

Referencijalni tipovi

Referencijalni tipovi

- Pored prostih tipova, u Javi postoje **referencijalni tipovi**
- Referencijalni tipovi u Javi su:
 - Klase
 - Interfejsi
 - Nizovi
 - Nabrojivi tipovi (od Jave 5)
 - Lambda izrazi (od Jave 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 predmetima Programске paradigme / Programski jezici i paradigme (u drugom programskom jeziku) i OOP2

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

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

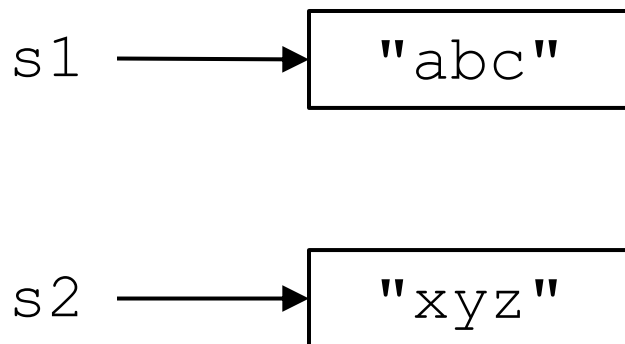


Šta znači “referencijalni”?

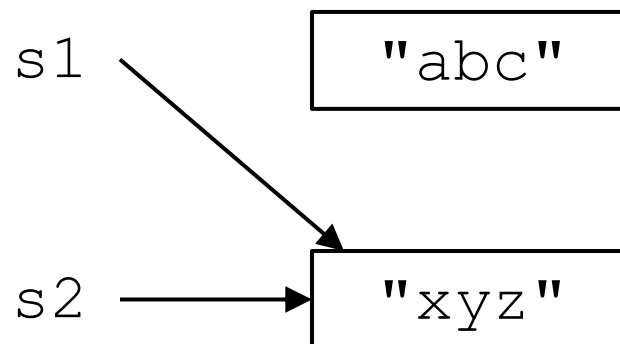
- Jednostavno rečeno, kod prostih tipova ime promenljive je zamena za **adresu**, a kod referencijalnih tipova ime promenljive je zamena za **adresu adrese**
- Promenljive prostog tipa predstavljaju jedan vid **apstrakcije**: promenljive svojim imenom apstrahuju memorijske adrese (skrivaju detalje o 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, 2, 3)

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 deklarišu 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 automatski 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) prosleđivati 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 deklariraju se na isti način kao lokalne promenljive, samo na drugom mestu:
 - Lokalne promenljive se deklariraju u okviru metoda i blokova
 - Polja klase se deklariraju 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 vrednosti 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 ();
```

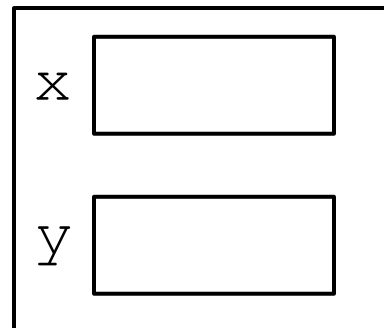
t1

Kreiranje objekata operatorom new

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

- Rezervise se memorijski prostor za objekat

t1

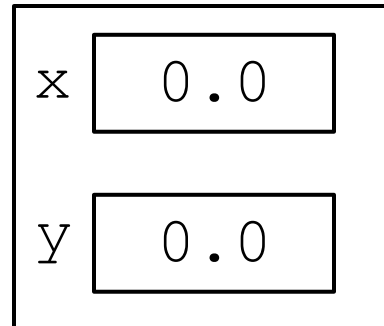


Kreiranje objekata operatorom new

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

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

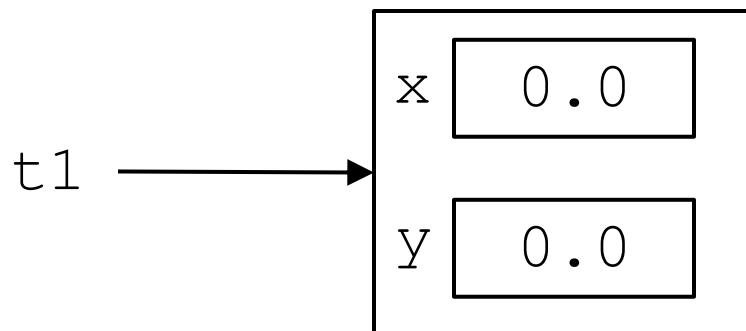
t1



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;  
    }  
}
```