

# Referencijalni tipovi

# Referencijalni tipovi

- Pored prostih tipova, u Javi postoje **referencijalni tipovi**
- Referencijalni tipovi u Javi su:
  - Klase
  - Interfejsi
  - Nizovi
  - Nabrojivi tipovi (uvedeni u Javi 5)
  - Lambda izrazi (uvedeni u Javi 8)
- Sem nizova, svi ovi tipovi su **uvedeni** (ili **korisnički**) tipovi, sto znači da korisnik sam može da definiše konkretne tipove, ili koristi već definisane tipove
- Na ovom kursu pokrićemo klase (delimično), nizove i nabrojive tipove
  - Još o klasama, interfejsima i nabrojivim tipovima čućete na kursevima OOP1 i OOP2
  - Lambda izrazi rade se na predmetu Programski jezici (u drugom programskom jeziku - Scheme)

# Šta znači “referencijalni”?

- Kod prostih tipova, sa vrednošću promenljive u memoriji se radi **direktno**: preko imena promenljive pristupa se vrednosti, vrednost se može menjati naredbom dodele, itd.
- Kod referencijalnih tipova, vrednosti se u memoriji skladište **indirektno**: preko imena promenljive pristupa se **referenci** (memorijskoj adresi) preko koje se pristupa vrednosti smeštenoj u memoriji na nekom drugom mestu
  - Mesto (deo memorije) gde se smeštaju vrednosti kojima se pristupa preko referenci naziva se hrpa (engl. **heap**)


# Šta znači “referencijalni”?

- Razliku između prostih i referencijalnih tipova ilustrovaćemo preko tipa `String`, koji je u stvari klasa
- Posmatrajmo dve promenljive tipa `int` i `String`:  

```
int num = 10; // prost tip
```

```
String name = "Hello"; // referencijalni tip
```
- Na sledećoj slici prikazano je kako izgleda meorija računara, sa datim adresama memorijskih lokacija, njima pridruženim imenima promenljivih, i samim podacima u memoriji

<b>Memory Address</b>	<b>Variable Name</b>	<b>Data</b>
1001	num	10
:		:
1563	name	Address(2000)
:		:
:		:
2000		"Hello"

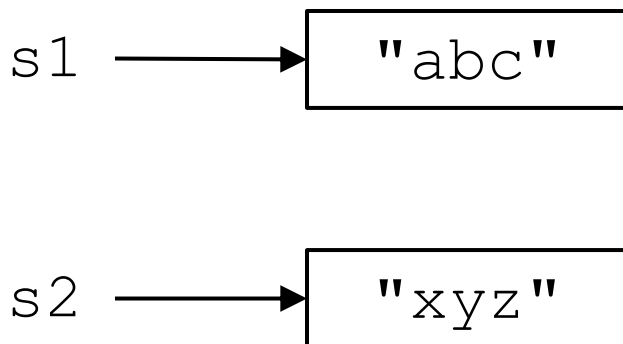


# Šta znači “referencijalni”?

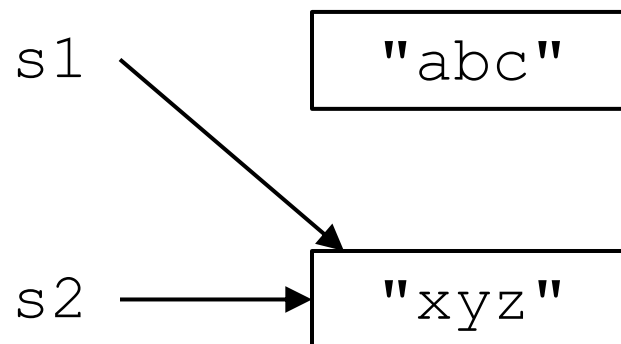
- Jednostavno rečeno, kod prostih tipova ime promenljive je zamena za **adresu**, a kod referencijalnih tipova ime promenljive je zamena za **adresu adrese**
- Promenljive prostog tipa predstavljaju jedan vid **apstrakcije**: promenljive svojim imenom apstrahuju memorijske adrese (skrivaju detalje o vrednostima adresa, organizaciji memorije, itd.)
- Promenljive referencijalnih tipova ovu apstrakciju podižu na viši (meta) nivo
  - Omogućava pravljenje dugačkih “lanaca” referenci, čime se mogu praviti složene strukture podataka (punu snagu ovog pristupa osetićete na kursevima Strukture podataka i algoritmi 1 i 2)

# Šta znači “referencijalni”?

```
String s1 = "abc";  
String s2 = "xyz";
```



```
String s1 = "abc";  
String s2 = "xyz";  
s1 = s2;
```



# Klase (1. deo)

- Klase su osnovni referencijalni tip u Javi
- Svi ostali referencijalni tipovi su “u pozadini” realizovani kao klase, mada se to na prvi pogled ne vidi
- Vrednosti promenljivih tipa neke klase su reference na **objekte** ili **instance**, i moraju se kreirati korišćenjem operatora `new`
  - Izuzetak je klasa `String`, čije se instance mogu kreirati i navođenjem literala:  
`String name = "Hello";`
- Mnogi programski jezici dizajnirani su po principu da je neki koncept osnovni, tzv. “građanin 1. reda”, oko kog se “sve vrti”
  - U slučaju Jave, taj koncept je **klasa**

# Klase (1. deo)

- Klase se deklariraju pomoću ključne reči `class`, nakon čega sledi ime klase, i u vitičastim zagradama navedeni članovi klase
- Od članova klase, na početku ćemo obraditi:
  - Polja
  - Statičke metode
- Do sada nam je svaki program u stvari bio klasa sa jednim statičkim metodom `main`, koji ima poseban status jer se on izvršava pri pokretanju programa u JVM
- Mi u klasi možemo deklarirati i druge statičke metode i pozivati ih iz metoda `main`
- Pri tom ćemo podatke (prostih i referencijalnih tipova podataka) prosledivati statičkim metodama kao argumente, čime ćemo dobiti programe pisane **proceduralnim stilom programiranja**
- Kasnije ćemo obraditi nestatičke metode i druge vrste članova klase, i objasniti osnove **objektno-orijentisanog stila programiranja**



# Polja klase

- Polja klase predstavljaju podatke, i deklarišu se na isti način kao lokalne promenljive, samo na drugom mestu:
  - Lokalne promenljive se deklarišu u okviru metoda i blokova
  - Polja klase se deklarišu u okviru klase
- Polja klase “vezuju” se za objekte, odnosno instance klase, i postoje nezavisno u okviru svakog objekta (tako da u stvari postaju *polja objekta*)
- Poljima objekta pristupa se pomoću operatora .
- Proširićemo definiciju pojma **promenljiva**, koja će sad da uključuje:
  - Lokalne promenljive
  - Polja
- Klase predstavljaju jedan od načina da se, pomoću polja, podaci grupišu i “spakuju” u logične celine (objekte)

# Polja klase: primer

```
class Tacka {  
    double x, y;  
}
```

```
class TackaTest {  
    public static void main(String[] args) {  
        Tacka t1 = new Tacka();  
        Tacka t2 = new Tacka();  
        t1.x = 1.0;  
        t1.y = 2.0;  
        t2.x = 5.0;  
        System.out.println("t1 = (" + t1.x + ", " + t1.y + ")");  
        System.out.println("t2 = (" + t2.x + ", " + t2.y + ")");  
    }  
}
```

## ■ Izlaz:

t1 = (1.0, 2.0)

t2 = (5.0, 0.0)

# Polja klase: anti-primer

- U ovom “anti-primeru” ne grupišemo promenljive u klase/objekte kao polja, već uvodimo posebne nezavisne (lokalne) promenljive za svaki podatak. Ovakav pristup postaje nezgodan čim se program malo poveća

```
class TackaBad {  
    public static void main(String[] args) {  
        double t1x = 1.0;  
        double t1y = 2.0;  
        double t2x = 5.0;  
        double t2y = 0.0;  
        System.out.println("t1 = (" + t1x + ", " + t1y + ")");  
        System.out.println("t2 = (" + t2x + ", " + t2y + ")");  
    }  
}
```

- Izlaz:

t1 = (1.0, 2.0)

t2 = (5.0, 0.0)

# Kreiranje objekata operatorom new

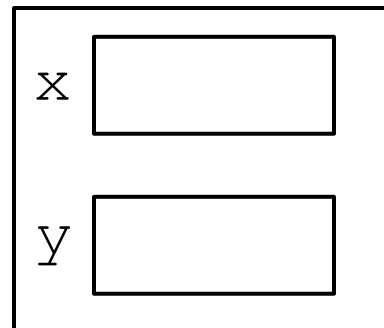
- U prethodnom primeru smo videli kako se kreira nova instanca klase:  
`Tacka t1 = new Tacka();`
- Pri izračunavanju izraza `new Tacka()` dešava se sledeće:
  - Rezerviše se (alocira) memorijski prostor na heap-u potreban da se smesti objekat sa svojim članovima
  - Polja objekta se inicijalizuju na podrazumevane vrednosti:
    - `0`, `0L`, `0.0`, `'\0'`, `false`, **`null`**...
    - Ili na vrednosti eksplicitno navedene kod polja korišćenjem operatora `=`
  - Izvršava se konstruktor (kod u koji se mogu staviti neke posebne inicijalizacije i slično, više o njima kasnije)
  - Referenca na napravljeni objekat se vraća kao vrednost izraza

# Kreiranje objekata operatorom new

```
Tacka t1 = new Tacka();
```

- Rezervise se memorijski prostor za objekat

t1

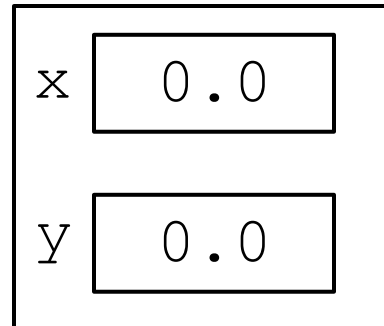


# Kreiranje objekata operatorom new

```
Tacka t1 = new Tacka();
```

- Rezervise se memorijski prostor za objekat
- Polja objekta se inicijalizuju

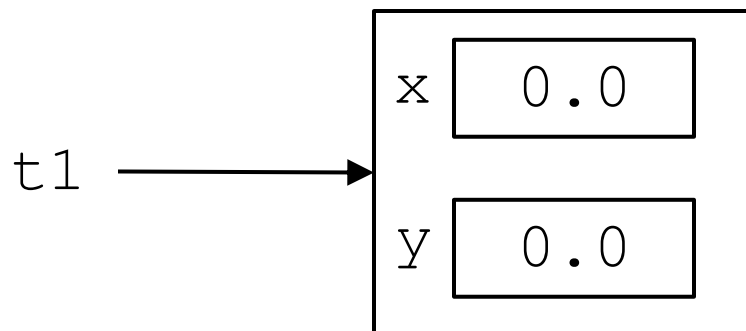
t1



# Kreiranje objekata operatorom new

Tacka t1 = new Tacka();

- Rezervise se memorijski prostor za objekat
- Polja objekta se inicijalizuju
- Referenca se vraća kao vrednost izraza `new`, i dodeljuje t1



# Literal `null`

- Promenljive referencijalnog tipa mogu se inicijalizovati i literalom `null`, koji je podrazumevana vrednost za referencijalne tipove (kao 0 za `int`, itd.): Tacka `t1 = null;`
- Literal `null` označava “praznu” referencu, odnosno referencu koja ne referencira “nigde”
- Drugim rečima, dodeljivanjem vrednosti `null` promenljivoj referencijalnog tipa kažemo da ne postoji objekat u memoriji na koji ta promenljiva referencira
- **Opasnost:** ako objekat nije inicijalizovan, odnosno ako je `null`, pokušaj pristupa nekom članu izazvaće grešku u toku izvršavanja programa



# Brisanje objekata

- U Javi ne postoji poseban operator kojim se objekti brišu iz memorije, tj. ne postoji pandan operatoru `new` koji radi suprotnu operaciju
- U Javi se brisanje objekata radi automatski, po potrebi, o čemu u principu programer ne bi trebao/la da brine
- Da bi objekat mogao u jednom trenutku da bude obrisani, neophodno je da na njega ne pokazuje ni jedna referenca
  - Ovo se postiže dodelom promenljivoj reference na neki drugi objekat, ili `null`
  - Ne postoji način da se u programu “povrati” pristup ovakvim objektima
- Za brisanje ovakvih objekata zadužen je poseban proces, tzv. sakupljač đubreta (engl. *garbage collector*)
- Ovaj proces pokreće se automatski, a moguće je eksplicitno zatražiti njegovo pokretanje pozivom metoda `System.gc()` ;

# Polja klase: lanci polja

- Za početak ćemo posmatrati klase i objekte samo kao strukture podataka koje sadrže polja
  - Analogno tipovima podataka `struct`, `record` i sl. u nekim drugim programskim jezicima
- Kao i sve promenljive, polja klase mogu biti bilo kog tipa podataka, pa i referencijalnog, odnosno mogu biti tipa neke klase
- Time se može postići “ulančavanje” polja objekata, jer polje objekta može biti objekat čije neko polje takođe može biti objekat, itd.
- Dakle, možemo imati lanac pristupa poljima preko operatora `.`

# Polja klase: lanci polja - primer

```
class Automobil {  
    String marka, proizvođjac;  
    int godinaProiz;  
    String boja;  
    int brKonja, brVrata = 5;  
    String regBroj;  
}
```

```
class Vlasnik {  
    String ime, prezime, JMBG;  
    Automobil auto;  
}
```

```
class Automobili {  
    public static void main(String[] args) {  
        Vlasnik pera = new Vlasnik();  
        pera.ime = "Pera";  
        pera.prezime = "Peric";  
        pera.JMBG = "0101900800001";  
        pera.auto = new Automobil();  
        pera.auto.marka = "Yugo Koral 55";  
        pera.auto.proizvođjac = "Crvena zastava";  
        pera.auto.godinaProiz = 1989;  
        pera.auto.boja = "crvena";  
        pera.auto.brKonja = 55;  
        pera.auto.brVrata = 3;  
    }  
}
```

# Nasleđivanje klasa

- Jedan od fundamentalnih koncepata objektno-orijentisanog programiranja je **nasleđivanje** klasa
- Kada se deklariše da klasa B nasleđuje klasu A, to znači da B preuzima članove od A, eventualno dodajući nove
- Na ovom kursu se konceptom nasleđivanja nećemo detaljno baviti, ali moramo biti svesni sledećih posledica:
  - Promenljivoj tipa A (“nadklase”) mogu se dodeljivati objekti tipa B (“podklase”)
  - Sve klase (i ostali referencijalni tipovi) implicitno nasleđuju specijalnu klasu `Object`
- Nasleđivanje će detaljno biti obrađeno na kursu OOP1

# Statički i nestatički članovi

- Statički članovi klase - metodi i polja koja su deklarirana pomoću ključne reči `static`
- Mogu se koristiti i bez prethodnog kreiranja instanci klase, oni se zapravo i ne odnose na instance klase, već na samu klasu
- Svaki objekat klase sadrži svoju kopiju svih polja i metoda klase, osim onih polja i metoda koji su statički, jer statička polja i metodi pripadaju klasi a ne njenim instancama

# Nizovi

- Nizovi predstavljaju grupu elemenata istog tipa koje se pojavljuju pod istim imenom
- Niz je specijalna vrsta objekta i sastoji se od elemenata kojima se pristupa pomoću njihovih indeksa
- Ubuduće, kada kažemo “niz” mislićemo na objekat nizovnog tipa podataka ili sam nizovni tip (biće jasno iz konteksta)
- Indeks prvog elementa u nizu je uvek 0, dok je indeks poslednjeg elementa za 1 manji od ukupnog broja elemenata u nizu
- Tip elemenata niza može biti bilo koji tip, uključujući i tip niza
- Ako su elementi niza nizovi, tada takav niz zovemo više-dimenzionalan niz, inače se radi o jednodimenzionalnom nizu

# Nizovi: deklaracija

- Promenljive tipa jednodimenzionalnog niza deklariramo navođenjem tipa, imena promenljive i jednog para uglastih zagrada i može se inicijalizovati odmah prilikom deklaracije
  - Par uglastih zagrada može se navesti posle imena tipa ili posle imena promenljive
- Prilikom deklaracije niza ne navodi se veličina niza - ona se zadaje tek prilikom njegovog kreiranja `new` operatorom
- Nakon kreiranja, inicijalne vrednosti elemenata niza su nule (brojevni nizovi) , `false` vrednosti (logički nizovi) ili `null` ako su elementi niza referencijalnog tipa

# Nizovi: deklaracija - primeri

## Primeri: Deklaracije nizova i nezavisno kreiranje instanci

```
int[] nizCelih1;  
int nizCelih2[];  
String[] imena1;  
String imena2[];  
Lopta[] lopte;  
Object objekti[];  
...  
nizCelih1 = new int[10];  
imena1 = new String[] {"aca", "ceca", "daca"};  
objekti = new Object[8];
```



# Nizovi: deklaracija - primeri

## Primeri: Deklaracije nizova i istovremeno kreiranje instanci

```
boolean[] logickiNiz = new boolean[18];  
int[] celi1 = new int[] {1, 2, 3};  
int[] celi2 = {1, 2, 3};  
Tacka[] mojeTacke = new Tacka[10];  
Object[] babeIZabe = new Object[] {new Tacka(),  
                                     new Automobil(),  
                                     new Vlasnik()};
```

# Nizovi: deklaracija

- U primerima se mogu uočiti dva vida inicijalizacije nizova, koji se međusobno isključuju:
  - Navođenjem broja elemenata
  - Navođenjem samih elemenata
- Kod načina sa navođenjem elemenata dozvoljeno je izostaviti deo `new TipElementa[]`, kad se inicijalizacija radi pri deklaraciji, u protivnom se taj deo mora navesti
  - Dakle, kod sledećeg koda kompajler prijavljuje grešku:

```
int[]  cel13;  
cel13 = {1, 2, 3};
```

# Nizovi: pristup elementima

- Elementima niza se pristupa tako što se prvo navede ime niza ili izraz čija je vrednost niz (pri čemu to ne sme biti izraz kreiranja niza), nakon čega se u uglastim zagradama navodi celobrojni izraz čija vrednost je indeks elementa kojem pristupamo
- Svaki niz ima i polje `length` koje sadrži broj elemenata niza zadat pri inicijalizaciji
- **Primeri:**

```
celi1[0]           // 1
imena1[1]          // "ceca"
logickiNiz[17]     // false
celi1.length       // 3
```

# Nizovi: pristup elementima - primer 1

```
class FibonaciNiz {  
    public static void main(String[] args) {  
        int[] fib = new int[4];  
        fib[0] = 0;  
        fib[1] = 1;  
        fib[2] = fib[0] + fib[1];  
        fib[3] = fib[1] + fib[2];  
        System.out.println("3. Fibonacijev broj je " + fib[3]);  
    }  
}
```

- Izlaz:

3. Fibonacijev broj je 2

# Nizovi: pristup elementima - primer 2

```
class Automobil {  
    String marka, proizvođač;  
    int godinaProiz;  
    String boja;  
    int brKonja, brVrata = 5;  
    String regBroj;  
}
```

```
class Vlasnik {  
    String ime, prezime, JMBG;  
    Automobil auto;  
}
```

```
class AutomobiliNiz {  
    public static void main(String[] args) {  
        Vlasnik[] vlasnici = new Vlasnik[10];  
        vlasnici[0] = new Vlasnik();  
        vlasnici[0].ime = "Pera";  
        vlasnici[0].prezime = "Peric";  
        vlasnici[0].JMBG = "01019008000001";  
        vlasnici[0].auto = new Automobil();  
        vlasnici[0].auto.marka = "Yugo Koral 55";  
        vlasnici[0].auto.proizvođač = "Crvena zastava";  
        vlasnici[0].auto.godinaProiz = 1989;  
        vlasnici[0].auto.boja = "crvena";  
        vlasnici[0].auto.brKonja = 55;  
        vlasnici[0].auto.brVrata = 3;  
    }  
}
```

# Nizovi: pristup elementima - primer 3

```
class MinNiz {  
    public static void main(String[] args) {  
        int brojeva;  
        do {  
            System.out.print("Unesite broj ulaznih brojeva > 0: ");  
            brojeva = Svetovid.in.readInt();  
        } while (brojeva <= 0);  
        int[] nizBr = new int[brojeva];  
        System.out.println("Unesite brojeve:");  
        for (int i = 0; i < nizBr.length; i++) {  
            System.out.print("Unesite " + i + ". broj: ");  
            nizBr[i] = Svetovid.in.readInt();  
        }  
        int min = nizBr[0];  
        for (int i = 1; i < nizBr.length; i++) {  
            if (nizBr[i] < min) {  
                min = nizBr[i];  
            }  
        }  
        System.out.println("Minimalna vrednost u nizu je: " + min);  
    }  
}
```

# Nizovi: pristup elementima - primer 3

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>javac MinNiz.java
d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java MinNiz
Unesite broj ulaznih brojeva > 0: 6
Unesite brojeve:
Unesite 0. broj: 7
Unesite 1. broj: 3
Unesite 2. broj: 5
Unesite 3. broj: 2
Unesite 4. broj: 99
Unesite 5. broj: 10
Minimalna vrednost u nizu je: 2
```

# Nizovi: pristup elementima - primer 4

```
class SadrzanUNizu {  
    public static void main(String[] args) {  
        final int DUZINA = 10;  
        int[] nizBr = new int[DUZINA];  
        System.out.println("Unesite brojeve:");  
        for (int i = 0; i < nizBr.length; i++) {  
            System.out.print("Unesite " + i + ". broj: ");  
            nizBr[i] = Svetovid.in.readInt();  
        }  
        System.out.print("Unesite trazeni broj: ");  
        int broj = Svetovid.in.readInt();  
        int i = 0;  
        while (i < nizBr.length && nizBr[i] != broj) {  
            i++;  
        }  
        if (i < nizBr.length)  
            System.out.println("Broj je sadrzan u nizu");  
        else  
            System.out.println("Broj nije sadrzan u nizu");  
    }  
}
```



# Nizovi: pristup elementima - primer 4

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java SadrzanUNizu
Unesite brojeve:
Unesite 0. broj: 4
Unesite 1. broj: 2
Unesite 2. broj: 5
Unesite 3. broj: 6
Unesite 4. broj: 8
Unesite 5. broj: 3
Unesite 6. broj: 4
Unesite 7. broj: 1
Unesite 8. broj: 12
Unesite 9. broj: 1
Unesite trazeni broj: 8
Broj je sadržan u nizu

d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java SadrzanUNizu
Unesite brojeve:
Unesite 0. broj: 1
Unesite 1. broj: 2
Unesite 2. broj: 3
Unesite 3. broj: 4
Unesite 4. broj: 5
Unesite 5. broj: 6
Unesite 6. broj: 7
Unesite 7. broj: 8
Unesite 8. broj: 9
Unesite 9. broj: 0
Unesite trazeni broj: 42
Broj nije sadržan u nizu
```

# Polje `length`

- Nakon kreiranja niza, broj njegovih elemenata je fiksni i više se ne može promeniti
  - Ako se niz ponovo inicijalizuje operatorom `new`, u stvari se pravi novi niz, ne proširuje se stari
- Broj elemenata niza se može dobiti pomoću `length` polja niza. Ovo polje je konstantno (`final`), tj. njegova vrednost se ne može modifikovati
- Polje `length` može da se koristi da bismo saznali koliko argumenata je korisnik naveo prilikom poziva programa
- Navedenim argumentima pristupamo pomoću jedinog parametra metoda `main`, tipa `String[]`

# Nizovi: pristup elementima - primer 5

```
class Argumenti {  
    public static void main(String[] args) {  
        if (args.length == 0) {  
            System.out.println("Niste naveli ni jedan argument.");  
        }  
        else {  
            System.out.println("Naveli ste " + args.length + " argumenata:");  
            for (int i = 0; i < args.length; i++) {  
                System.out.println(args[i]);  
            }  
        }  
    }  
}
```

# Nizovi: pristup elementima - primer 5

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java Argumenti
Niste naveli ni jedan argument.

d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java Argumenti 1 dva tri 4 pet
Naveli ste 5 argumenata:
1
dva
tri
4
pet
```