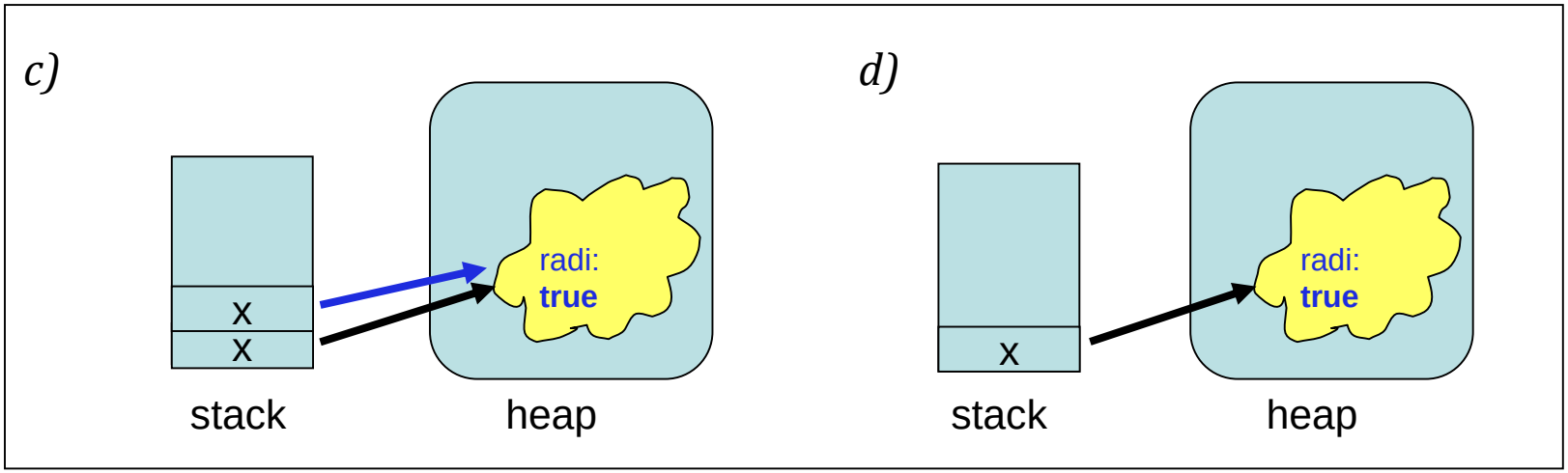
  Nakon toga pozvana je metoda *test*, sa referencom *x* kao parametrom. Parametri metoda se smeštaju na stek prilikom poziva metode. Tako je i referenca *x* iskopirana na stek još jednom (slika b).



Slika 1.4.ab. Promena stanja objekta koji je parametar metode

Metode i njihovi parametri

- U okviru tela metode *test* ova kopija reference *x* se koristi kao parametar metode i preko nje se pristupa istom onom objektu na koji ukazuje i originalna referenca *x*.
- Pristup objektu se u ovom slučaju svodi na promenu vrednosti atributa *radi* na true (slika c).
  - Kod vraćanja iz metode nazad, sa steka se uklanjaju parametri korišćeni prilikom poziva metode. Tako se sa steka uklanja kopija reference *x* i ostaje samo originalna referenca. Kada preko nje pristupimo atributu *radi*, videće se da je on promenio vrednost (slika d).



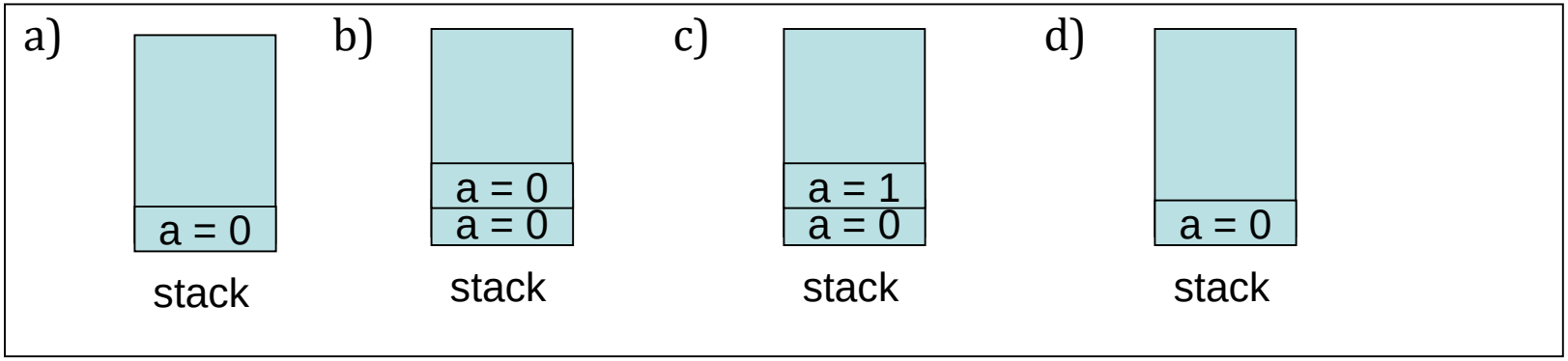
Slika 1.4.cd. Promena stanja objekta koji je parametar metode

Metode i njihovi parametri

- Promene nad parametrima metode načinjene u okviru tela metode koji su reference na objekte su vidljive nakon povratka iz metode. Efekat je isti kao kod *prenosa parametara po adresi*.
- Primitivni tipovi se kao parametri metoda ponašaju kao kod *prenosa parametara po vrednosti*.

Sledi primer:

```
void test(int a) {
 a = 1;
}
...
int a = 0;
test(a);        //nakon ove naredbe a = 0
```



Slika 1.5. Promena vrednosti parametra metode koji je primitivnog tipa

Metode i njihovi parametri

- Slika 1.5 prikazuje šta se dešava u ovom slučaju:
  - deklariše se promenljiva *a* tipa int i odmah se inicijalizuje na vrednost 0.
  - lokalne promenljive primitivnog tipa se smeštaju na stek (trenutno stanje ilustruje slika a)
  - poziva se metoda *test* sa parametrom *a*;

    parametar *a* se smešta na stek (njegova vrednost se kopira još jednom slika b).
  - u okviru metode *test* vrednost parametra se menja u 1, pri čemu se menja druga kopija na steku (slika c).
  - nakon povratka iz metode, parametar se uklanja sa steka i na steku ostaje originalna vrednost promenljive *a* koja nije menjana (slika d).
- Preklopljene metode (*method overloading*) su metode koje imaju isto ime, ali se razlikuju po listi parametara.
- Kompajler ih smatra za sasvim različite metode, bez obzira što imaju isto ime.
- Metode ne mogu da se razlikuju samo po tipu rezultata kojeg vraćaju.

  Primer klase sa tri preklopljene metode:

```
class A { int metoda()          { ... }
          int metoda(int i)     { ... }
          int metoda(String s)  { ... }      }
```

- Preklapanje metoda se odnosi i na konstruktore.

Ključna reč final

- Ključna reč final se može naći ispred definicije atributa ili metode unutar definicije klase.
- Ako se nađe ispred atributa, označava atribut kome nije moguće promeniti vrednost.

  final atribut predstavlja konstantu.
- Inicijalizacija prilikom deklaracije atributa je obavezna.

  Primer definicije final atributa :

```
final int size = 100;
```

- Ključna reč final ima drugo značenje kod metoda:

  označava metode koje se ne mogu redefinisati prilikom nasleđivanja date klase.

  Primer *final* metode glasi:

```
final int metoda(int i) { ... }
```

Ključna reč static

- Ključna reč static se može naći ispred definicije atributa ili metode, nezavisno od pojave ključne reči final.
- Kada se nađe ispred definicije atributa, označava atribut koji pripada klasi, a ne objektima (kao instancama klase).
- Svi objekti date klase dele istu vrednost statičkog atributa.

Primer: klasa koja poseduje jedan statički atribut:

```
class StaticTest {
    static int i = 0;
    static void metoda() { i++; }
}
```

Tada će sledeći programski segment izazvati promenu vrednosti atributa *i* u oba objekta:

```
StaticTest a = new StaticTest();
StaticTest b = new StaticTest();
a.i++;                                  //  a.i == b.i == 1
```

Ključna reč static

- Statički atribut je pridružen klasi, a ne njenim instancama.
- Statičkom atributu se može pristupiti i kada nije kreirana nijedna instanca klase.

  Tada se atributu pristupa tako što se navodi ime klase, pa zatim ime atributa. Primer:

```
StaticTest.i++;
```

- Statičke metode su metode koje mogu biti pozvane nad klasom.

```
a.metoda();              //nepravilno
StaticTest.metoda();     //pravilno
```

- Statičke metode imaju pristup samo statičkim atributima klase.
- Često korišćeni primer upotrebe statičkog atributa je ispisivanje na konzolu pri čemu se poziva metoda *println* objekta *out* koji je statički atribut klase *System* (*out* zapravo predstavlja standardni izlaz). :

```
System.out.println("Hello, world!");
```

Nizovi

- Nizovi se u Javi definišu slično kao u jeziku C++. Prvi element niza ima indeks nula.

  Primer:

```
int[] a; //niz čiji su elementi tipa int
int[] b; // niz čiji su elementi tipa int
```

  Ovim je samo definisana referenca na niz; niz se nakon toga mora kreirati na način sličan kreiranju objekata.

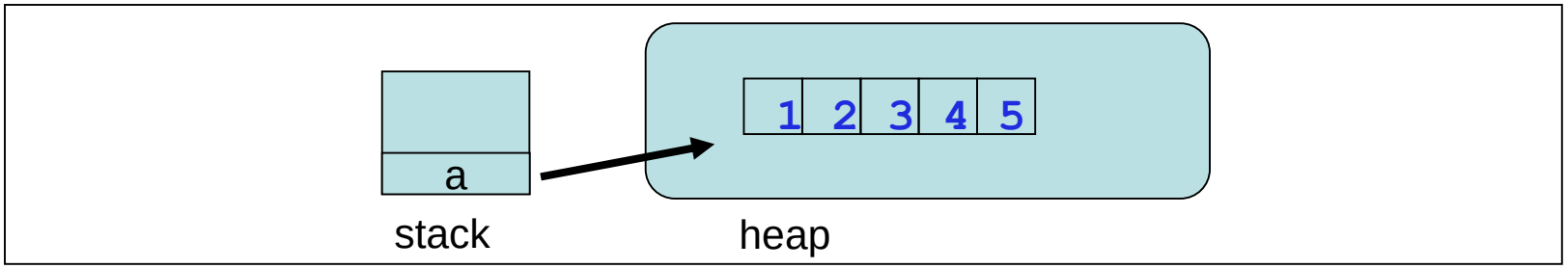
  Primer:

```
a = new int[5]; //alociranje niza od pet int elemenata
```

  Primer: niz se definiše, alocira memorija za njega i odmah inicijalizuje

```
int[] a = { 1, 2, 3, 4, 5 };
```

- Prilikom definicije niza, referenca na niz se čuva na steku, dok se elementi niza čuvaju na *heap*-u, slično kao i objekti. Slika 1.6 ilustruje ovu situaciju



Slika 1.6. Inicijalizovan niz

Nizovi

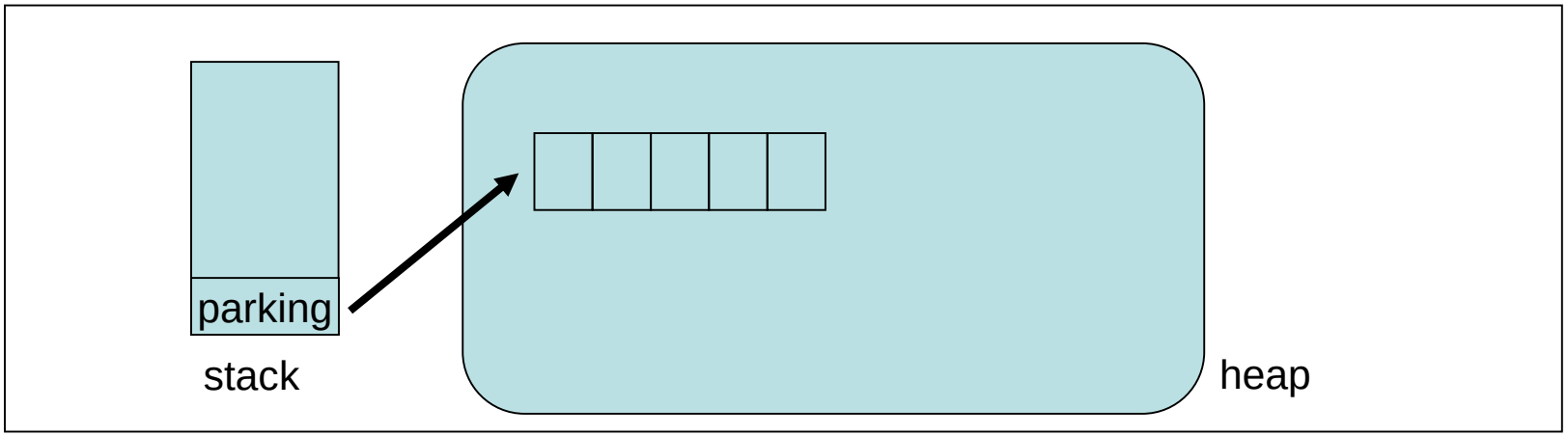
- Kada je u pitanju niz čiji su elementi primitivnog tipa, alokacija memorije za elemente niza se odvija automatski.
- To nije slučaj kada je u pitanju niz čiji su elementi objekti neke klase.

Primer:

niz od 5 elemenata klase *Automobil* se definiše :

```
Automobil[] parking = new Automobil[5];
```

Ovim je zapravo definisan niz čiji elementi su reference na objekte klase *Automobil*. Slika 1.7 ilustruje ovu situaciju.



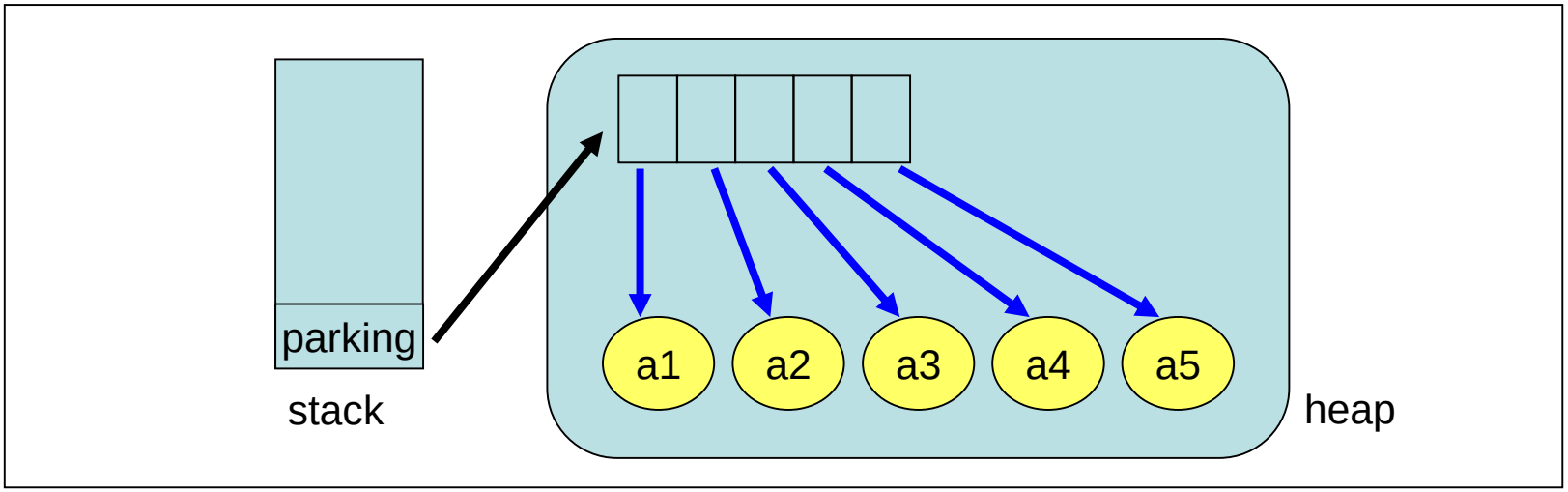
Slika 1.7. Niz objekata pri čemu objekti nisu inicijalizovani

Nizovi

- Ovakav niz referenci na objekte se može inicijalizovati, na primer, u odgovarajućoj for petlji, kao u primeru:

```
for (int i = 0; i < parking.length; i++)
    parking[i] = new Automobil();
```

Svaki niz ima definisan atribut *length* koji predstavlja dužinu alociranog niza. Sada će stanje u memoriji izgledati kao na slici 1.8.



Slika 1.8. Niz inicijalizovanih objekata

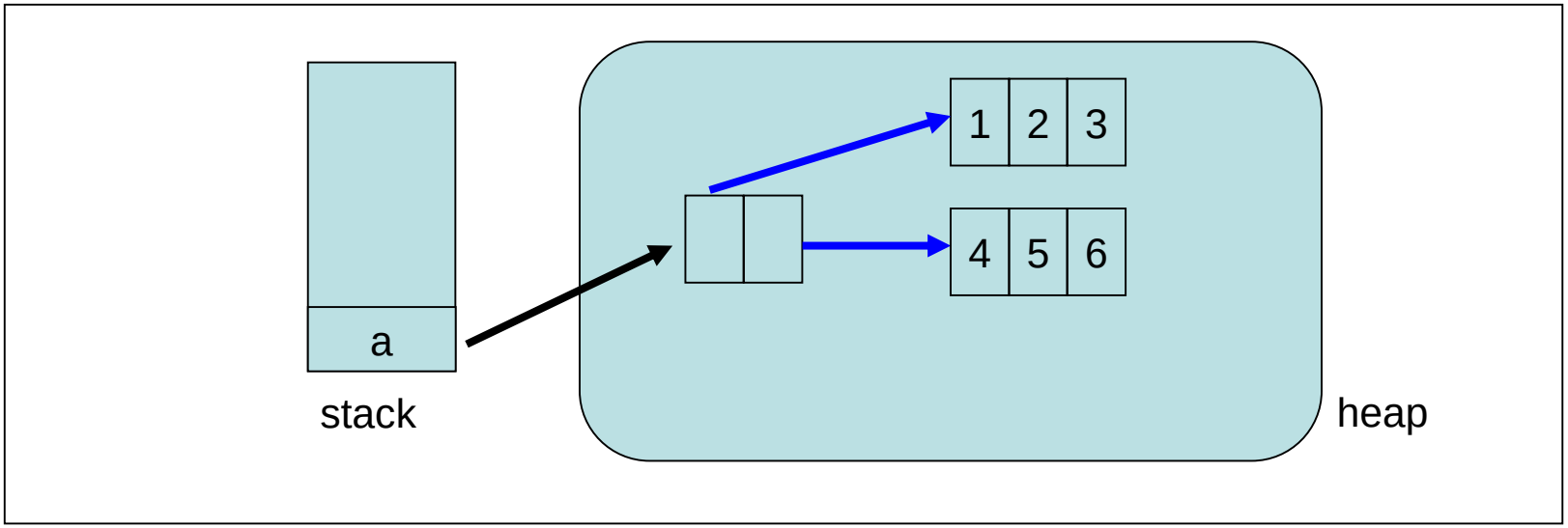
Višedimenzionalni nizovi

- Višedimenzionalni nizovi se predstavljaju kao nizovi nizova.
- Sintaksa je slična jeziku C++.

  Primer: sledeća definicija će kreirati niz od dva elementa koji su reference na nizove od tri elementa tipa int.

```
int[][] a = { {1, 2, 3}, {4, 5, 6} };
```

Stanje u memoriji nakon kreiranja ovakvog niza izgleda kao na slici 1.9.



Slika 1.9. Dvodimenzionalni niz

Višedimenzionalni nizovi

- Viaedimenzionalni niz se može kreirati na sledeći način:

```
int[][] a = new int[2][3];
```

Ovim je izvršena potrebna alokacija memorije, ali ne i inicijalizacija vrednosti elemenata niza.

- Dvodimenzionalni niz se može kreirati i postupno.

Primer:

```
int[][] a = new int[2][];
for (int i = 0; i < a.length; i++)
   a[i] = new int[3];
```

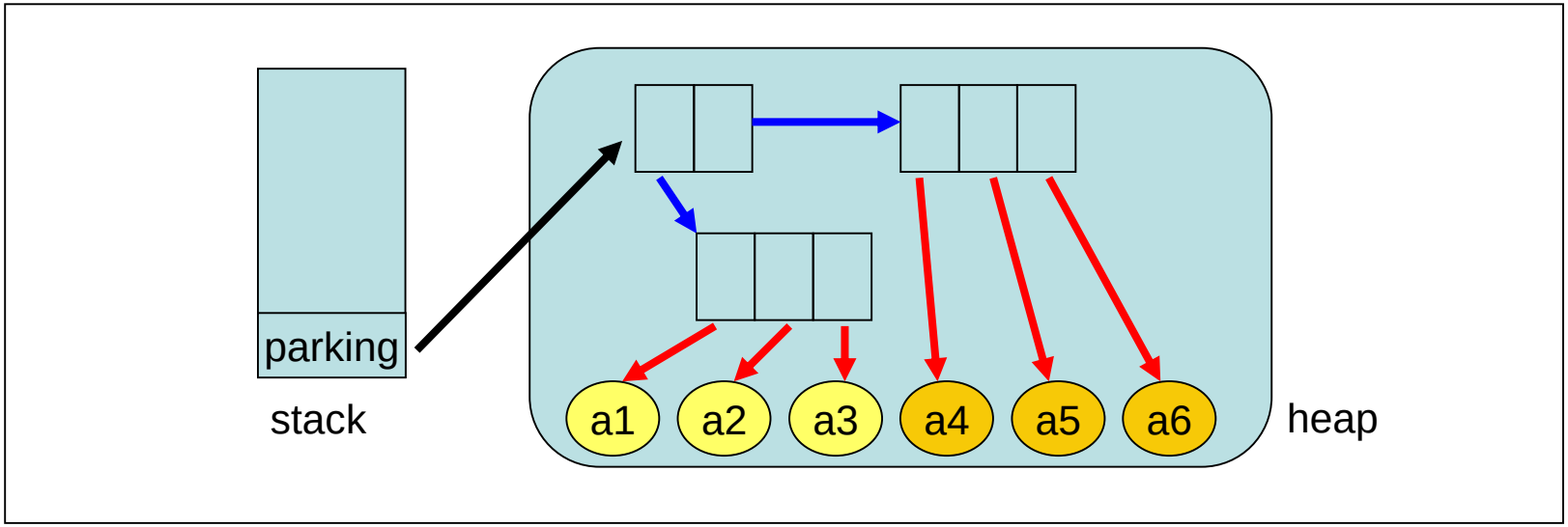
- Prilikom kreiranja višedimenzionalnog niza čiji su elementi objekti, a ne primitivni tip, potrebno je još izvršiti i dodatno kreiranje svakog od objekata.

Primer:

```
Automobil[][] parking = new Automobil[2][];
for (int i = 0; i < parking.length; i++) {
   parking[i] = new Automobil[3];
   for (int j = 0; j < parking[i].length; j++)
      parking[i][j] = new Automobil();
}
```

Stanje u memoriji nakon kreiranja ovakvog niza izgledaće kao na slici 1.10.

Višedimenzionalni nizovi



Slika 1.10 Višedimenzionalni niz čiji su elementi objekti

- Višedimenzionalni niz objekata može se inicijalizovati odmah prilikom definicije, slično kao kod višedimenzionalnog niza primitivnih tipova.

  Primer:

```
Automobil[][] a = {
     { new Automobil(), new Automobil() },
     { new Automobil(), new Automobil() }
};
```