Tipovi, vrijednosti i varijable u programskom jeziku *Java*

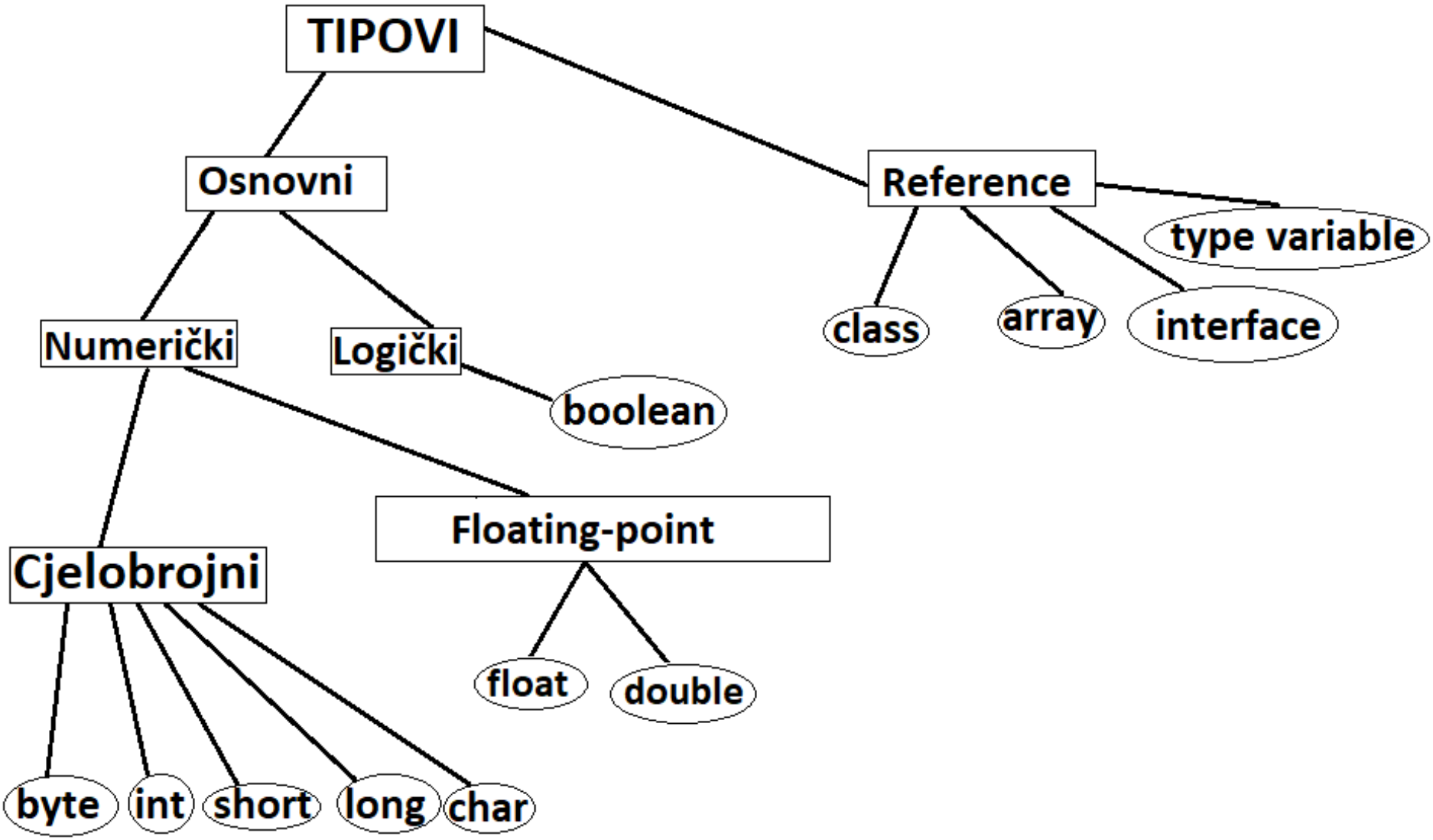
Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

11. listopada, 2022.

Tipovi u programskom jeziku *Java*



Tipovi u programskom jeziku *Java*

Numerički rasponi tipova *byte*, *short*, *int*, *long*, *char*, *float*, *double* **odgovaraju rasponima odgovarajućih tipova** kod *Java virtualnog stroja*.

Operatori na cjelobrojnim i floating-point tipovima:

- **Operatori usporedbe:** $<$, $<=$, $>$, $>=$.
- **Operatori numeričke jednakosti:** $==$, $!=$.

  Povratna vrijednost ovih operatora je tipa *boolean*.
- **Unarni plus:** $+$ (vrši numeričku promociju, npr. *byte*, *short*, *char* u *int*) i **unarni minus:** $-$ (vrši numeričku promociju i mijenja predznak broja).
- **Multiplikativni operatori:** $\cdot$ (množenje dva broja), / (dijeljenje dva broja: cjelobrojno dijeljenje kod cijelobrojnih tipova - vrši zaokruživanje prema 0), % (modulo, ostatak pri dijeljenju broja. Kod cjelobrojnih tipova broj takav da: $(a/b) \cdot b + (a\%b) = a$, kod realnih tipova $r = n - (d \cdot q)$, gdje je q najveći cijeli broj manji od n/d, istog predznaka kao n/d.

Tipovi u programskom jeziku *Java*

Operatori na cjelobrojnim i floating-point tipovima:

- **Aditivni opratori:** $+$, $-$.

  Povratna vrijednost multiplikativnih i aditivnih operatora je tipa koji odgovara najširem tipu od tipova operanada.
- **Inkrement/dekrement operatori:** $++$, $--$ (prefiks i postfiks).

  Povratna vrijednost ovih operatora je istog tipa kao ulazni operand.
- **Uvjetni operator:** ? (`izraz1`? `izraz2`:`izraz3`).
- **Cast operator:** oblika (`cast_tip`) *objekt*. Pretvara tip objekta u zadani tip `cast_tip` (može biti bilo koji numerički tip).
- **String konkatenacija:** $+$ (pretvara tip u string i konkatenira vrijednost s postojećim stringom).

Operatori na cjelobrojnim tipovima:

- **Operatori pomaka nad bitovima** $<<$ (lijevi), $>>$ (desni s predznakom), $>>>$ (desni bez predznaka).
- **Komplement nad bitovima** $\sim$.

Tipovi u programskom jeziku *Java*

Operatori na logičkom i cjelobrojnim tipovima:

- **Bit-po-bit operatori *i*, *ili* i *xor*:** $\&$, $|$, $\hat{}$.

Tip povratne vrijednosti navedenih operatora je jednaka tipu ulaznog operanda.

Operatori na logičkom tipu:

- **Relacijski operatori:** $==$, $!=$.
- **Logička negacija:** $!$.
- **Logički operatori *i*/*ili*:** $\&\&$, $||$.

  Povratna vrijednost navedenih operatora je logičkog tipa.
- **Uvjetni operator:** $?$.
- **Cast operator:** samo u tip *boolean*, *Boolean* ili *Object*.
- **String konkatenacija:** $+$.

Izvanredna stanja i iznimke kod operatora

Cjelobrojni operatori:

- Izbacuje `NullPointerException` ukoliko dolazi do otpakiravanja *null* reference.
- Operatori / i % izbacuju `ArithmeticException` ukoliko je drugi (desni) operand jednak nuli.
- Inkrement i dekrement operatori izbacuju `OutOfMemoryError` ukoliko je potrebno pakiranje typa ali nema dovoljno memorije.

Realni operatori:

- Vrijednost izraza $0.0 == -0.0$ je istina (`true`), a vrijednost izraza $0.0 > 0.0$ je laž (`false`).
- $1.0/0.0 = +\infty,\ 1.0/-0.0 = -\infty$.
- `NaN` (eng. not a number), nije broj - izvanredno stanje računanja (npr. 0.0/0.0). `NaN` je neuređen: ($<$, $<=$, $>$, $>=$, $==$) vraćaju laž ukoliko je jedan ili su oba operanda `NaN`. Operator $!=$ vraća istinu ukoliko je barem jedan operand `NaN`. $x != x$ je istina ukoliko x ima vrijednost `NaN`.

Izvanredna stanja i iznimke kod operatora

- Floating-point operacija vraća beskonkačno s predznakom ukoliko se dogodi overflow.
- Floating-point operacija vraća ne-normaliziranu vrijednost ili nulu s predznakom ukoliko se dogodi underflow.
- Floating-point operacija koja nema jedinstveno definirani matematički rezultat vraća `NaN`.
- Sve numeričke operacije koje imaju `NaN` kao operand vraćaju `NaN` kao rezultat.

Iznimke kod realnih operatora:

- Ukoliko dođe do otpakiravanja *null* reference realni operatori izbacuju `NullPointerException`.
- Inkrement i dekrement operatori izbacuju `OutOfMemoryError` ukoliko treba zapakirati vrijednost a nema dovoljno slobodne memorije.

Referencirani tipovi i vrijednosti u programskom jeziku *Java*

Referencirani tipovi su tip *klase*, *sučelja*, *polja*, *varijable tipa*. Pravila koja definiraju načine deklaracije instanci tih tipova su navedeni u nastavku.

```
ReferenciraniTip:
  TipKlaseIliSučelja
  VarijablaTipa
  TipPolja

TipKlaseIliSučelja:
  TipKlase
  TipSučelja
```

Referencirani tipovi i vrijednosti u programskom jeziku *Java*

`TipKlase:`
  `{Anotacija`[1]`}` `IdentifikatorTipa` `[ArgumentiTipa]`
  `ImePaketa . {Anotacija}` `IdentifikatorTipa` `[ArgumentiTipa]`
  `TipKlaseIliSučelja . {Anotacija}` `IdentifikatorTipa` `[ArgumentiTipa]`

`IdentifikatorTipa:`
  `Identifikator` ali ne `permits`, `record`, `sealed`, `var` ili `yield`

`ArgumentiTipa`[2]`:`
  `< ListaArgumenataTipa >`

[1] Posebne konstrukcije tipa @imeTipa(lista parova element-vrijednost). Bit će detaljnije obrađeno na kasnijim predavanjima.

[2] Definirano u nastavku predavanja.

Referencirani tipovi i vrijednosti u programskom jeziku *Java*

`TipSučelja:`
`TipKlase`

`VarijablaTipa:`
`{Anotacija} IdentifikatorTipa`

`TipPolja:`
`OsnovniTip`[3] Dim
`TipKlaseIliSučelja` Dim
`VarijablaTipa` Dim

`Dim:` `{Anotacija} [ ] {{Anotacija} [ ]}`

Tip klase ili sučelja se sastoji od **identifikatora** ili **niza identifikatora** odijeljenih točkom gdje su svakom identifikatoru potencijalno dodijeljeni identifikatori tipa (u tom slučaju se radi o parametriziranom tipu).
Ako tip klase ili sučelja ima oblik `T.id`, tada `id` treba biti oznaka dohvatljivog tipa člana od `T`.

[3] jedan od definiranih numeričkih tipova ili logički tip

Referencirani tipovi i vrijednosti u programskom jeziku *Java*

```
1
2  class Automobil{
3    int brojVrata, jacinaMotora;
4    protected String marka;
5    String boja;
6    private String identifikator;
7
8    public Automobil(){
9      brojVrata = 4;
10     jacinaMotora = 120;
11     marka = "RenaultMegan";
12     boja = "Siva";
13   }
14  }
15
16 interface Kretanje { void kreni(int duljinaPuta); }
```

Definicija klase i sučelja

Referencirani tipovi i vrijednosti u programskom jeziku *Java*

Objekt je instanca klase ili polje.

Reference su pokazivači na objekte, gdje posebna referenca *null* ne pokazuje niti na jedan objekt.

Instanca klase se kreira **posebnim izrazom** za stvaranje klase:

```
1
2 Automobil prvi = new Automobil();
```

Kreiranje instance klase

Polje se kreira izrazom za stvaranje polja:

```
1
2 double realniBrojevi[] = new double[100];
```

Kreiranje instance polja

Neki izrazi mogu implicitno kreirati instance klasa ili polja (npr. konstruktori, inicijalizacije polja, operator konkatenacije).

Klasa *Object*

Klasa *Object* je nadklasa **svih** drugih klasa u *Java* programu. **Sve klase i polja** nasljeđuju metode klase *Object*.

- Metoda `clone` se koristi za stvaranje duplikata objekta. Vrši *plitko kopiranje*. Svi članovi koji su primitivnog tipa se kopiraju, međutim kopiraju se samo reference na članove koji su objekti.
- Metoda `equals` provjerava jednakost objekata (po vrijednosti a ne po jednakosti memorijske lokacije na koju pokazuju odgovarajuće reference).
- Metoda `finalize` se aktivira prije uništavanja objekta (ekvivalent C++ destruktora).
- Metoda `getClass` vraća objekt klase koji reprezentira klasu danog objekta (postoji za svaki referencirani objekt). Pomoću objekta klase možemo saznati ime klase, članove, nadklase, sučelja koja implementira klasa.
- Metoda `hashCode` vraća hash kod objekta na kojem je pozvana.

Klase *Object*, *String* i jednakost referenciranih tipova

Metode klase *Object:*

- Metode `wait`, `notify` i `notifyAll` kod više-dretvenih programa.
- Metoda `toString` vraća string reprezentaciju objekta.

Instance klase *String* reprezentiraju **nizove** `Unicode` kodova. String objekt ima **nepromijenjivu (konstantnu) vrijednost**. **String literali i tekstualni blokovi** su reference na instance klase *String*. Operator konkatenacije stringa (+) stvori **novi** *String* objekt kada rezultat nije nepromijenjivi (konstantni, eng. `const`) izraz.

Dva referencirana tipa su **jednaka (tijekom prevođenja)** ukoliko : a) deklarirani su u kompilacijskim jedinicama (jedinicama za prevođenje) koje su asocirane s jednakim modulom, b) imaju jednako binarno ime, c) njihovi tipovni argumenti su jednaki.

Dva referencirana tipa su **jednaka (tijekom izvođenja)** ako: a) oba su tipa klase ili su oba tipa sučelja, b) definirani su od strane identičnog učitavača klase, c) imaju jednako binarno ime, d) oba su tipa polja i tipovi komponenti su jednaki tipovi prilikom izvođenja.

Varijabla tipa

Varijabla tipa je **identifikator** koji se koristi kao tip u tijelima klasa, sučelja, metoda i konstruktora.
Varijabla tipa se **definira deklaracijom parametra tipa generičke klase**, sučelja, metode ili konstruktora.

```
ParametarTipa:
  {ModifikatorParametraTipa} IdentifikatorTipa [GranicaTipa]

ModifikatorParametraTipa:
  Anotacija

GranicaTipa:
  extends VarijablaTipa
  extends TipKlaseIliSučelja {DodatnaGranica}

DodatnaGranica
  & TipSučelja
```

Varijabla tipa

Svaka varijabla tipa deklarirana kao parametar tipa **ima granicu**. Ukoliko granica nije definirana, granica je *Object*. Ukoliko je granica definirana sastoji se od: a) jedne varijable tipa `T` ili b) tipa klase ili sučelja `T` nakon kojih mogu slijediti tipovi sučelja I_1 & ... & I_n (tzv. tip presjeka).

```
1
2  class RabljeniAutomobil extends Automobil implements
       Kretanje{
3    int prijedenoKM;
4
5    RabljeniAutomobil(){
6        super();  //pozovemo konstruktor nadklase Auto
7      prijedenoKM = 0;
8    }
9
10   public void kreni(int duljinaPuta){
11           prijedenoKM+=duljinaPuta;
12   }
13 }
```

Nasljeđivanje u Javi

Varijabla tipa

```
1
2 class AutomobilTest{
3     <T extends Automobil & Kretanje> void test(T t) {
4           t.brojVrata = 4;    // OK (ako u istom paketu)
5           t.kreni(2);       // OK (public, dio sucelja)
6           t.marka = "Fiat";  // OK (u istom paketu i
    nasljeduje Automobil)
7           System.out.println(t.identifikator);  // greska kod
    prevodenja, pristup moguc samo unutar klase Automobil
8      }
9 }
```

Nasljeđivanje u Javi

`T` je **varijabla tipa** koja ima iste članove kao i tipovi članovi tipa presjeka `Automobil & Kreni`. Članovi sučelja su **uvijek javni** (`public`) i **nasljeđuju se** osim ako nisu predefinirani. Privatni (`private`) članovi su dostupni samo iz klase **unutar koje su definirani**.

Parametrizirani tipovi

Parametrizirani tip je tip klase ili sučelja oblika $C < T_1, \ldots, T_n >$, gdje je C ime generičkog tipa a $< T_1, \ldots, T_n >$ je lista argumenata tipa koji parametriziraju generički tip.
Parametri tipa $F_1, \ldots F_n$ generičkog tipa mogu imati odgovarajuće granice $B_1, \ldots, B_n$.
Parametrizirani tip $C < T_1, \ldots, T_n >$ je dobro definiran ako vrijedi:

- `C` je ime generičkog tipa.
- Broj argumenata tipa je jednak broju parametara tipa kod generičke deklaracije `C`.
- Ukoliko dođe do izmjene neodređenog/ih tipova određenim tipovima te je rezultatni tip $C < X_1, \ldots, X_n >$, tada svaki argument tipa X_i mora biti podtip tipa $S[F_1 := X_1, \ldots, F_n := X_n]$ za svaku granicu tipa S u B_i.

```
1 ArrayList<String>
2 ArrayList<Integer>
3 Pair<String,Double>
```

Primjer parametriziranih tipova

Argumenti tipa parametriziranih tipova

Argumenti tipa mogu biti **referencirani tipovi** ili **neodređeni tip** (eng. *wildcard*). Neodređeni tipovi su korisni kada je potrebna samo **djelomična informacija** o parametru tipa.

```
ArgumentiTipa:
  <ListaArgumenataTipa>

ListaArgumenataTipa:
  ArgumentTipa {,ArgumentTipa}

ArgumentTipa:
  ReferenciraniTip
  NeodređeniTip

NeodređeniTip:
  {Anotacija} ? [GranicaNeodređenogTipa]

GranicaNeodređenogTipa:
 extends ReferenciraniTip
  super ReferenciraniTip
```

Argumenti tipa parametriziranih tipova

Neodređenim tipovima možemo definirati granice:

- `? extends B` - definira gornju granicu.
- `? super B` - definira donju granicu.
- `? extends Object` je ekvivalentno neodređenom tipu `?`.

Argument tipa T_1 sadrži argument tipa T_2 u oznaci $T_2 \leq T_1$ ukoliko je skup tipova T_2 dokazivi podskup skupa tipova T_1 pod refleksivnim i tranzitivnim zatvorenjem pravila (gdje $<:$ označava podtip):

- `? extends T` $\leq$ `? extends S` ako `T` $<:$ `S`
- `? extends T` $\leq$ `?`
- `? super T` $\leq$ `? super S` ako `S` $<:$ `T`
- `? super T` $\leq$ `?`
- `? super T` $\leq$ `extends Object`
- `T` $\leq$ `T`
- `T` $\leq$ `extends T`
- `T` $\leq$ `? super T`

Argumenti tipa parametriziranih tipova

Dva tipa su dokazivo različita ako vrijedi nešto od:

- Niti jedan argument nije varijabla tipa ili neodređeni tip i dva argumenta nisu isti tip.
- Jedan argument tipa je varijabla tipa ili neodređeni tip s gornjom granicom S a drugi argument tipa T nije varijabla tipa niti neodređeni tip i ne vrijedi $|S| <: |T|$ niti $|T| <: |S|$.
- Svaki argument tipa je varijabla tipa ili neodređeni tip s gornjim granicama S i T i ne vrijedi $|S| <: |T|$ niti $|T| <: |S|$.

Za neki argument tipa S, $|S|$ označava neparametrizirani tip koji nastaje zamjenom varijabli tipa neparametriziranom gornjom granicom ili tipom *Object*.

Argumenti tipa parametriziranih tipova

```
1  public class GenericPolje<T> {
2      T podaci[];
3
4      GenericPolje(){
5          podaci = (T[])new Object[1];
6      }
7
8       GenericPolje(int duljina){
9          podaci = (T[])new Object[duljina];
10     }
11
12       public void dodijeliVrijednostSvima(T vrijednost){
13
14         for(int index=0;index<podaci.length;index++)
15                 podaci[index] = vrijednost;
16      }
17                              .
18                              .
19                              .
```

Definiranje generičke klase

Argumenti tipa parametriziranih tipova

```
1                                   .
2                                   .
3     public void dodijeliVrijednost(int index, T vrijednost)
      throws ArrayIndexOutOfBoundsException{
4         if(podaci.length<=index){
5             throw new ArrayIndexOutOfBoundsException("
    Indeks je izvan granica polja!");
6         }
7
8         podaci[index] = vrijednost;
9
10    }
11
12    public void ispisiPolje(){
13        for(int i=0;i<podaci.length;i++)
14            System.out.print(podaci[i]+" ");
15        System.out.println();
16    }
```

Definiranje generičke klase

Argumenti tipa parametriziranih tipova

```
1  public class ImenovanoPolje<T> extends GenericPolje<T> {
2      String ime;
3
4      ImenovanoPolje(){
5          super();
6          ime = "polje";
7      }
8
9      ImenovanoPolje(int n, String imePolja){
10         super(n);
11         ime = imePolja;
12     }
13
14     @Override  public void ispisiPolje(){
15         System.out.println("Ime polja: "+ime);
16         for(int i=0;i<podaci.length;i++)
17             System.out.print(podaci[i]+" ");
18         System.out.println();
19     }
```

Nasljeđivanje kod generičkih klasa

Argumenti tipa parametriziranih tipova

```
1  public class Klase {
2
3   static void ispisiObjektPolje(GenericPolje<?> c) {
4
5    System.out.println("Obj. klase:"+c.getClass().getName());
6          for (Object o : c.podaci) {
7              System.out.print(o+" "); }
8          System.out.println(); }
9
10  public static void main(String[] args) {
11    GenericPolje<Integer> polje1 = new GenericPolje(10);
12    ImenovanoPolje<Integer> polje2 = new ImenovanoPolje(10, "
       PoljeCijelihBrojeva");
13    GenericPolje<Object> polje3 = new ImenovanoPolje(10, "
       PoljeObjekata");
14
15    ispisiObjektPolje(polje1); ispisiObjektPolje(polje2);
       ispisiObjektPolje(polje3); /*sva tri poziva su OK!*/ }
16 }
```

Primjer definiranja objekata generičkih klasa i korištenja neodređenog tipa.

Fiksiranje tipova

Fiksiranje tipova (eng. *type erasure*) je postupak eliminacije varijabli tipa i parametriziranih tipova iz tipova koji ih sadrže. Parametrizirani tipovi i varijable tipa se zamijenjuju odgovarajućom granicom tipa ili nadtipom.

```
1 ImenovanoPolje d = new ImenovanoPolje();//fiksirani tip
```

Fiksiranje tipa

U navedenom primjeru se kao tip varijable `d` koristi tip koji nastaje fiksiranjem tipa `ImenovanoPolje<T>` u tip `ImenovanoPolje`. Tim postupkom se sva pojavljivanja tipa `T` mijenjaju nadtipom *Object*. Ova funkcionalnost je omogućena zbog **kompatibilnosti sa starim** (eng. *legacy*) **kodovima**. Međutim, istu funkcionalnost koristi i prevoditelj da bi tipovi bili poznati prilikom pokretanja programa (time se eliminiraju potencijalne pogreške prilikom izvršavanja). Tip `ImenovanoPolje` se naziva **nerafinirani tip** (eng. *raw type*).

Relacije nad tipovima

Relacije podtip (eng. *subtype*) i nadtip (eng. *supertype*) su **binarne relacije** nad **tipovima**.
Nadtipovi tipa se računaju iz **refleksivnog** i **tranzitivnog zatvorenja** relacije nadtipova (u oznaci $>_1$ - bit će definirana u nastavku).

$S :> T$ označava da je tip S nadtip tipa T. S je pravi nadtip od T ($S > T$) ako $S :> T$ i $S \neq T$.

Podtipovi tipa T su svi tipovi U takvi da je T nadtip od U. `null` tip je podtip tipa T. $T <: S$ označava da je tip T podtip tipa S. T je pravi podtip od S ($T < S$) ako $T <: S$ i $S \neq T$.

T je direktni podtip od S ($T <_1 S$) ako $S >_1 T$.
Relacija podtipova ne vrijedi trivijalno na parametriziranim tipovima ($T <: S \not\Rightarrow C < T >< : C < S >$).

Relacije nad tipovima

Relacije nad osnovnim tipovima:

- `double` $>_1$ `float`
- `float` $>_1$ `long`
- `long` $>_1$ `int`
- `int` $>_1$ `char`
- `int` $>_1$ `short`
- `short` $>_1$ `byte`

Relacije nad tipovima klasa i sučelja:

Za tip C koji nije generički, direktni nadtipovi tipa C su:

- Direktna nadklasa od C (ona koju klasa nasljeđuje).
- Direktna nadsučelja od C (ona koje klasa implementira).
- Tip *Object* ukoliko je C sučelje bez direktnih nadsučelja.

Relacije nad tipovima

Relacije nad tipovima klasa i sučelja:

Za generički tip $C < F_1, \ldots, F_n >$, $(n > 0)$, direktni nadtipovi nerafiniranog tipa C jesu:

- Direktna nadklasa nerafiniranog tipa C.
- Direktna nadsučelja nerafiniranog tipa C.
- Tip *Object* ukoliko je $C < F_1, \ldots, F_n >$ tip generičkog sučelja koji nema direktnih nadsučelja.

Za generički tip $C < F_1, \ldots, F_n >$, $(n > 0)$, direktni nadtipovi generičkog tipa $C < F_1, \ldots, F_n >$ jesu:

- Direktna nadklasa od $C < F_1, \ldots, F_n >$.
- Direktno nadsučelje od $C < F_1, \ldots, F_n >$.
- Tip *Object* ako je $C < F_1, \ldots, F_n >$ generičko sučelje koje nema direktno nadsučelje.
- Nerafinirani tip C.

Relacije nad tipovima

Za deklaraciju generičkog tipa $C < F_1, \ldots, F_n >$, $(n > 0)$, direktni nadtipovi generičkog parametriziranog tipa $C < T_1, \ldots, T_n >$ (gdje su $T_i, 1 \leq i \leq n$ jesu:

- $D < U_1\Theta, \ldots, U_k\Theta >$, gdje je $D < U_1, ..., U_k >$ generički tip koji je direktni nadtip generičkog tipa $C < F_1, \ldots, F_n >$, Θ je substitucija $[F_1 := T_1, ..., F_n := T_n]$.
- $C < S_1, \ldots, S_n >$, gdje S_i sadrži T_i $(1 \leq i \leq n)$.
- Tip *Object* ako je $C < F_1, \ldots, F_n >$ generičko sučelje bez direktnih nadsučelja.
- Nerafinirani tip C.

Za generički tip $C < F_1, \ldots, F_n >$, $(n > 0)$, direktni nadtipovi parametriziranog tipa $C < R_1, \ldots, R_n >$ (gdje je barem jedan $R_i, 1 \leq i \leq n$ neodređeni tip) je direktni nadtip parametriziranog tipa $C < X_1, ..., X_n >$ koji nastaje zamjenom neodređenih tipova određenim tipom.

Relacije nad tipovima

Direktni nadtipovi tipa presjeka $T_1 \& \ldots \& T_n$ su $T_i (1 \leq i \leq n)$.

Direktni nadtipovi varijable tipa su njezini tipovi gornje granice.

Varijabla tipa je direktni nadtip svojih donjih granica.

Svi referencirani tipovi (osim `null`) su nadtipovi tipa `null`.

Relacije nad tipovima polja:

- Ako su S i T referencirani tipovi, tada $S[] >_1 T[]$ ako i samo ako $S >_1 T$.
- `Object` $>_1$ `Object[]`
- `Cloneable` $>_1$ `Object[]`
- `java.io.Serializable` $>_1$ `Object[]`

Za osnovni tip P:

- `Object` $>_1$ `P[]`
- `Cloneable` $>_1$ `P[]`
- `java.io.Serializable` $>_1$ `P[]`

Varijable

Varijabla je **oznaka** za **memorijsku lokaciju**. Ona ima **dodijeljen tip** koji može biti **osnovni** ili **referencirani**.

Vrijednost varijable se mijenja **pridruživanjem**, **primjenom prefiks/postfiks inkrement** ili **dekrement operatora**.

Inicijalne vrijednosti varijabli određenog tipa odgovaraju inicijalnim vrijednostima odgovarajućeg tipa kod *Java virtualnog stroja*.

Varijable osnovnog tipa sadrže **vrijednost** tog osnovnog tipa. Varijable tipa klase `T` sadrže ili `null` **referencu**, **referencu** na instancu neke klase `T` ili **referencu** na instancu bilo koje podklase klase `T`. Varijable tipa sučelja sadrže `null` **referencu** ili **referencu** na bilo koju klasu koja implementira to sučelje.

Ako je T osnovni tip, tada varijabla tipa *polje od* T može biti `null` **referenca** ili **referenca** na polje tipa *polje od* T ili **referenca** na polje tipa *polje od* S, gdje je S podklasa ili podsučelje tipa T.

Varijable

Varijabla tipa `Object[]` može sadržavati referencu na polje bilo kojeg referenciranog tipa.

Varijabla tipa `Object` može sadržavati `null` referencu, referencu na proizvoljni objekt (instancu klasa ili polje).

Moguće je da varijabla parametriziranog tipa referencira objekt koji nije tog parametriziranog tipa. Taj slučaj se naziva **onečišćenje hrpe**.

```
1 List l = new ArrayList<Number>();
2 List<String> ls = l;  // Unchecked warning
```

Onečišćenje hrpe

Do onečišćenja hrpe može doći **samo onda kada koristimo nerafinirani tip** (kao u primjeru). Pošto nerafinirani tip **ne sadrži informaciju o parametrima tipa**, pri prevođenju se **ne može detektirati potencijalna povreda tipova** pri pridruživanju referenci (no prevoditelj izbacuje `Unchecked warning` - **upozorenje o neprovjerenom pretvaranju tipova** koje može dovesti do iznimaka).

Vrste varijabli

U programskom jeziku *Java* postoji 8 tipova varijabli:

1. **varijabla klase** je član unutar definicije klase deklariran ključnom riječju `static` ili član unutar definicije sučelja deklariran sa ili bez ključne riječi `static`. Kreira se zajedno s klasom ili sučeljem, inicijalizira na početnu vrijednost, a uništava se pri isključenju klase ili sučelja.
2. **Varijabla instance** je član deklariran unutar definicije klase bez ključne riječi `static`. Kreira se i inicijalizira kod kreiranja svakog novog objekta klase ili podklase. Uništava se kada objekt kojem pripada više nije referenciran te nakon što je izvršena finalizacija objekta (ekvivalent C++ destruktora).
3. **Komponente polja** su neimenovane varijable koje se kreiraju i inicijaliziraju na početne vrijednosti kod kreiranja novog objekta polja. Uništavaju se kada polje više nije referencirano u programu.

Vrste varijabli

4. **Parametri metoda** označavaju vrijednosti argumenata koji se proslijeđuju metodi. Parametri deklarirani kod definicije metode se stvaraju ponovo pri svakom pozivu metode i inicijaliziraju s vrijednosti argumenta. Uništavaju se nakon završetka poziva metode.
5. **Parametri konstruktora** označavaju vrijednosti argumenata konstruktora. Kod svakog kreiranja nove instance klase ili eksplicitnog poziva konstruktora, stvaraju se nove varijable inicijalizirane vrijednostima odgovarajućih argumenata. Parametri konstruktora se uništavaju pri završetku izvođenja tijela konstruktora.
6. **Lambda parametri** oznavaju vrijednosti argumenata proslijeđenih lambda izrazu. Kod svakog poziva metode implementirane lambda izrazom, stvaraju se novi parametri koji se inicijaliziraju vrijednostima odgovarajućih argumenata. Lambda parametri se uništavaju pri završetku izvođenja izraza.

Vrste varijabli

7. **Parametar iznimke** se stvara svaki puta kada je iznimka uhvačena `catch` izrazom. Nova varijabla je inicijalizirana stvarnim objektom asociranim s iznimkom. Parametar iznimke se uništava nakon završetka izvođenja dijela koda unutar `catch` izraza.
8. **Lokalne varijable** su deklarirane izrazima deklaracije lokalnih varijabli. Stvaraju se kada se počne izvršavati dio koda (odvojeni blok) ili `for` petlja unutar kojih su deklarirane varijable. Lokalne varijable se mogu inicijalizirati i izrazom, međutim ne mogu se koristiti dok nije izvršen dio koda koji inicijalizira varijablu. Uništavaju se nakon izlaska iz odgovarajućeg bloka koda ili tijela `for` petlje.