Izrazi u programskom jeziku *Java*

Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

8. studenoga, 2022.

Tip nakon evaluacije izraza

Ukoliko izraz označava **vrijednost ili varijablu**, tada je **tip koji dobijemo nakon evaluacije izraza poznat prilikom prevođenja**. Tip nakon evaluacije **samostojećih izraza** se **može utvrditi iz sadržaja izraza** dok tip koji dobijemo evaluacijom **izraza ovisnih o kontekstu može ovisiti o ciljnom tipu izraza**. **Vrijednost izraza je kompatibilna s povratnim tipom izraza** (osim kod zagađenja hrpe), analogno je i **vrijednost spremljena u varijablu uvijek kompatibilna s tipom varijable**. Vrijednost izraza čiji je tip T se uvijek može dodijeliti varijabli tipa T.

Izraz **nije** *FP*-strog (kod Java verzija <17) ako i samo ako **nije konstantan izraz** i **ne javlja se niti u jednoj deklaraciji** koja ima **oznaku** `strictfp`. Kod *FP*-strogih izraza, **svi međurezultati moraju biti** tipa *float* ili *double* (`IEEE` 754 aritmetika na operandima reprezentiranim formatima `binary32` i `binary64`). Kod izraza koji **nisu** *FP*-strogi se **može koristiti prošireni raspon exponenta** da se prikažu međurezultati (omogućava izračun izraza kod kojih bi inače dobili *overflow* - preveliki brojevi po apsolutnoj vrijednosti ili *underflow* - jako mali brojevi - po apsolutnoj vrijednosti blizu nula).

Izrazi i provjere pri izvršavanju

Ukoliko je vrijednost nekog izraza **referenciranog tipa**, tada klasa referenciranog objekta i/ili je li on `null` ili referenca na objekt **nije nužno poznato pri prevođenju**.

Situacije u kojima klasa referenciranog objekta utječe na izvršavanje na način koji se ne može utvrditi iz tipa očekivane vrijednosti izraza pri prevođenju:
- Takva situacija se **često javlja kod poziva metoda**, gdje se metoda koja se poziva određuje iz **imena metoda koji su članovi klase ili sučelja odgovarajućeg tipa** (utvrđenog pri prevođenju), ali i iz **klase koje odgovaraju tipu objekta koji je referenciran pri izvršavanju** od strane objekta (potencijalna podklasa koja nadjačava ili predefinira metodu).
- operatorom `instanceof` se može **provjeriti klasa objekta koji je referenciran pri izvođenju od vrijednosti izraza**. Služi za **ispitivanje mogućnosti konverzije u neki drugi referencirani tip**.

Izrazi i provjere pri izvršavanju

- **Pretvorba** - **klasa objekta** koji je referenciran pri izvođenju od strane vrijednosti operanda izraza **može biti nekompatibilna s tipom navedenim u operatoru pretvorbe**. Kod referenciranih tipova to **može dovesti do provjera pri izvršavanju** koje **mogu prijaviti iznimku** (u slučaju da konverzija nije moguća).
- **Pridruživanje vrijednosti komponenti polja referenciranog tipa** može dovesti do provjere pri izvršavanju (koja može proizvesti iznimku). Razlog je što `S[]` možemo smatrati podtipom od `T[]` ako je S podtip od T.
- **Obrada iznimaka** - iznimka se obrađuje od strane `catch` **samo ako je klasa objekta prijavljene iznimke** `instanceof` tipa **formalnog parametra** `catch` bloka naredbe.

Ukoliko nisu **statički poznate** klase objekata, **može doći do grešaka pri izvođenju**.

Postoje slučajevi kada **statički poznati tip nije točan pri izvođenju** (kod programa koji vraćaju `unchecked warning`).

Izrazi i provjere pri izvršavanju

Do njega dolazi jer se **statički ne može utvrditi da je operacija sigurna** a **dinamička provjera nije moguća jer neke informacije o tipu nedostaju** (npr. kod fiksiranja tipa).

Do greške pri izvođenju može doći u sljedećim slučajevima:
- **Kod pretvorbe** kada klasa objekta referenciranog od vrijednosti operanda izraza **nije kompatibilna** s ciljnim tipom operatora pretvorbe (dolazi do `ClassCastException`).
- **Kod automatske pretvorbe** koja je uvedena da bi se osiguralo ispravno izvođenje operacije na tipu koji sadrži nepotpune informacije.
- **Kod pridruživanja vrijednosti komponenti polja referenciranog tipa**. Događa se kada **klasa objekta koji je referenciran od strane vrijednosti koju treba pridijeliti** nije kompatibilna sa **stvarnim tipom pri izvođenju komponente polja** (dolazi do `ArrayStoreException`).
- **Kada iznimka nije obrađena niti od jednog** `catch`-a. Tada **dretva prvo pokuša pozvati proceduru za obradu neobrađenih iznimki** nakon čega **prekida izvođenje**.

Normalno i neočekivano izvođenje izraza

Svaki izraz ima **normalan način izvrednjavanja** (ukoliko nije došlo do iznimke niti u jednom koraku). Inače, **izvrednjavanje završava neočekivano** (razlog se prijavljuje koristeći `throw`).
Iznimke pri izvođenju se prijavljuju od sljedećih operatora:
- a) Izraz kreiranja instance klase, b) izraz kreiranja polja, c) izraz reference metode (dohvata), d) izraz inicijaliziranja polja, e) izraz operatora konkatenacije stringova, f) lambda izraz, vraćaju `OutOfMemoryError` uoliko nema dovoljno dostupne memorije.
- Izraz stvaranja polja prijavljuje `NegativeArraySizeException` (ukoliko je vrijednost izraza dimenzije manja od nule), `NullPointerException` ukoliko je vrijednost reference izraza `null`.
- Izraz pristupa polju prijavljuje `ArrayIndexOutOfBoundsException` ukoliko je vrijednost indeksa negativna ili veća (jednaka) duljini polja.
- izraz pristupa elementu članu klase i izraz poziva metode javljaju `NullPointerException` ako je vrijednost reference objekta (odnosno ciljne metode) `null`.

Normalno i neočekivano izvođenje izraza

- Izraz pretvorbe tipa prijavljuje `ClassCastException` ukoliko se pretvorba ne može provesti tijekom izvršavanja.
- Dijeljenje cijelih brojeva ili ostatak prijavljuju `ArithmeticException` ukoliko je vrijednost desnog operanda izraza 0.
- Pridruživanje vrijednosti komponenti polja referenciranog tipa, izraz poziva metode, prefix/postfix inkrement i dekrement operator mogu prijaviti `OutOfMemoryError` kao rezultat pakiranja.
- Prilikom pridjeljivanja vrijednosti komponenti polja koja je referenciranog tipa može doći do `ArrayStoreException` kada se pridjeljuje vrijednost neodgovarajućeg tipa.
- `Switch` izraz prijavljuje `IncompatibleClassChangeError` ukoliko niti jedna switch labela/case slučaj ne odgovaraju uvjetu izbora.

Normalno i neočekivano izvođenje izraza

Izraz poziva metode može prijaviti iznimku i ako se iznimka javi negdje unutar tijela metode (prouzrokuje neočekivani kraj izvođenja). Isto vrijedi i za izraz stvaranja instance ukoliko dođe do iznimke pri izvršavanju konstruktora.
Prilikom izvrednjavanja izraza može doći i do raznih grešaka pri povezivanju te grešaka virtualnog stroja (teško se obrađuju).
Ukoliko dođe do iznimke, može doći do neočekivanog prekida izvršavanja nekoliko izraza.
Ukoliko dođe do iznimke prilikom izvršavanja podizraza, tada dolazi i do neočekivanog prekida izvršavanja izraza (kao i daljnjih koraka dok ne dođe do obrade iznimke).

Redoslijed izvrednjavanja izraza

Java **garantira** da se operandi operatora **izvrednjavaju u definiranom poretku (s lijeva na desno)**.

Izvrednjavanje operanda s lijeve strane:
Operand s lijeve strane binarnog operatora se **u potpunosti izvrijedni prije nego što počne izvrednjavanje desnog operanda** (uslijed iznimke kod izvrednjavanja lijevog operanda, desni se ne izvrijedni). Kod složenih operatora pridruživanja (npr. $+=$), izvrednjavanje lijevog operanda **uključuje spremanje varijable tog operanda**, **dohvaćanje i spremanje njezine vrijednosti** za **korištenje kod operacije**.

```
1 int i = 2;
2 int j = (i=3) * i;//prvo se izvrsi i=3, zatim 3*3
3 System.out.println(j);//ispise 9
4 i += (i = 2);  // prvo se dohvati i zapamti i=3
5 System.out.println(a);// ispis 5: i=(3+2)
6 j = j + (j = 3);  // prvo lijevi operand
7 System.out.println(j);// j=9+3 = 12
```

Izvrednjavanje izraza - lijevi operand.

Redoslijed izvrednjavanja izraza

```
1  int j = 1;
2          try {
3              int i = fIzn() / (j = 2);//prijavljuje se
       iznimka (java.lang.Exception: "Kraj izvrsavanja!")
4          } catch (Exception e) {
5              e.printStackTrace();
6              System.out.println("Vrijednost: j = " + j);
7          }//ispisuje j = 1
8
9      static int fIzn() throws Exception {
10         throw new Exception("Kraj izvrsavanja!");
11     }
```

Izvrednjavanje izraza - lijevi operand.

Izvrednjavanje operanada prije operacije:
Java **garantira** da će se **svaki operand u potpunosti izvrijediti prije izvršavanja operatora**. To vrijedi za sve operatore osim &&, ||, ? :.

Redoslijed izvrednjavanja izraza

Binarni operatori / i % mogu javiti `ArithmeticException`. Ta iznimka se javlja tek nakon izvrednjavanja oba operanda (ukoliko izvrednjavanje završi normalno).

```
1   int j = 0;
2         try {
3             int i = 1 / (j * fIzn());
4         } catch (Exception e) {
5             System.out.println(e);
6         }
7     }
```

Izvrednjavanje izraza - operandi.

Gornji program ispisuje `java.lang.Exception: "Kraj izvrsavanja!"`, a ne `java.lang.ArithmeticException: "/ by zero"`.

Redoslijed izvrednjavanja izraza

Izvrednjavanje poštuje zagrade i prioritete:
Java **poštuje poredak izvrednjavanja određen zagradama ili prioritetom operatora.**

Implementacija jezika **ne smije koristiti algebarske identitete** (npr. asocijativnost) da zapiše izraz na drugačiji način, **osim ako se ne može dokazati da je izmijenjeni izraz ekvivalentan po vrijednosti početnom** (čak i slučaju višedretvenog programa) za sve moguće vrijednosti. Ukoliko se radi o aritmetici realnih brojeva, to se odnosi i na `NaN` i `infinity`. Npr. $!(x < y)$ i $x \geq y$ daju različite vrijednosti ukoliko su x, y ili oba `NaN`. *Java* prevodioc neće izraz $4.0 \cdot x \cdot 0.5$ evaluirati kao $2.0 \cdot x$ (zato što prvi izraz vraća `infinity`, dok drugi izraz vraća konačnu vrijednost za $x = 8e + 307$). Zbrajanje **cijelih** brojeva je **dokazivo asocijativno** u *Javi* (kao i u *C*-u).

Redoslijed izvrednjavanja izraza

Argumenti se izvrednjavaju s lijeva na desno:
Argumenti i izrazi argumenata se **mogu javljati kod poziva metoda**, **konstruktora** ili **kreiranja instanci klase** (mogu se javljati i unutar zagrada). Svaki argument se **izvrijedni prije nego počne izvrednjavanje argumenta s njegove desne strane**. Ukoliko izvrednjavanje izraza argumenata **završi neočekivano**, **niti jedan izraz argumenata s njegove desne strane neće biti izvrijednjen**.

```
1 public static void main(String[] args) {
2         String s = "tekst, ";
3         ispis(s, s, s = "kraj!");//izrazi argumenata
4     }
5
6  static void ispis(String a, String b, String c) {
7         System.out.println(a + b + c);//tekst, tekst, kraj!
8     }//String je immutable! (svaka promjena stvara novi
    objekt, isto vrijedi za Integer, Double, Float itd.)
```

Evaluacija izraza argumenata.

Redoslijed izvrednjavanja izraza

```
1  public class Broj {
2      int broj;
3
4      Broj(){broj = 0;}
5
6          public Broj set(int b){
7              broj = b;
8              return this;
9      }
10
11     @Override public String toString(){
12         return broj+""; }
13 }
14
15  Broj b = new Broj(); b.set(45);
16  ispis(b,b,b);//45 45 45
17  ispis(b,b.set(4),b);//4 4 4 -> Zasto????
```

Evaluacija izraza argumenata.

Redoslijed izvrednjavanja izraza

```
1     static int id;
2     public static void main(String[] args) {
3         try {
4             test(id = 1, fIzn(), id = 3);
5         } catch (Exception e) {
6             System.out.println(e + ", id=" + id);
7         }//java.lang.Exception: "Kraj izvrsavanja!", id=1
8     }
9     static int test(int a, int b, int c) {
10        return a + b + c;
11    }
```

Evaluacija izraza argumenata.

Redoslijed izvršavanja ostalih izraza:

Kod **izvrednjavanja izraza** kao što su izraz kreiranja instance klase, izraz kreiranja polja, izraz pristupa polju, izraz poziva metode, izraz reference metode, pridruživanje komponenti polja, lambda izrazi može doći do iznimaka. U takvim stanjima, **redosljed izvršavanja može biti bitno drugačiji od uobičajenog**. Kod **nomalnog izvođenja**, evaluacija tih izraza slijedi **prije navedena pravila**.

Primarni izrazi

Primarni izrazi su **najednostavniji izrazi** od kojih se **grade svi drugi izrazi**. Ovdje spadaju **literali**, **kreiranje objekata**, **pristup elementima članovima klase**, **poziv metoda**, **reference metoda**, **pristup poljima**, **izrazi u zagradama**.

Literal označava fiksnu nepromijenjivu vrijednost. Cjelobrojni literal koji završava oznakom 'l' ili 'L' je tipa *long*, ostali su tipa *int*. Realni literal koji završava oznakom 'f' ili 'F' je tipa *float*, inače je tipa *double*. Logički, znakovni i string literali su tipa *boolean*, *char* i *String* dok je tip `null` literala *null* tip. **Izvrednjavanje literala uvijek završava normalno**.

Literali klase:
Literal klase je izraz koji se sastoji od imena klase, sučelja, polja, osnovnog tipa ili riječi void, točke i ključne riječi `class`.

Ime tipa mora označavati klasu ili sučelje koje je dostupno, inače dolazi do greške pri prevođenju.

Primarni izrazi

```
1 System.out.println(Integer.class);// class java.lang.Integer
2 System.out.println(int.class);// int
3 System.out.println(void.class);// void
4 System.out.println(java.util.ArrayList.class);// class java.
    util.ArrayList
```

Literal klase.

this:
Ključna riječ `this` se **može koristiti u sljedećim kontekstima**: a) tijelo metode instance ili `default` metode, b) tijelo konstruktora klase, c) u kodu inicijalizacije klase, d) u kodu inicijalizacije instance varijable klase, e) za označavanje parametra primatelja. Ukoliko se javi u drugim kontekstima **dolazi do greške pri prevođenju**. `this` se **može koristiti unutar lambda izraza** ali **samo u dozvoljenim kontekstima**. `this` predstavlja **vrijednost reference na objekt za koji je pozvana metoda instance ili** `default` **metoda**. Može se koristiti i kod eksplicitnog poziva konstruktora.

Primarni izrazi

```
1  class Tocka {
2      int x, y;
3      Tocka(int x, int y) { this.x = x; this.y = y; }
4  }
5  class ObojanaTocka extends Tocka {
6   static final int BIJELA = 0, CRNA = 1;
7   int boja;
8   ObojanaTocka(int x, int y) {
9     this(x, y, boja);  // promijeni boju, eksplicitni poziv
       konstruktora
10 }
11  ObojanaTocka(int x, int y, int boja) {
12    super(x, y);//poziv konstruktora nadklase
13    this.boja = boja;//koristenje this unutar konstruktora
14     }
15 }
```

Eksplicitni poziv konstruktora.

Ukoliko želimo pristupiti `this` pokazivaču **vanjske klase** iz **unutarnje (ugniježdene)** klase tada navodimo ImeTipa.`this`.

Izraz kreiranja instance klase

Anonimna klasa je **klasa koja se deklarira u sklopu izraza kreiranja instance klase**. **Nije joj moguće deklarirati konstruktor** (ima implicitno deklarirani anonimni konstruktor).

```
1  public interface Poruka {
2      public void poruka(String s);
3      public void poruka();
4  }
5
6  Poruka pz = new Poruka(){
7          String poruka = "Poruka ove anonimne klase!";
8          public void poruka(String s){
9              poruka = s;
10             System.out.println(poruka);
11         }
12         public void poruka(){ System.out.println(poruka); }
       };
13     pz.poruka();
14     pz.poruka("Nova poruka");
```

Stvaranje anonimne klase.

Izraz kreiranja instance klase

Izraz kreiranja instance klase se **evaluira pri izvršavanju**.

- Određivanje klase - `NullPointerException` ukoliko se radi o null tipu, neočekivano završavanje ukoliko dođe do iznimke pri određivanju.
- Kreiranje instance klase - `OutOfMemoryError` ukoliko nema dovoljno memorije za spremanje objekta.
- Inicijalizacija elementa članova novo kreirane instance klase.
- Izvrednjavanje argumenata konstruktora (uvažavajući sva pravila o redosljedu izvršavanja i mogućim iznimkama).
- Izabir odgovarajućeg konstruktora (po pravilima o izabiru odgovarajućeg konstruktora).
- Vrijednost izvrednjavanja izraza je referenca na novo stvoreni objekt zadane klase. Kod svakog poziva se stvara nova instanca objekta.

Izraz kreiranja polja i pristupa

Izraz kreiranja polja se **evaluira pri izvršavanju**.

- Ukoliko nema izraza dimenzija, tada vrijednost izraz kreiranja polja postaje inicijalizirana vrijednost.
- Inače se izvrijedne izrazi dimenzija (s lijeva na desno). Ukoliko dođe do neočekivanog završetka, preostali izrazi se ne evaluiraju.
- Vrši se provjera vrijednosti izraza dimenzija. Ukoliko je bilo koja od tih vrijednosti manja od nula prijavljuje se `NegativeArraySizeException`.
- Alocira se memorija za novo polje. Ukoliko nema dovoljno memorije prijavljuje se `OutOfMemoryError`.
- Ukoliko se radi o jednodimenzionalnom polju, stvara se polje i elementi se inicijaliziraju na standardne vrijednosti (ovisno o tipu).
- Kod višedimenzionalnih polja s n dimenzija, izvršava se skup ugniježdenih petlji dubine $n - 1$ da bi se stvorila odgovarajuća polja koja sadrže polja (ona nisu nužno jednakih veličina na svakoj razini).

Evaluacija može prijaviti `OutOfMemoryError`.

Izraz kreiranja polja i pristupa

Izraz pristupa polju se **evaluira pri izvršavanju**.

- Prvo se izvrednjuje izraz reference polja. Ukoliko dođe do neočekivanog izvršavanja, tada i pristup polju završi neočekivano i izraz indeksa se ne izvrednjuje.
- Inače se izvrednjuje izraz indeksa. Ukoliko dođe do iznimke, pristup polju završava neočekivano.
- Ukoliko je vrijednost izraza reference polja `null` tada se prijavljuje `NullPointerException`.
- Ukoliko je vrijednost indeksa izraza manja od nula ili veća/jednaka duljini polja, prijavljuje se `ArrayIndexOutOfBoundsException`.
- Inače je rezultat pristupa polju varijabla tipa T koja se izabire prema vrijednosti izraza indeksa.

```
1 int[] a = { 11, 12, 13, 14 }, b = { 0, 1, 2, 3 };
2 System.out.println(a[(a=b)[3]]);//izvrijedni izraz reference
      i dohvati a. Izvrijedni a=b, dohvati 3 = b[3], stoga je
      ispis a[3]=14.
```

Izvrednjavanje pristupa polju.

Izraz pristupa elementima članovima klase

Izraz pristupa elementima članovima klase **može pristupiti elementu objekta ili polja** čija referenca je vrijednost izraza ili ključna riječ super.

```
1  class Planina {
2      static String planina = "Velebit";
3      static Planina velika(){
4          System.out.print("Sjeverni ");
5          return null;
6      }
7      public static void main(String[] args) {
8          System.out.println(velika().planina);
9      }
10 }
```

Pristup statičkom članu preko null reference.

```
1 class S           { int x = 0; int z() { return x; } }
2 class T extends S { int x = 1; int z() { return x+1; } }
3  T t = new T(); System.out.println(t.z()+" "+t.x); //2 1
4  S s = new S(); System.out.println(s.z()+" "+s.x); //0 0
5  s = t; System.out.println(s.z()+" "+s.x); //2 0
```

Poziv metode preko bazne klase.

Izraz poziva metode

Izraz poziva metode je **ovisan o kontekstu** ako **vrijedi sve od navedenog**:
- poziv se odvija u kontekstu pridjeljivanja ili kontekstu poziva.
- Ukoliko se koristi poziv punim imenom (može sadržavati ime klase, izraz, tip itd.) tada se argumenti tipa koji se navode s lijeve strane identifikatora ispuštaju.
- metoda koja se poziva je generička i ima povratni tip koji sadrži barem jedan parametar tipa metode.

U svim drugim situacijama, poziv metode je **samostojeći** izraz.

Dohvaćanje imena metode pri prevođenju je **kompliciranije** od dohvaćanja imena polja zbog **mogućnosti preopterečivanja metoda**. Poziv metode je **kompliciraniji** i pri izvođenju zbog **mogućnosti nadjačavanja metode**. Određivanje metode koja će se pozvati od strane izraza poziva metode uključuje **nekoliko koraka**.

Izraz poziva metode

1. korak pri prevođenju - određivanje klase ili sučelja za pretraživanje.
- Ime metode je definirano imenom ili identifikatorom koji se nalazi prije lijeve zagrade kod poziva metode.
- Postoji šest mogućnosti za pretragu klase ili sučelja ovisno o izrazu koji se nalazi ispred lijeve zagrade:

- Ukoliko je izraz `ImeMetode` (samo identifikator), tada:
  Ukoliko se identifikator javlja u dosegu deklaracije metode s tim imenom tada:
  - Ukoliko postoji deklaracija tipa koji obuhvaća metodu i metoda je član tog tipa, neka je T najdublja unutarnja deklaracija tipa. Tražena klasa ili sučelje je T (tzv. **pravilo češlja** - prvo pretraga ugniježdene klase i hijerarhije njezinih nadklasa, a zatim pretraga klase koja ju obuhvaća i njezine hijerarhije nadklasa).
  - Inače metoda može biti u dohvatu zbog **jednog ili više statičkih uključivanja** ili **deklaracija uključivanja na zahtijev**. U tom slučaju se ne vrši pretraga za klasom ili sučeljem pošto se metoda za poziv utvrđuje kasnije.

Izraz poziva metode

- Ukoliko je izraz oblika `ImeTipa . [ArgumentiTipa] Identifikator`, tada treba pretražiti tip `ImeTipa`.
- Ukoliko je izraz `ImeIzraza.[ArgumentiTipa] Identifikator`, tada je klasa ili sučelje za pretragu deklarirani tip T varijable označene s `ImeIzraza` ukoliko je T tip klase ili sučelja ili gornja granica od T ukoliko je T varijabla tipa.
- Ukoliko je izraz oblika `Primarni.[ArgumentiTipa] Identifikator`, tada je T tip primarnog izraza. Klasa ili sučelje za pretragu je T ukoliko je T tip klase ili sučelja ili gornja granica od T ukoliko je T varijabla tipa. Ukoliko T nije referencirani tip, dolazi do greške pri prevođenju.
- Ukoliko je izraz oblika `super.[ArgumentiTipa] Identifikator`, tada je klasa za pretragu nadklasa klase koja sadrži poziv metode. Tip koji neposredno sadrži poziv metode nesmije biti klasa `Object` ili sučelje.

Izraz poziva metode

- Ukoliko je izraz oblika `ImeTipa.super.[ArgumentiTipa]` tada:
  - Ukoliko `ImeTipa` nije niti klasa niti sučelje dolazi do greške pri prevođenju.
  - Ukoliko `ImeTipa` označava klasu C, tada je klasa za pretragu nadklasa od C. Ukoliko C nije obuhvaćajuća klasa trenutne ili je klasa `Object` dolazi do greške pri prevođenju. Neka je T tip deklaracije koja neposredno obuhvaća poziv metode. Ukoliko je T klasa `Object` dolazi do greške pri prevođenju.
  - Inače `ImeTipa` označava sučelje I za pretragu. Neka je T deklaracija tipa koja neposredno obuhvaća poziv metode. Ukoliko I nije direktno nadsučelje od T ili ako postoji neka druga nadklasa ili nadsučelje od T, J, takvo da je J podtip od I, tada dolazi do greške pri prevođenju.

Izraz poziva metode

```
1  class Nadklasa {
2    void ispis() { System.out.println("Bok!"); }
3  }
4
5  class Podklasa1 extends Nadklasa {
6   void ispis() { throw new UnsupportedOperationException(); }
7
8      Runnable pokreni = new Runnable() {
9        public void run() {
10             Podklasa1.super.ispis();  // "Bok!"
11         }
12     };
13 }
```

Poziv metode nadklase unutar anonimne klase.

Izraz poziva metode

```
1  interface Nadsucelje {
2      default void ispis() { System.out.println("Bok!"); }
3  }
4
5  class Podklasa2 implements Nadsucelje {
6    public void ispis() { throw new
       UnsupportedOperationException(); }
7
8     public void pokreni() {
9           Nadsucelje.super.ispis();  // "Bok!"
10     }
11 }
```

Poziv metode nadsučelja.

Izraz poziva metode

```
1  class Podklasa3 implements Nadsucelje {
2      public void ispis() { throw new
       UnsupportedOperationException(); }
3
4      void pozoviNadSucelje(){ Nadsucelje.super.ispis(); }
5
6    Runnable pokreni = new Runnable() {
7      public void run() {
8      //Podklasa3.Nadsucelje.super.pozoviNadSucelje();  //
       Nije dozvoljeno
9        Podklasa3.this.pozoviNadSucelje();//rjesenje je
       definicija nove privatne funkcije unutar Podklase3 koja
       poziva funkciju nadsucelja. Ta funkcija se moze pozvati
       unutar ove anonimne klase.
10         }
11     };
12 }
```

Poziv metode nadsučelja iz anonimne ugniježdene klase.

Izraz poziva metode

2. korak pri prevođenju - određivanje prototipa metode.
Drugi korak **pretražuje metode članice tipa određenog u prijašnjem koraku**. Koristi **ime metode** i **izraze argumenata** da bi locirao metode koje su **dohvatljive i primijenjive** (mogu se korektno pozvati s danim parametrima). Ukoliko postoji više takvih metoda, uzima se **najspecifičnija**. **Prototip i povratna vrijednost najspecifičnije metode** se koriste pri izvršavanju za poziv. **Samo izrazi argumenata**, ne i ciljni izraz poziva utječu na test primijenjivosti metode.
- Određivanje primijenjivih metoda.
- Verifikacija kompatibilnosti:
  - Prva faza - pretraga predefiniranih metoda bez dopuštanja pakiranja i otpakiravanja ili pozivanja metode s varijabilnim brojem argumenata. Ukoliko nije pronađena odgovarajuća metoda, prelazi se na drugu fazu.
  - Druga faza - pretraga predefiniranih metoda uz dopuštanje pakiranja i otpakiravanja, bez dopuštanja metoda s varijabilnim brojem argumenata. Ukoliko nije pronađena odgovarajuća metoda, prelazi se na treću fazu.
  - Treća faza - dopušta kombiniranje predefiniranja i metoda s varijabilnim brojem argumenata, pakiranjem i otpakiravanjem.

Izraz poziva metode

Metoda je primjenjiva ako se može pozvati na **strogi način**, na **relaksirani način** ili kao **poziv funkcije s varijabilnim brojem argumenata**. Analiza generičkih metoda zahtjeva analizu **argumenata tipa**. Ukoliko se šalju implicitno, treba odrediti i **odgovarajuće granice argumenata** tipa iz izraza argumenata.

Određivanje primijenjivih metoda:
- Ime metode članice mora biti identično imenu pozvane metode.
- Član je dostupan klasi ili sučelju u kojem se javlja poziv metode.
- Ukoliko je član metoda s fiksnim brojem parametara n, tada broj parametara pozvane metode mora biti n i i-ti parametar mora biti potencijalno kompatibilan s odgovarajućim parametrom poziva.
- Ukoliko je član metoda s varijabilnim brojem argumenata koji sadrži n parametara, tada i-ti argument poziva metoda mora biti potencijalno kompatibilan s tipom i-tog parametra metode. Na mjestu gdje n-ti parametar metode ima $T[]$, mora vrijediti:

Izraz poziva metode

- Broj parametara pozvane metode je $n - 1$.
- Broj parametara pozvane metode je n i n-ti argument poziva metode je potencijalno kompatibilan s T ili $T[]$.
- Broj parametara poziva metode je m $(m > n)$ i za sve $n \leq i \leq m$, i-ti parametar poziva metode je potencijalno kompatibilan s T.

- Ukoliko poziv metode uključuje eksplicitne argumente tipa i član je generička metoda, tada je broj argumenata tipa jednak broju parametara tipa metode.

Ukoliko pretraga ne pronađe niti jednu primjenjivu metodu, dolazi do greške pri prevođenju.

Kompatibilnost izraza s ciljnim tipom:

- Lambda izraz je potencijalno kompatibilan s tipom funkcijskog sučelja T ako vrijedi sve od navedenog:

- broj argumenata funkcionalnog tipa T je jednak broju parametara lambda izraza.

Izraz poziva metode

- Ukoliko funkcionalni tip T ne vraća vrijednost, tada je tijelo lambda izraza ili izraz naredbe (npr. uključen u inkrement/dekrement, pridruživanje) ili blok kompatibilan s tipom `void`.
- Ukoliko funkcionalni tip T ima non-void povratni tip, tada je tijelo lambda izraza izraz ili blok kompatibilan s povratnom vrijednosti.

- Izraz reference metode je potencijalno kompatibilan s funkcijskim sučeljem tipa T ako za funkcijski tip T koji prima n argumenata, postoji barem jedna potencijalno primijenjiva metoda (kada izraz reference metode ukazuje na funkcijski tip s brojem argumenata n) i vrijedi jedno od:

- Izraz reference metode ima oblik `ReferenciraniTip::[ArgumentiTipa] Identifikator` i barem jedna potencijalno primijenjiva metoda je ili `static` i podržava n parametara ili nije static i podržava $n - 1$ parametar.
- Izraz reference metode ima neki drugi oblik i barem jedna potencijalno primjenjiva metoda nije `static`.

Izraz poziva metode

- Lambda izraz ili izraz reference metode je potencijalno kompatibilan s varijablom tipa ako je varijabla tipa parametar tipa metode kandidata.
- Izrazi u zagradi su potencijalno kompatibilni s tipom ako je izraz koji sadrže potencijalno kompatibilan s tim tipom.
- Uvjetni izraz je potencijalno kompatibilan s tipom ako je svaki od njegovih drugih i trećih izraza operanada kompatibilan s tim tipom.
- Switch izraz je potencijalno kompatibilan s tipom ako su svi njegovi izrazi rezultata potencijalno kompatibilni s tim tipom.
- Izraz kreiranja instance klase, izraz poziva metode ili samostojeći izraz je potencijalno kompatibilan s proizvoljnim tipom.

Uz broj argumenata, kod primijenjivosti se ispituju i prisutnost i oblik ciljnih tipova funkcijskih sučelja (što potencijalno uključuje rekonstruiranje tipa argumenta).

Izraz poziva metode

Izabir najspecifičnije metode:
Ukoliko više metoda članica zadovoljava svojstvo dostupnosti i primijenjivosti za poziv neke metode, tada *Java* izabire **najspecifičniju** metodu. Intuitivno, jedna metoda je specifičnija od druge ukoliko bi se svaki poziv proslijeđen prvoj metodi, mogao proslijediti drugoj metodi bez pogrešaka pri prevođenju. U nekim situacijama je dozvoljena određena adaptacija da bi se uskladili prototipi pozvane funkcije i potencijalnog kandidata.

3. korak pri prevođenju - je li izabrana metoda prikladna?
- Ukoliko postoji najspecifičnija deklaracija za poziv metode, zove se **deklaracija pri prevođenju za poziv metode**.
Ukoliko argument kod poziva metode **nije kompatibilan s ciljnim tipom** (deklaracije pri prevođenju tipa poziva) dolazi do greške pri prevođenju.
Ukoliko je deklaracija pri prevođenju primjenjiva po broju argumenata, tada u slučaju da je zadnji parametar kod poziva metode $F_n[]$, dolazi do greške pri prevođenju ukoliko **fiksirani tip** F_n **nije dostupan na mjestu poziva**.

Izraz poziva metode

- Ukoliko je deklaracija pri prevođenju `void`, tada poziv metode mora biti **izraz koji je naredba ili izraz u inicijalizaciji ili u inkrement dijelu for petlje**, inače dolazi do greške pri prevođenju. Takav poziv **ne vraća vrijednost**, stoga se mora koristiti u situacijama u kojima vrijednost nije potrebna.

Prihvatljivost deklaracije pri prevođenju može ovisiti i o **obliku izraza poziva metode prije zagrada**:
- ukoliko je izraz oblika `ImeMetode`, samo identifikator i deklaracija pri prevođenju je **metoda instance**, tada je **greška pri prevođenju ukoliko se poziv javlja u statičkom kontekstu**. Inače, ako je C **najbliža obuhvačajuća klasa** klase čiji je **član deklaracija pri prevođenju**. Ukoliko poziv metode nije **direktno obuhvaćen klasom** C ili **unutarnjom klasom** od C, dolazi do greške pri prevođenju.
- Ukoliko je izraz oblika `ImeIzraza.[ArgumentiTipa] Identifikator`, tada deklaracija pri prevođenju **mora biti** `static` ili dolazi do greške pri prevođenju.

Izraz poziva metode

- Ukoliko je izraz `super.[ArgumentiTipa]Identifikator`, tada je **greška pri prevođenju ako je deklaracija pri prevođenju apstraktna ili ako se poziv metode odvije u statičkom kontekstu**.
- Ukoliko je izraz `ImeTipa.super.[ArgumentiTipa] Identifikator`, tada je greška pri prevođenju ako je deklaracija pri prevođenju apstraktna ili ako se poziv metode odvije u statičkom kontekstu. Ukoliko `ImeTipa` označava klasu C, ukoliko **poziv metode nije direktno obuhvaćen klasom** C ili **njezinom unutarnjom klasom**, **dolazi do greške pri prevođenju**. Ukoliko `ImeTipa` označava sučelje, neka je T deklaracija tipa koji neposredno obuhvaća poziv metode. Ukoliko **postoji metoda različita od deklaracije pri prevođenju koja nadjačava deklaraciju pri prevođenju iz direktne nadklase ili nadsučelja** od T, dolazi do greške pri prevođenju.

Izraz poziva metode

Tipovi parametara i rezultata pri prevođenju:
- Ukoliko deklaracija pri prevođenju poziva metode nije polimorfna u prototipu (dopušta korištenje ulaznih varijabli šireg raspona tipova) tada:

- Tipovi parametara pri prevođenju su tipovi formalnih parametara deklaracije pri prevođenju.
- Rezultat pri prevođenju je rezultat tipa poziva deklaracije pri prevođenju.

- Ukoliko je deklaracija pri prevođenju poziva metode polimorfna u prototipu tada:

- Tipovi parametara pri prevođenju su tipovi izraza stvarnih argumenata. Izraz argumenta koji je `null` literal se tretira kao da ima tip `Void`.
- Rezultat pri prevođenju se određuje kao:
  - Ukoliko je polimorfna metoda void ili ima povratni tip različit od `Object`, rezultat pri prevođenju je rezultat tipa poziva deklaracije pri prevođenju.
  - Ukoliko je izraz poziva metode izraz naredbe, rezultat je `void`.

Izraz poziva metode

- Rezultat pri prevođenju se određuje kao:
  - Ukoliko je izraz poziva metode operand izraza pretvorbe, rezultat prevođenja je fiksirani tip izraza pretvorbe.
  - Inače, rezultat pri prevođenju je polimorfan povratni tip `Object`.

Metoda je **polimorfna** ako vrijedi:
- Deklarirana je u klasi `java.lang.invoke.MethodHandle` ili `java.lang.invoke.VarHandle`
- Ima jedan argument čiji deklarirani tip je `Object[]`.
- Sistemska je (*native*).

Informacije pri prevođenju povezane s pozivom metode za korištenje kod izvođenja:
- Ime metode
- Puni tip poziva metode
- Broj parametara i tipovi parametara pri prevođenju (po redu)
- Rezultat pri prevođenju.

Izraz poziva metode

Informacije pri prevođenju povezane s pozivom metode za korištenje kod izvođenja:

- Način poziva koji se računa kao:

  - Ukoliko deklaracija pri prevođenju ima modifikator `static` tada je izvršavanje `static`.
  - Inače, ukoliko je poziv metode prije lijeve zagrade oblike `super.Identifikator` ili oblika `ImeTipa.super.Identifikator`, tada je način poziva `super`.
  - Inače, ukoliko je tip preko kojeg se poziva metoda sučelje, tada je način poziva sučelje.
  - Inače je način poziva virtualan.

Ukoliko rezultat tipa poziva deklaracije pri prevođenju nije `void`, tada se tip izraza poziva metode dobiva primjenom konverzija na povratni tip tipa poziva deklaracije pri prevođenju.

Izraz poziva metode

Evaluacija poziva metoda tijekom izvršavanja:
Provodi se u **pet koraka**. Prvo se računa **referenca cilja** (ukoliko treba), nakon toga se **izvrijedne izrazi argumenata**. Slijedi **provjera pristupačnosti metode** koja se treba pozvati. **Pronalazi se kod metode koju treba izvršiti** i konačno se **kreira novi okvir, provodi se sinkronizacija i kontrola se prenosi kodu metode**.

- **Traženje reference cilja** se izvršava jako slično 1. koraku pri prevođenju (određivanje klase ili sučelja za pretraživanje).
- **Izvrednjivanje argumenata** - prvo se parametri svedu na odgovarajući oblik (kod metoda s varijabilnim argumentima zadnji argument mora uvijek biti tipa T[]). Nakon toga se vrši izvrednjavanje izraza argumenata s lijeva na desno.
- Provjera dostupnosti se vrši prema pravilima dopuštenja pristupa klase, paketa i modula.

Izraz poziva metode

- Lociranje metode za poziv **ovisi o tipu poziva**. Kod **statičkog poziva nije potrebna referenca metode**, **poziva se metoda klase ili sučelja**. Ukoliko je ciljna referenca `null` prijavljuje se `NullPointerException`. Inače, se ciljna referenca koristi kao `this`. Ukoliko je način poziva `super` tada **nije dozvoljeno predefiniranje**. Kod virtualnog poziva dolazi do **dinamičkog traženja odgovarajuće metode** (klasa -> nadklase -> nadsučelja).
- **Stvaranje okvira, sinkronizacija i prijenost kontrole:**
  - Stvara se novi okvir koji sadrži ciljnu referencu (ako postoji) i vrijednosti argumenata (ukoliko postoje). Novi okvir mora imati dovoljno memorije za lokalne varijable, stog, programski brojač, reference na prijašnje okvire i slično. U suprotnom se prijavljuje `StackOverflowError`.
  - Novo stvoreni okvir postaje trenutni (vrijednosti argumenata se pridjeljuju varijablama parametara, ciljna referenca se sprema u `this`, vrši se konverzija ulaznih parametara).
  - Vrši se provjera prototipa metode s tipom deklaracije pri prevođenju.

Izraz poziva metode

- Ukoliko se radi o sistemskoj metodi kod koje nije učitan binarni kod (ovisan o implementaciji) prijavljuje se `UnsatisfiedLinkError`.
- Ukoliko metoda nije sinkronizirana, kontrola se prebacuje u tijelo metode koja se poziva.
- Ukoliko je metoda sinkronizirana, tada se objekt zaključa prije prijenosa kontrole. Daljnje izvršavanje mora čekati dok trenutna dretva ne dobije pristup lokotu.
- Ukoliko postoji ciljna referenca, ciljni objekt mora biti zaključan, inače se zaključava objekt klase koji sadrži metodu.
- Kontrola se prebacuje u tijelo metode koja se poziva.
- Objekt se automatski otključava kada završi izvođenje tijela metode (bez obzira je li izvođenje završilo normalno ili neočekivano). Ponašanje je kao da je metoda unutar naredbe `synchronized`.

Izraz reference metode

Pokazuje na poziv metode bez izvršavanja. Omogućuje stvaranje klasa i polja kao da su poziv metode.

```
1  String::length                    // metoda instance
2  System::currentTimeMillis         // static metoda
3  List<String>::size  // eksplicitni argument tipa za
       genericku metodu i dohvacanje reference funkcije size
4  List::size    // argumenti tipa se rekonstruiraju
5  int[]::clone
6  T::tvarClan
7  System.out::println
8  "abc"::length
9  foo[x]::bar
10 (test ? list.replaceAll(String::trim) : list) :: iterator
11 super::toString
12 ArrayList<String>::new
13 int[]::new
14 Vanjska.Unutarnja::new
```

Primjeri reference metode.