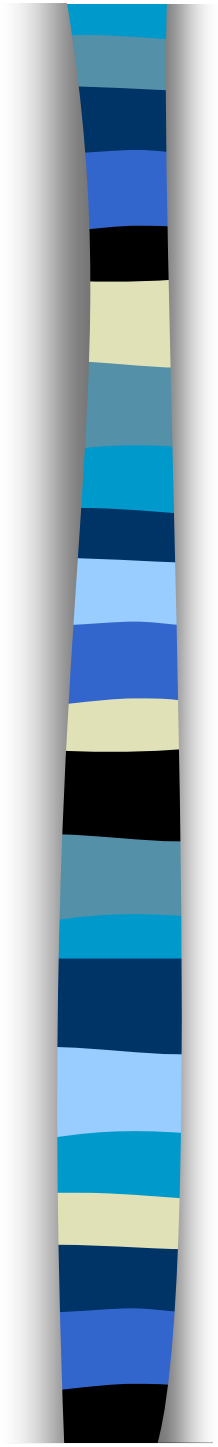


Izuzeci

Izuzeci

- `try` naredba
- `throw` naredba
- Pravljenje novih klasa izuzetaka
- Deklarisanje izuzetaka u zaglavlju metoda
- Štampanje podataka o izuzetku
- `try-with-resources`

Izuzeci

- Izuzeci se koriste da bi se u Java programu kontrolisale neželjene situacije (npr. celobrojno deljenje nulom, iznenadni prekid mrežne komunikacije, pristup elementu niza sa nepostojećim indeksom...)

- Izuzetak je „signal“ koji Java virtuelna mašina generiše ukoliko detektuje grešku prilikom izvršavanja programa.

```
int[] niz = { 4, 6, 1, 2 };
int indeks = /* ucitavanje indeksa od korisnika */;
System.out.println(niz[indeks]);
```

- U primeru ne vršimo kontrolu ulaznih podataka, te korisnik može uneti vrednost koja je van granica niza (npr. -1, ili 5).

- U nekim drugim programskim jezicima, ovo bi bila ozbiljna greška koja bi mogla dovesti do nepredviđenog ponašanja programa. Na sreću, Java virtuelna mašina će prilikom pristupa nizu proveriti da li je vrednost promenljive korektna i, ako nije, generisaće izuzetak.

Izuzeci

- Generisani izuzetak možemo „uhvatiti“ koristeći try-catch naredbu.
- Ukoliko se u bloku kôda koji je ograničen ovom naredbom generiše izuzetak, kontrola toka izvršavanja se prebacuje na odgovarajuću catch granu.
- Ova tehnika nam omogućuje da na adekvatan način obradimo izuzetak, i obezbedimo dalje nesmetano izvršavanje programa.

Izuzeci

- Ako je mesto u programu gde se izuzetak generisao ograničeno try-catch naredbom, izuzetak može biti „uhvaćen“ i tada se kontrola toka programa prebacuje na odgovarajuću catch granu

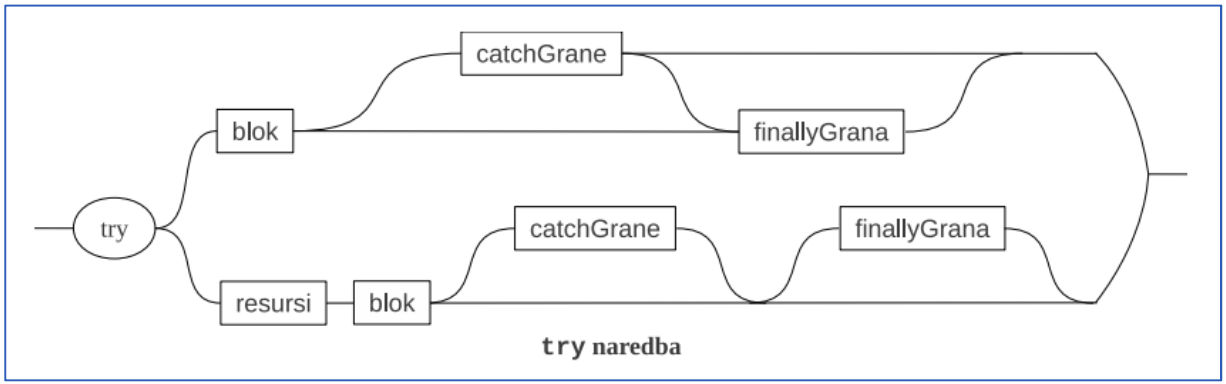
Generisanje i hvatanje izuzetka

```
try{
  System.out.println("dolazi kriticni deo");
  y = a / b;
  System.out.println("sve je dobro proslo");
}
catch (ArithmeticException e){
  System.out.println("deljenje nulom !!!");
}
```

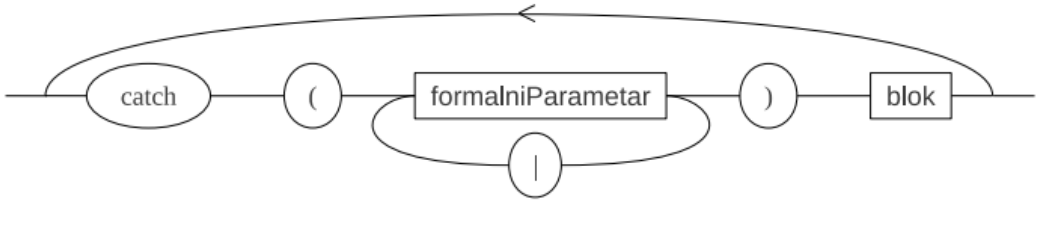
- Ipak ni sa izuzecima ne treba preterivati
- Korišćenje izuzetaka samo da bi se u programu realizovala naredba skoka se ne preporučuje i smatra se lošim stilom programiranja

`try` naredba

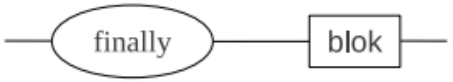
- **try** naredba služi za “kontrolisanje” kritičnih delova koda u toku čijeg izvršavanja može da se generiše izuzetak
- Ako se izuzetak generiše, kontrola toka se prebacuje na odgovarajuću **catch** granu. Ako se izuzetak ne uhvati ni u jednoj **catch** grani, tekuća nit izvršavanja se završava
- Blok koda u **finally** grani će se uvek izvršiti i to neposredno pre nego što kontrola toka izvršenja programa napusti **try** naredbu



try naredba



catchGrane



finallyGrana

`try` naredba

Da bismo znali koji tip izuzetka se desio, navodimo više **catch** grana od kojih svaka odgovara jednom tipu izuzetka koji je mogao da se generiše u navedenom bloku koda.

U svakoj **catch** grani se navodi blok koda koji treba izvršiti ako se desi izuzetak tog tipa. **try** naredba može imati i **finally** granu, koja ako se navede mora doći na kraju **try** naredbe.

Blok koda u **finally** grani će se uvek izvršiti i to neposredno pre nego što kontrola toka izvršenja programa napusti **try** naredbu. Blok koda u **finally** grani će se izvršiti bez obzira na to:

- da li je izuzetak uopšte generisan,
- da li je generisani izuzetak uhvaćen u jednoj od **catch** grana ili ne,
- da li je kontrola toka programa prebačena van **try** naredbe naredbom **break**, **return** ili **throw**.

Ako se navede **finally** grana, **try** naredba ne mora imati ni jednu **catch** granu.

Inače, mora imati bar jednu **catch** granu.

Ovo pravilo ne važi samo u slučaju try-with-resources konstrukcije, o čemu će više reči biti u nastavku.

`try` naredba

- Mehanizam generisanja i hvatanja izuzetaka se često koristi kada se želi izbeći kontrola ulaznih podataka
- Tada se ulazni podaci koriste bez kontrole, koja uvek kvari čitljivost koda, a ceo segment se uokviri u jednu try-catch naredbu
- Ako ulazni podaci nisu dobro zadati, doći će do greške i generisaće se izuzetak koji će biti uhvaćen i preduzeće se odgovarajuća akcija (npr. ponoviće se unos)

Primer - Kontrola ulaznih podataka

```
BufferedReader ulaz = new BufferedReader(new
InputStreamReader(System.in));
try {
  System.out.println("Unesite dva cela broja.");
  int x = Integer.parseInt( ulaz.readLine() );
  int y = Integer.parseInt( ulaz.readLine() );
  int z = x + y;
} catch(NumberFormatException e) {
  System.out.println("Niste dobro uneli brojeve.");
} catch(IOException e) {
  System.out.println("Nesto ne valja sa ulazom.");
}
```

`try` naredba

- Mehanizam generisanja i hvatanja izuzetaka se često koristi kada se želi izbeći kontrola ulaznih podataka
- Tada se ulazni podaci koriste bez kontrole, koja uvek kvari čitljivost koda, a ceo segment se uokviri u jednu try-catch naredbu
- Ako ulazni podaci nisu dobro zadati, doći će do greške i generisaće se izuzetak koji će biti uhvaćen i preduzeće se odgovarajuća akcija (npr. ponoviće se unos)

Primer - Kontrola ulaznih podataka

```
BufferedReader ulaz = new BufferedReader(new
InputStreamReader(System.in));
try {
  System.out.println("Unesite dva cela broja.");
  int x = Integer.parseInt( ulaz.readLine() );
  int y = Integer.parseInt( ulaz.readLine() );
  int z = x + y;
} catch(NumberFormatException e) {
  System.out.println("Niste dobro uneli brojeve.");
} catch(IOException e) {
  System.out.println("Nesto ne valja sa ulazom.");
}
```

Tipovi izuzetaka i greške

- Generisani izuzetak je objekat klase **Exception** ili neke od njenih podklasa, što zavisi od tipa izuzetka.
- Programiranje uz pomoć izuzetaka čini izvršavanje Java programa robustnijim a njegov izvorni tekst čitljivijim.
- Svi izuzeci u Javi su izvedeni iz klase **Exception** ili neke od njenih podklasa. U zavisnosti od konkretnog tipa greške, generiše se izuzetak odgovarajuće klase.
- Tako, da bismo znali koja je tačno greška u pitanju, navodimo više catch grana - po jednu za svaki tip izuzetka koji može biti generisan.
- U okviru svake catch grane navodimo kôd koji treba izvršiti ako se desi izuzetak posmatranog tipa .

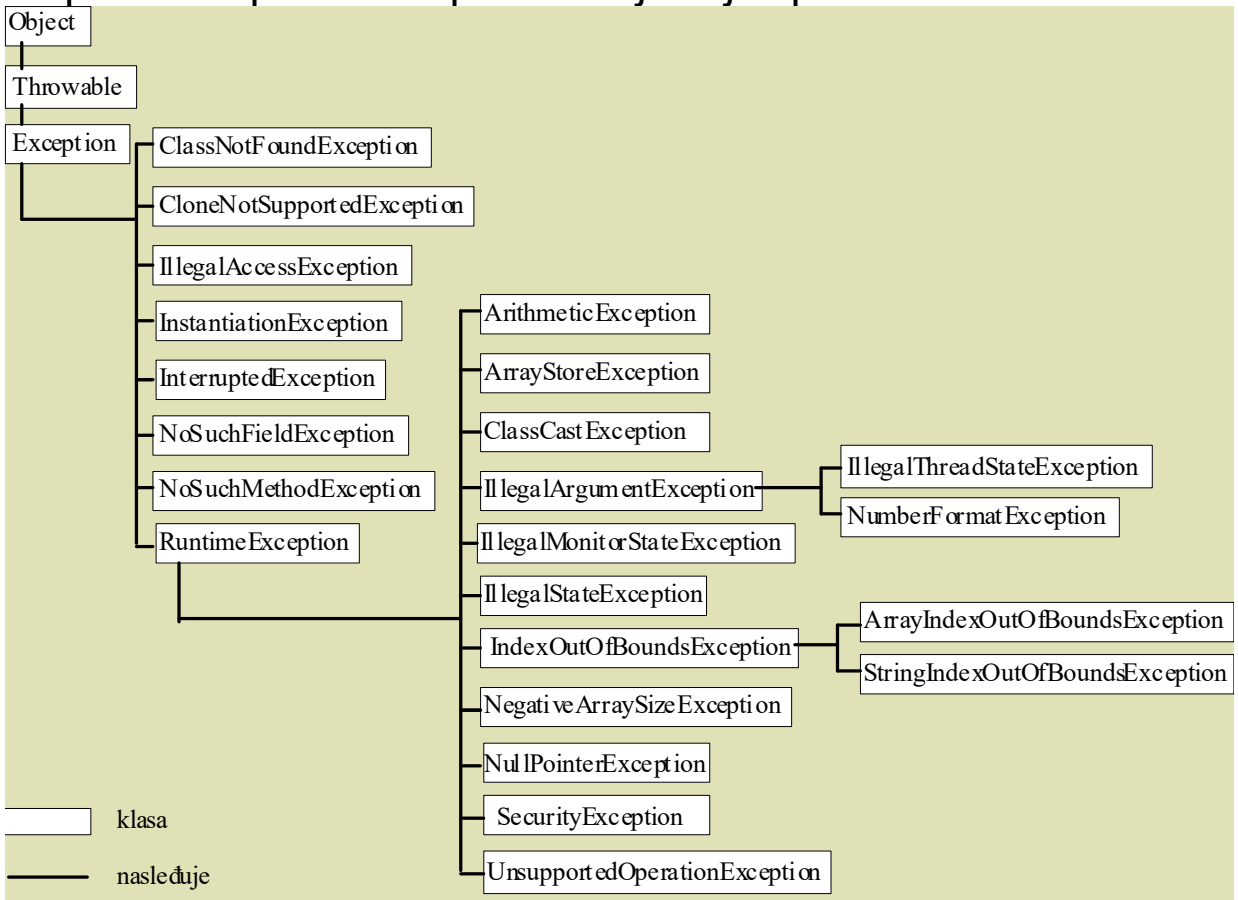
`try` naredba

Primer – `try` naredba sa više `catch` i `finally` granom

```
try {
  DataOutputStream izlaz =
    new DataOutputStream(new FileOutputStream("izlazna.dat"));
  izlaz.writeInt(niz[5] / niz[6]);
} catch(FileNotFoundException e) {
  System.err.println("Datoteka iz nekog razloga ne moze biti
kreirana!");
} catch(NullPointerException e) {
  System.err.println("Niz 'niz' ne postoji!");
} catch(ArrayIndexOutOfBoundsException e) {
  System.err.println("Suvise veliki indeks elementa u nizu!");
} catch(ArithmeticException e) {
  System.err.println("Celobrojno deljenje nulom!");
} catch(IOException e) {
  System.err.println("U fajl nije moguce vise pisati!");
} catch(Exception e) {
  System.err.println("Desilo se nesto nepredvidjeno!");
} finally {
  System.out.println("Mozda se desio izuzetak a mozda ne.");
}
```

`try` naredba

- Redosled navođenja **catch** grana je veoma bitan. Kada se generiše izuzetak, uhvatiće ga prva **catch** grana čiji je tip izuzetka jednak tipu generisanog izuzetka ili ga tip generisanog izuzetka nasleđuje
- Ako imamo catch blokove s nekoliko tipova izuzetaka u istoj klasnoj hijearhiji, potrebno je blokove postaviti tako da se prvo hvata izuzetak najniže podklase pa redom prema najvišoj superklasi



`try` naredba

- Da bi se programeri ohrabrili da bolje obrađuju izuzetke, počev od Jave 7 više srodnih catch grana može biti grupisano u jednu, tzv. multi-catch granu.
- U multi-catch grani može biti navedeno više tipova izuzetaka, odvojenih simbolom „|“. Korišćenje multi-catch grane nam omogućuje da sačuvamo detaljne informacije o generisanom izuzetku, a, pored toga, dovodi i do konciznijeg i efikasnijeg kôda, jer se izbegava dupliranje naredbi:

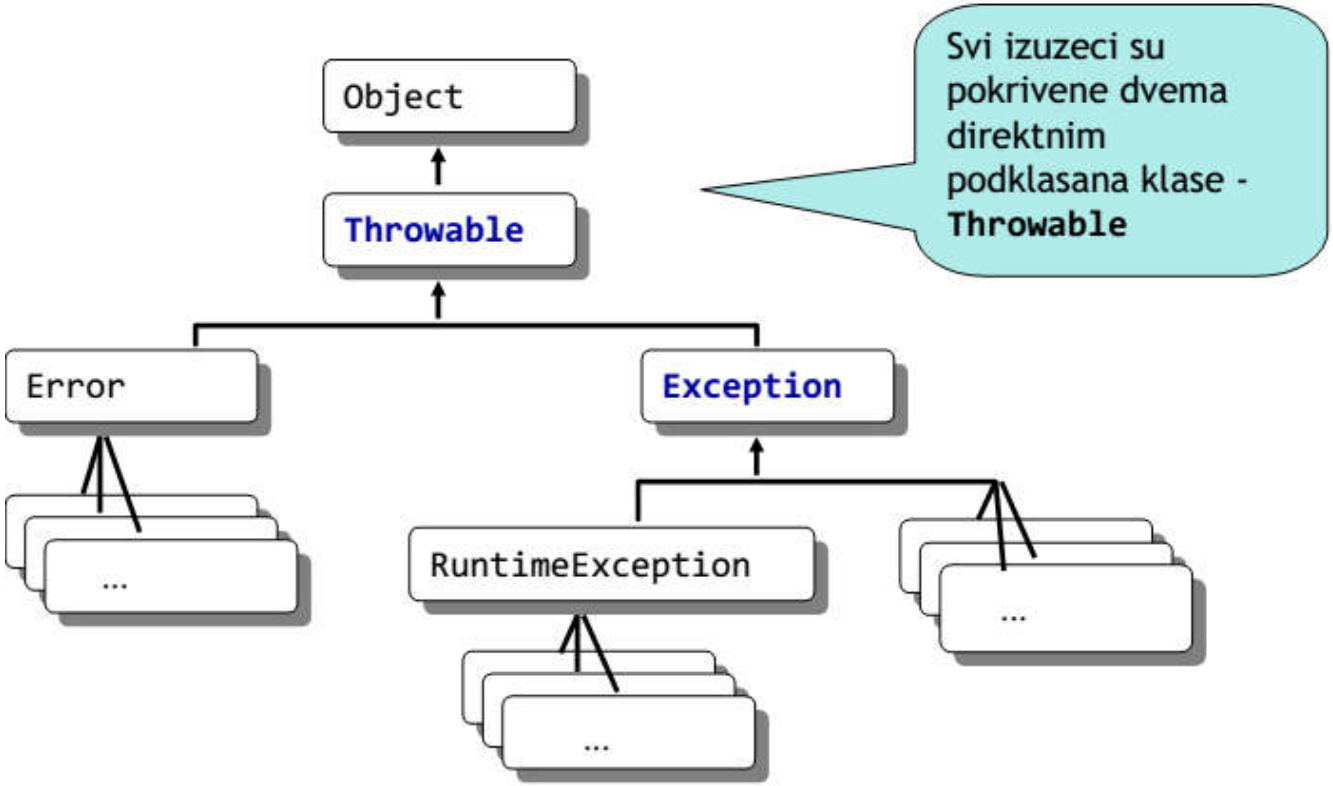
```
// kod koji ne koristi multi-catch granu
try {
   // pokusaj upisivanja podataka u bazu
} catch (SQLException e) {
   System.out.println("Doslo je do greske prilikom upisa.");
} catch (IOException e) {
   System.out.println("Doslo je do greske prilikom upisa.");
}

// koncizniji kod koji koristi multi-catch granu
try {
   // pokusaj upisivanja podataka u bazu
} catch (SQLException | IOException e) {
   System.out.println("Doslo je do greske prilikom upisa.");
}
```

Dva tipa izuzetaka

- Izuzetak koji se **mora** obraditi ili proslediti, “checked” izuzetak (proveravani). Program ne prolazi proces kompajliranja ako checked izuzetak nije
  - obrađen/prosleđen
  - npr. IOException, FileNotFoundException
- Izuzetak koji se ne mora obraditi, “unchecked” izuzetak (neproveravani)
  - Program prolazi proces kompajliranja ako unchecked izuzetak nije obrađen
  - Svaki izuzetak koji direktno ili indirektno nasleđuje **RunTimeException** klasu (RunTimeException extends Exception).
    - NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, ClassCastException, IllegalArgumentException, NumberFormatException,...

Tipovi izuzetaka

- Izuzetak je uvek objekat neke podklase standardne klase ***Throwable***. To važi i za izuzetke koje sami definišemo, kao i za standardne izuzetke
- Dve direktne podklase klase ***Throwable*** – klasa ***Error*** i klasa ***Exception*** – pokrivaju sve standardne izuzetke
- Obe ove klase imaju podklase za specifične izuzetke



RuntimeException izuzeci

- Za skoro sve izuzetke predstavljene podklasama klase ***Exception***, moramo u naš program uključiti kod koji će rukovati njima ukoliko naš kod može izazvati njihovo izbacivanje.

- Ako metod u našem programu može da generiše izuzetak tipa koji ima ***Exception*** kao superklasu, moramo ili rukovati izuzetkom unutar tog metoda ili registrovati da naš metod može izbaciti takav izuzetak.

- Pravilo kojim se zahteva deklaracija neuhvaćenih izuzetaka se ne primenjuje na sve izuzetke.

RuntimeException izuzeci

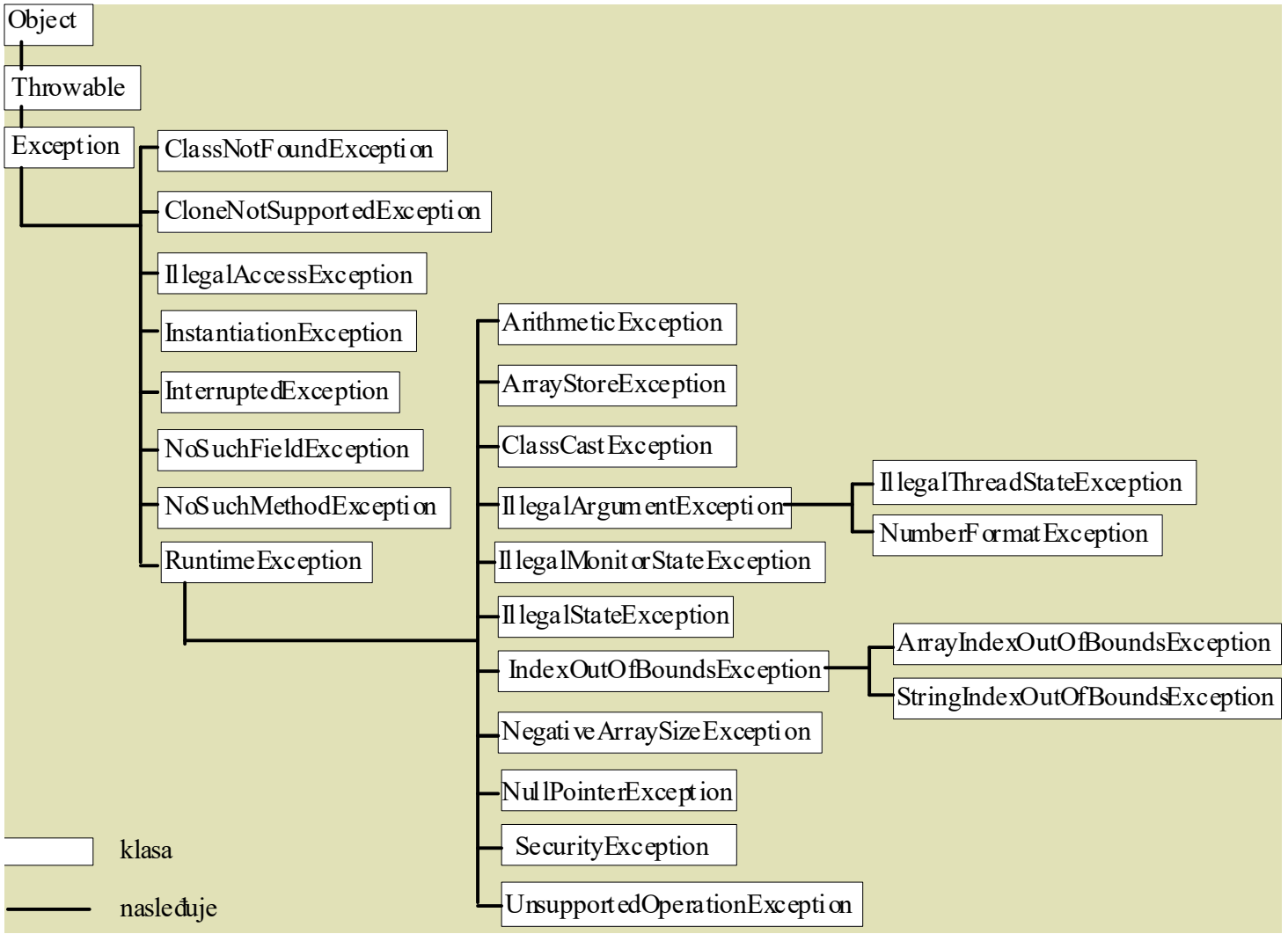
- Naime, izuzeci koji su direktno ili indirektno izvedeni iz klase ***RuntimeException*** ne moraju biti deklarisani u zaglavlju.

- To su izuzeci koji mogu biti generisani na gotovo svakom mestu u programu, kao, na primer, deljenje nulom, pristupanje nepostojećem elementu niza, pristupanje objektu koji ne postoji, i slično.

- Njihovo obavezno deklarisanje bi smanjilo čitljivost programa, a prevodilac bi bio preopterećen proverama mogućnosti pojave i ovih izuzetaka.

- Isto pravilo važi i za greške (klasa Error i njene naslednice).

Klasa Exception



RuntimeException izuzeci

- Podklase ***RuntimeException*** definisane u standardnom paketu *java.lang* su:
  - ArithmeticException (npr. Neispravan rezultat aritmetičke operacije poput deljenja nulom)
  - IndexOutOfBoundsException (npr. Indeks koji je izvan dozvoljenih granica za objekat poput niza, Stringa ili Vector objekat)
  - NegativeArraySizeException (pokušaj definisanja niza negativne dimenzije)
  - NullPointerException (korišćenje promenljive koja sadrži null u slučaju kada ona treba da sadrži referencu na objekat da bi se pozvao metod ili pristupilo atributu)
  - ClassCastException (pokušaj kastovanja objekta neodgovarajućeg tipa)
  - SecurityException (pokušaj narušavanja sigurnosnih pravila)
  - ArrayStoreException, IllegalArgumentException, IllegalMonitorStateException, IllegalStateException, UnsupportedOperationException

Ostale podklase klase Exception

- Za sve ostale klase izvedene iz klase *Exception*, kompajler će proveriti da li smo izvršili rukovanje izuzetkom u metodu gde je izuzetak mogao biti izbačen ili smo naznačili da metod može izbaciti takav izuzetak. Ukoliko nismo ništa od toga, kod se neće iskompajlirati.

- Osim nekoliko koji imaju *RuntimeException* kao osnovu, svi izuzeci izbačeni od metoda Javine biblioteke klasa zahtevaju odgovarajuće rukovanje.

Rukovanje izuzecima

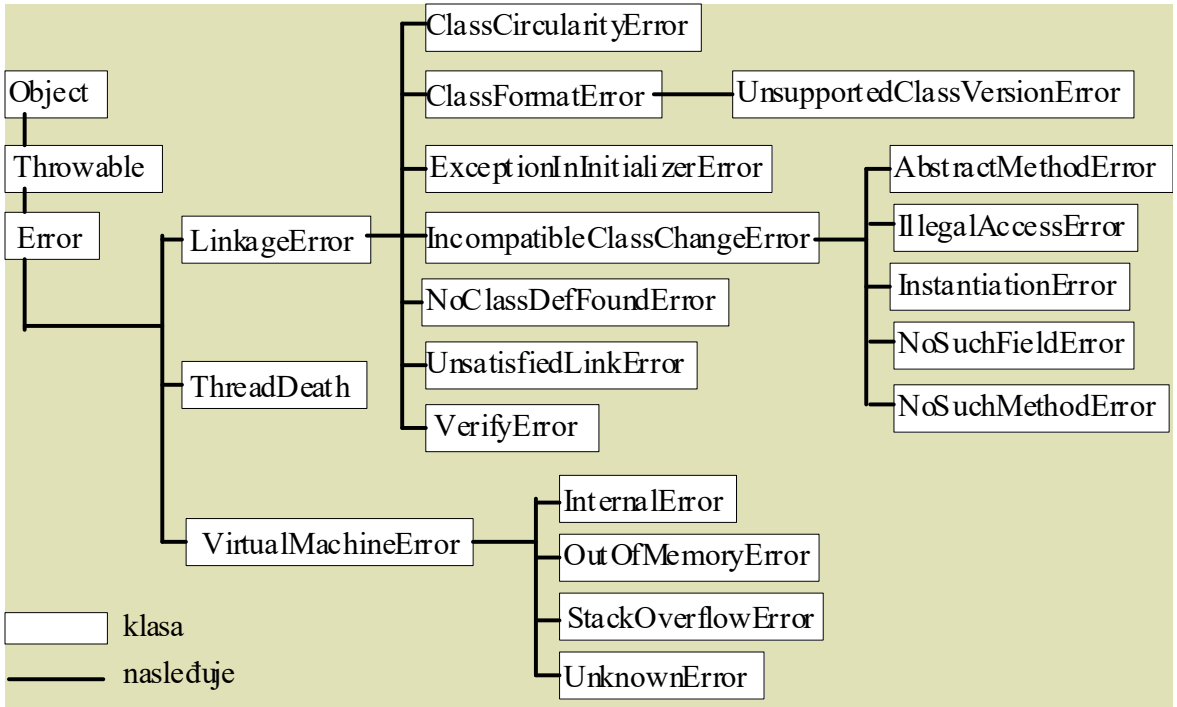
- Dakle, ako naš kod može izbaciti izuzetke koji nisu tipa *Error* ili *RuntimeException* (i njihovih podklasa – što se podrazumeva), moramo nešto da preduzmemo povodom toga.

- Kad god pišemo kod koji može da izbaci izuzetak, imamo izbor.

- Možemo obezbediti kod unutar metoda za rukovanje proizvoljnim izbačenim izuzetkom, ili ga možemo ignorisati omogućujući da metod koji sadrži kod koji može izbaciti izuzetak prosledi taj izuzetak kodu koji je pozvao metod.

Error klasa

- Pored izuzetaka, u toku izvršavanja programa se mogu pojaviti i znatno ozbiljnije greške (npr. greške kod učitavanja klasa, nemogućnost Java virtuelne mašine da pravilno radi zbog nedostatka resursa...)
- Ovakve greške nisu izuzeci, predstavljaju se klasom **Error** i njenim klasama naslednicama, a takođe se mogu obrađivati **try** - **catch** - **finally** naredbom



Error klasa

- Izuzeci definisani klasom *Error* i njenim podklasama karakterišu se činjenicom da se od nas ne očekuje da preduzimamo ništa, ne očekuje se da ih hvatamo.
- *Error* ima 3 direktne podklase:
  - ***ThreadDeath*** – izbacuje se kada se nit koja se izvršava namerno stopira
  - ***LinkageError*** – ozbiljni problemi sa klasama u našem programu, npr. nekompatibilnost među klasama ili pokušaj kreiranja objekta nepostojećeg klasnog tipa su neke od stvari koje mogu izazvati izbacivanje izuzetka ovog tipa
  - ***VirtualMachineError*** – ima 4 podklase izuzetaka koji se izbacuju kada se desi katastrofalni pad JVM

```
public class VMGreske {
    private static void beskonacnaRekurzija() {
        beskonacnaRekurzija();
    }

    public static void main(String[] args) {
        try {
            beskonacnaRekurzija();
        } catch (StackOverflowError e) {
            System.out.println("Prepunjen call stack");
        }

        try {
            long[][][][] hiperKocka =
                new long[Integer.MAX_VALUE][Integer.MAX_VALUE]
                        [Integer.MAX_VALUE][Integer.MAX_VALUE];
        } catch (OutOfMemoryError e) {
            System.out.println("Malo memorije za hiperkocku. Povecaj xmx");
        }

        System.out.println("Nastavljam sa radom... ");
    }
}
```

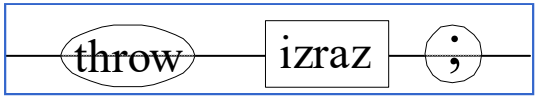
Problems | Javadoc | Declaration | Console | Call Hierarchy | History | Synch

<terminated> VMGreske [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 12, 2013 7:57:38 PM)

```
Prepunjen call stack
Malo memorije za hiperkocku. Povecaj xmx
Nastavljam sa radom...
```

`throw` naredba

- Programer može i sam da generiše izuzetke u svom programu - za to se koristi **throw** naredba
- **throw** naredba generiše (baca) objekat klase **Throwable** ili neke njene podklase
- Iako to sintaksa Jave dozvoljava, u programu nema potrebe za generisanjem ozbiljnih grešaka već samo za generisanjem izuzetaka
- Ovako generisan izuzetak može biti uhvaćen u **catch** grani **try** naredbe. Ako se to ne desi, tekuća nit programa će prestati da se izvršava



Primer – Generisanje izuzetka

```
System.out.print("Unesite rec sa bar dva slova: ");
String str = ulaz.readLine();
if (str.length() < 2)
  throw new Exception();
char prviZnak = str.charAt(0);
char drugiZnak = str.charAt(1);
```

```
public class Matrica {
    private int m, n;
    private double[][] data;

    public Matrica(int m, int n) {..

    public Matrica mnoziSa(Matrica mat) {
        if (this.n != mat.m) {
            throw new IllegalArgumentException(
                "Matrica formata (" + m + "," + n + ")" +
                " se ne moze pomnoziti sa matricom formata" +
                "(" + mat.m + "," + mat.n + ")"
            );
        }

        Matrica p = new Matrica(this.m, mat.n);
        for (int i = 0; i < this.m; i++) {
            for (int j = 0; j < mat.n; j++) {
                double sum = 0;
                for (int k = 0; k < this.n; k++) {
                    sum += this.data[i][k] * mat.data[k][j];
                }
                p.data[i][j] = sum;
            }
        }
        return p;
    }
}
```

```java
public class BrojacLinija {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Koriscenje: BrojacLinija <imeUlaznogFajla>");
            return;
        }

        File f = new File(args[0]);
        if (!f.exists()) {
            System.err.println("Fajl " + args[0] + " ne postoji");
            return;
        }

        int brPokusaja = 0, brLinija = 0;
        boolean uspeo = false;
        do {
            BufferedReader br = null;
            try {
                br = new BufferedReader(new FileReader(f));
                while (br.readLine() != null) brLinija++;
                System.out.println("Broj linija: " + brLinija);
                uspeo = true;
            } catch (IOException ioe) {
                System.err.println("Greska prilikom citanja fajla, pokusavam ponovo... ");
                brPokusaja++;
            } finally {
                try {
                    if (br != null)
                        br.close();
                } catch (IOException e) {
                    System.err.println("Greska prilikom zatvaranja fajla...");
                }
            }
        } while (!uspeo && brPokusaja < 3);
    }
}
```

Pravljenje novih klasa izuzetaka

- Da bi se definisala nova klasa izuzetaka potrebno je napraviti klasu koja nasleđuje klasu **Exception**

Definisanje novog izuzetka

```
class KratakStringIzuzetak extends Exception {

  int duzina;

  KratakStringIzuzetak(int duzina) {
    this.duzina = duzina;
  }

  String poruka() {
    return "Duzina stringa " + duzina + " je suvise mala.";
  }
}
```

Korišćenje novog izuzetka

```
System.out.print("Unesite rec sa bar dva slova: ");
String str = ulaz.readLine();
if (str.length() < 2)
  throw new KratakStringIzuzetak(str.length());
char prviZnak = str.charAt(0);
char drugiZnak = str.charAt(1);
```

Deklarisanje izuzetaka u zaglavlju metoda

- Metod u kojem može doći do generisanja izuzetka koji neće biti uhvaćen u okviru njega samog, mora to imati deklarisano u svom zaglavlju
- Metod koji poziva ovakav metod mora:
  - da uhvati eventualni izuzetak u **try** naredbi, ili
  - da u svom zaglavlju deklariše da njegov poziv može rezultirati izuzetkom koji on neće uhvatiti
- Izuzeci koji su klase **RuntimeException** ili neke od njenih klasa naslednica (npr. pristupanje objektu koji ne postoji, deljene nulom...) ne moraju biti deklarisani u zaglavlju, oni se mogu desiti gotovo na svakom mestu u programu
- Isto važi i za greške (klasa **Error** i njene klase naslednice)

Primer – Metod koji ne hvata mogući izuzetak

```
static int izStringaUBroj(String str) throws
NumberFormatException {
  return Integer.parseInt(str);
  // poziv ovog metoda moze da generise NumberFormatException
}
```

Deklarisanje izuzetaka u zaglavlju metoda

- Zaglavlje metoda mora sadržati listu svih neuhvaćenih izuzetaka - izuzetaka koji mogu biti generisani u metodu, ali koje sam metod ne hvata u catch granama.
- Neuhvaćeni izuzeci se navode nakon ključne reči **throws**:

Primer 9.11: Metod koji ne hvata mogući izuzetak.

```
void obradiString(String str) throws KratakStringIzuzetak {
  if (str.length() < 2)
    throw new KratakStringIzuzetak(str.length());
  // dalja implementacija metoda...
}
```

- Dalje, svaki metod koji poziva obradiString() mora ili uhvatiti navedni izuzetak, ili ga navesti u svom zaglavlju.

Deklarisanje izuzetaka u zaglavlju metoda

- Ukoliko pozivamo metod koji ima više neuhvaćenih izuzetaka, svaki od njih moramo ili uhvatiti ili ponovo navesti u svom zaglavlju:

Primer 9.12: Rad sa više neuhvaćenih izuzetaka.

```
void obradiString(String str) throws KratakStringIzuzetak,
    DugacakStringIzuzetak {
  if (str.length() < 2)
    throw new KratakStringIzuzetak(str.length());
  if (str.length() > 10)
    throw new DugacakStringIzuzetak(str.length());
  // dalja implementacija metoda...
}

void glavni(String str) throws DugacakStringIzuzetak {
  try {
    obradiString(str);
  } catch (KratakStringIzuzetak e} {
    System.out.println(e.getMessage());
  }
}
```

Štampanje podataka o izuzetku

- Kada se desi izuzetak, ponekad je korisno da saznamo više informacija o tome šta se zapravo desilo. U tom slučaju najčešće koristimo dva metoda klase **Throwable**: getMessage() i printStackTrace(). Prvi metod će vratiti detaljan tekstualni opis greške.
- Metod **printStackTrace()** daje tačno mesto u programu gde je izuzetak generisan i koji metodi su tom prilikom izvršavani

Štampanje informacije o izuzetku – fajl `proba.java`

```
class mojaKlasa {
  public static void main(String [] args) {
    try {
      obradi(null);
    } catch(Exception e) {
      e.printStackTrace();
    }
  }

  static void obradi(double [] d) {
    stampaj(d);
  }

  static void stampaj(double [] d) {
    System.out.println(d[0]);
  }
}
```

```
java.lang.NullPointerException:
        at mojaKlasa.stampaj(proba.java:15)
        at mojaKlasa.obradi(proba.java:11)
        at mojaKlasa.main(proba.java:4)
```

Primer: BracketPrettyPrinter

```
if [<= n 1] {1} (* {n} (factorial (- n 1)))
```

```
if
  <= n 1
  1
  *
    n
    factorial
      - n 1
```

```
Neke([]{ } zagrade )])))
Zatvorena ]:20 bez otvorene

Ot (((  nema zatvorene
Otvorena (:5 nema zatvorenu

Zdravko{Zdr(av)ko[dren]]
Otvorena {:7 zatvorena sa ]:23
```

```
public class Zagrada {
    private int pozicija;
    private char z;

    public Zagrada(char z, int pozicija) {
        this.z = z;
        this.pozicija = pozicija;
    }

    public String toString() {
        return z + ":" + pozicija;
    }

    public char getZagrada() {
        return z;
    }
}
```

```
public class NepravilneZagrade extends Exception {
    private Zagrada z;

    public NepravilneZagrade(String poruka, Zagrada z) {
        super(poruka);
        this.z = z;
    }

    public Zagrada getZagrada() { return z; }
}
```

```
public class OtvorenaViseca extends NepravilneZagrade {
    public OtvorenaViseca(Zagrada z) {
        super("Otvorena " + z + " nema zatvorenu", z);
    }
}
public class ZatvorenaViseca extends NepravilneZagrade {
    public ZatvorenaViseca(Zagrada z) {
        super("Zatvorena " + z + " bez otvorene", z);
    }
}
public class ZatvorenaPogresna extends NepravilneZagrade {
    public ZatvorenaPogresna(Zagrada otvorena, Zagrada zatvorena) {
        super("Otvorena " + otvorena + " zatvorena sa " + zatvorena, otvorena);
    }
}
```

```
private static boolean kompatibilna(char z1, char z2) {
    if (z1 == '{') return z2 == '}';
    else if (z1 == '[') return z2 == ']';
    else return z2 == ')';
}

private static StringBuilder uvuci(int level) {
    StringBuilder sb = new StringBuilder();
    sb.append('\n');
    for (int i = 0; i < level; i++) {
        sb.append("  ");
    }
    return sb;
}
```

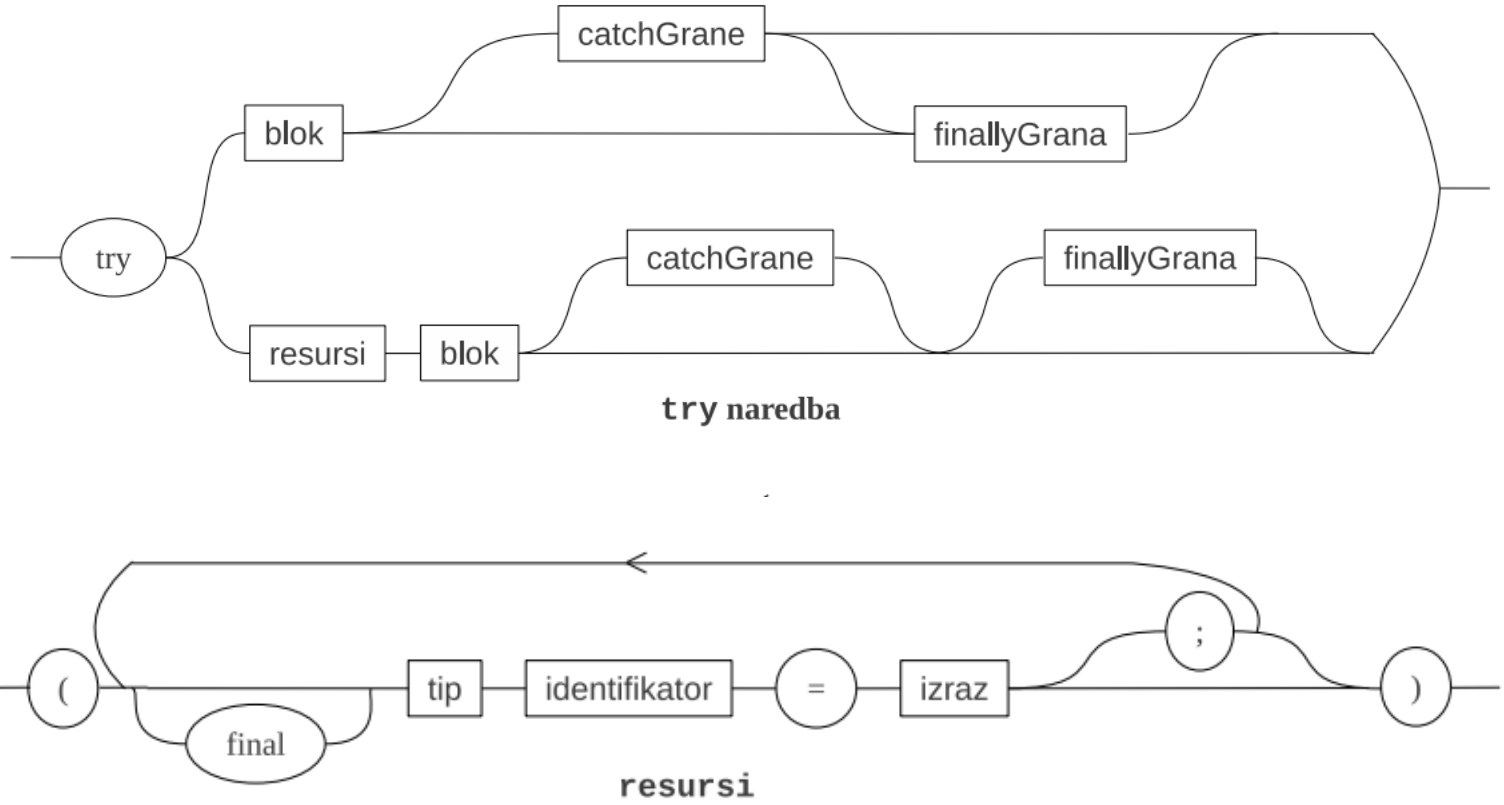
```
public static String format(String ulaz) throws NepravilneZagrade {
    Stack<Zagrada> stek = new Stack<Zagrada>();
    StringBuilder sb = new StringBuilder();
    int level = 0;

    for (int i = 0; i < ulaz.length(); i++) {
        char c = ulaz.charAt(i);
        if (c == '(' || c == '[' || c == '{') {
            stek.push(new Zagrada(c, i));
            ++level;
            sb.append(uvuci(level));
        }
        else
        if (c == ')' || c == ']' || c == '}') {
            if (stek.empty())
                throw new ZatvorenaViseca(new Zagrada(c, i));

            Zagrada poslednjaOtvorena = stek.pop();
            if (!kompatibilna(poslednjaOtvorena.getZagrada(), c)) {
                throw new ZatvorenaPogresna(poslednjaOtvorena, new Zagrada(c, i));
            }

            level--;
        }
        else
            sb.append(c);
    }

    if (!stek.empty()) { throw new OtvorenaViseca(stek.pop()); }
    return sb.toString();
}
```

```
public static void main(String[] args) {
    String[] testSlucajevi = {
        "if [<= n 1] {1} (* {n} (factorial (- n 1)))",
        "Neke([]{ } zagrade )]))))",
        "Ot (((  nema zatvorene",
        "Zdravko{Zdr(av)ko[dren]]"
    };

    int brZatvorenaPogresna = 0, brViseca = 0, ok = 0;
    for (int i = 0; i < testSlucajevi.length; i++) {
        try {
            System.out.println(PrettyPrinter.format(testSlucajevi[i]));
            ok++;
        } catch (NepravilneZagrade e) {
            System.err.println(e.getLocalizedMessage());

            if (e instanceof ZatvorenaPogresna)
                brZatvorenaPogresna++;
            else
                brViseca++;
        }
    }

    System.out.println("Proslo test primera: " + ok);
    System.out.println("Zatvoreno pogresnih: " + brZatvorenaPogresna);
    System.out.println("Visecih            : " + brViseca);
}
```

try-with-resources

- Kao što je rečeno ranije, finally grana se najčešće koristi za oslobađanje resursa na kraju izvršavanja.
- Iako će Javin sakupljač otpadaka osloboditi memoriju odbačenih objekata, program je i dalje dužan da „ručno“ oslobodi određene resurse (npr. da zatvori fajlove).
- Kako bi ovaj postupak bio olakšan, u Javi 7 je uvedena posebna konstrukcija **try-with-resources**.
- Naime, resurs koji se navede kao parametar try naredbe će biti automatski oslobođen na kraju njenog bloka.
- Pri tome se pod resursom podrazumeva objekat klase koja implementira **java.lang.AutoCloseable**, odnosno **java.io.Closeable** interfejs, što, između ostalog, uključuje sve tokove podataka.

try-with-resources



- `try-with-resources` konstrukcija, kao što je prikazano na slici, ne mora imati ni finally ni catch grane.

try-with-resources

- Ukoliko neka operacija unutar ovako zapisanog try bloka ili sam postupak oslobađanja resursa mogu generisati izuzetak, tada važi standardno pravilo:
  izuzetak mora biti obrađen u catch grani ili mora biti deklarisan u zaglavlju metoda.
- Konačno, u okviru `try`- grane može biti navedeno više resursa, odvojenih znakom „ ; “.
- Oblast vidljivosti promenljive koja predstavlja resurs je try blok.

Primer 9.14: Automatsko zatvaranje fajla pomoću `try-with-resources`.

```
void ucitajPodatke() throws IOException {
  try (BufferedReader in =
    new BufferedReader(new FileReader("ulaz.txt")) {
    /* nema potrebe za finally granom: fajl ce na kraju bloka biti
     automatski zatvoren. Posto otvaranje fajla moze generisati
     IOException, izuzetak mozemo ili uhvatiti ovde, ili ga deklarisati
     u zaglavlju metoda */
  }
}
```