Izjeme (`Exceptions`)

Programiranje 2, Tomaž Dobravec

Kaj so izjeme

- Med izvajanjem programa lahko pride do *posebnih okoliščin* (napaka, težava, ...)
  - zaradi resnih posebnih okoliščin (napaka na navideznem stroju, težava s strojno opremo) mora program končati
  - manj problematične so tiste posebne okoliščine, ki jih lahko z Javinimi mehanizmi preprečimo ali zmanjšamo njihov posledice
- Javin mehanizem izjem omogoča avtomatsko upravljanje s posebnimi okoliščinami

Izvor posebnih okoliščin

- Posebne okoliščine so posledica
  - višje sile (na primer: napaka na strojni opremi)
  - nepremišljenega programiranja.

- Izjemne okoliščine so neizbežne, saj
  - na višje sile ne moremo vplivati
  - pri kompleksnih programih je nemogoče predvideti vse možnosti

Primer izjemne okoliščine

- Kaj se zgodi, če poskusimo izvesti naslednji ukaz

  `int i = a/0?`

- Če ukaz izvedemo v jeziku C, se na zaslon izpiše

  `Floating point exception`

  in program se konča

- Če pa ukaz izvedemo v Javi, se izpiše

  `java.lang.ArithmeticException: / by zero`

  in program se konča

NI RAZLIKE?

Primer posebne okoliščine

- Razlika je BISTVENA!
  - V programskem jeziku C na noben način ne moremo zagotoviti, da bi se program po napaki "deljenja z nič" nadaljeval
  - V Javi to lahko storimo!

- Mehanizem: upravljanje z izjemami (exception handling)
  - posebni okoliščini rečemo izjema
  - elegantnejši pristop k reševanju težav
  - trdoživost programov

Izjeme

- Osnovna ideja izjem:
  - kodo, pri kateri lahko pride do težav, izvršimo v posebnem okolju (t.i. poskusni ali `try` blok)
  - če pri izvajanju te kode do težave res pride, se izvrši za ta primer predvidena koda (`catch` blok)

```
try {
   // stavki, ki lahko sprožijo izjemo
} catch(TipIzjeme izjema) {
   // stavki za odziv na izjemo
}
// nadaljevanje programa
```

```
int i; int a = preberiStevilo();
try {
   i =   10/a;
} catch (Exception e) {
   i = 0;
}
System.out.println(i);
```

Primer – branje datoteke brez `try-catch`

```
if (datoteka obstaja) {
  odpri datoteko
  if (uspesno odprl) {
    preberi dolzino datoteke
    if (uspesno prebral dolzino) {
      if (na voljo je zadosti pomnilnika) {
        rezerviraj pomnilnik
        if (uspelo rezervirati pomnilnik) {
          preberi datoteko v pomnilnik
          if (uspelo prebrati datoteko) {
            uspeh!
            zapri datoteko
          } else javi napako pri branju datoteke
        } else javi napaka pri rezerviranju pomnilnika
      } else napaka zaradi premalo pomnilnika
    } else javi napako pri branju dolzine datoteke
  } else javi napako pri odpiranju datoteke
} else javi napako zaradi neobstajanja datoteke
```

Primer – branje datoteke s `try-catch`

```
try{
  odpri datoteko
  preberi njeno dolzino
  rezerviraj pomnilnik
  preberi datoteko v pomnilnik
  uspeh!
  zapri datoteko
} catch (izjema pri odpiranju datoteke) {
      javi napako pri odpiranju
} catch (izjema pri branju dozine datoteke) {
      javi napako pri branju
} catch (izjema pri rezerviranju pomnilnika) {
      javi napako pri rezerviranju pomnilnika
} catch (izjema pri branju v pomnilnik) {
      javi napako pri branju datoteke
}
```

Izjema je objekt

- Izjema v Javi je objekt
- Vse izjeme so potomke razreda `java.lang.Throwable`

Nekatere skupne metode:

- public String **getMessage**()
- public String **toString**()
- public void **printStackTrace**()

```
int i = 3/0;
```

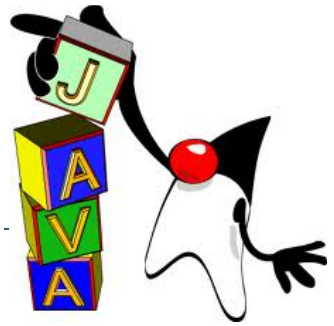
```
/ by zero
```

```
java.lang.ArithmeticException: / by zero
```

```
java.lang.ArithmeticException: / by zero at
    izjeme.Deli.main(Deli.java:7)
```

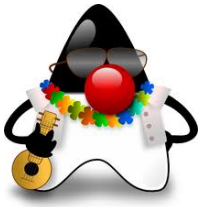
Naloga

Metode izjeme

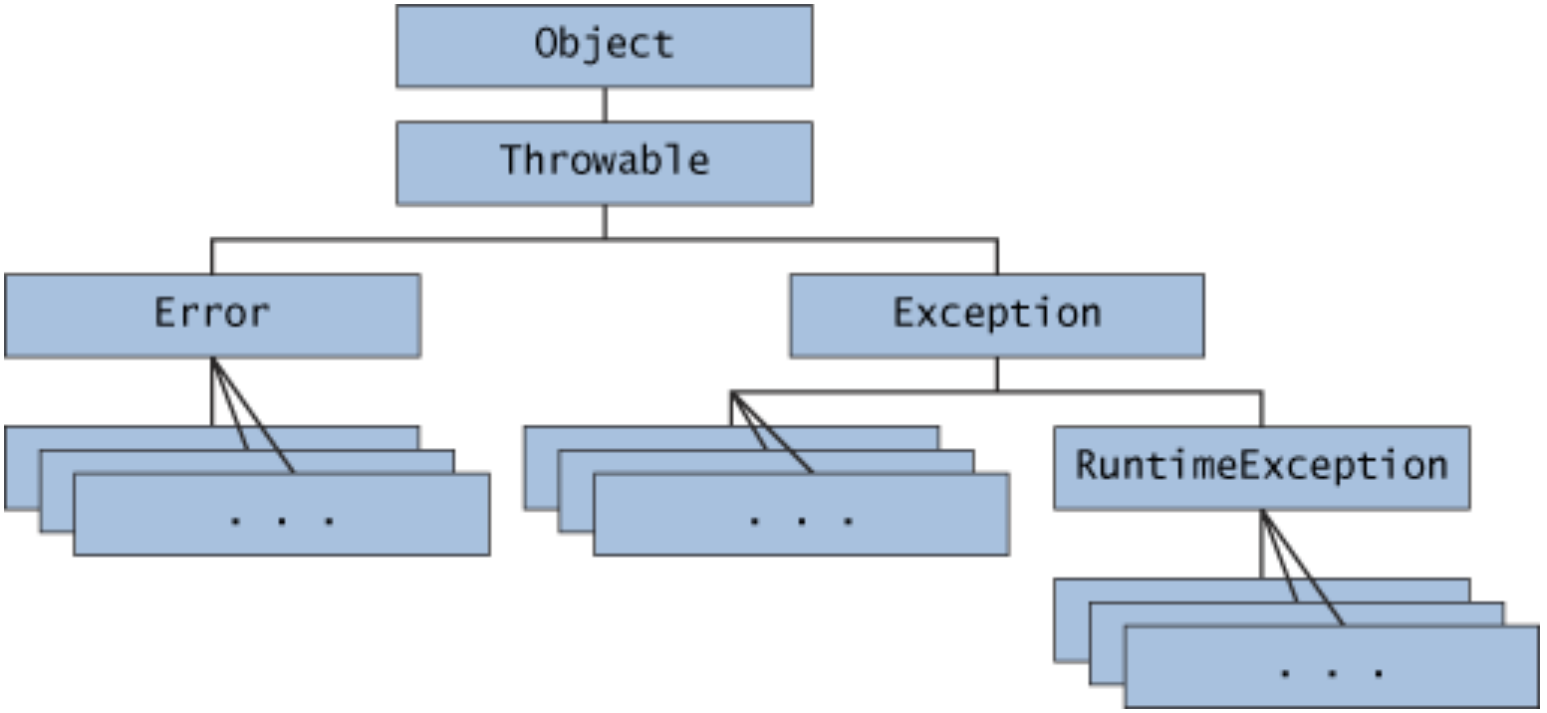


izjeme/Deli.java

V programu v `try` bloku deli z nič in nato v `catch` bloku kliči metode `getMessage`(), `toString`() in `printStackTrace`().



Drevo izjem



- Vse izjeme so potomke enega izmed treh osnovnih razredov:
  - `java.lang.Error`
  - `java.lang.Exception`
  - `java.lang.RuntimeException`

java.lang.Error

- Resne težave, ki se med izvajanjem načeloma ne bi smele pripetiti, potreben je človekov poseg:
  - napaka na pomnilniku,
  - omrežje ne deluje zaradi strojne napake,
  - CPE je okvarjena,
  - napaka virtualnega stroja.

- Napake java.lang.Error ni potrebno loviti (saj načeloma ne moremo nič storiti).

java.lang.Exception

- Manj resne (bolj obvladljive) posebne okoliščine, ker so načeloma popravljive
- Program se lahko na tovrstne izjeme pripravi in se iz njih reši
- Primeri:
  - poskus branja datoteke z diska, datoteka ne obstaja,
  - poskus zapisa podatkov na poln ali na neformatiran disk,
  - poskus branja neveljavnega podatka, ki ga je vnesel uporabnik,
  - negativna vrednost pri starosti osebe,
  - ...

java.lang.RuntimeException

- To so izjeme, ki se pojavijo pri “običajnem” delu :
  - deljenje z nič
  - delo s tabelami (indeks preseže dovoljeno območje)
  - ...
- Čeprav izjem tega tipa NI POTREBNO loviti, to lahko storimo, in s tem omogočimo nemoteno delovanje programa.

Preverljive in nepreverljive izjeme

- **Nepreverljive** so tiste izjeme, ki jih ni treba preverjati (ni potrebno pisati bloka `try/catch`).
- Nepreverljive so vse izjeme razredov `Error` in `RuntimeException` in njunih podrazredov.
- Vse izjeme, ki niso nepreverljive, so **preverljive**.
- Preverljive izjeme moramo OBVEZNO preveriti!

Obravnavanje preverljivih izjem

- Preverljive izjeme MORA nekdo obravnavati.

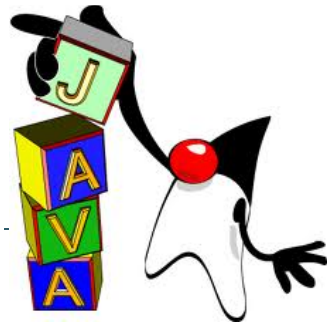
- Dve možnosti:

  – izjemo obravnavamo tam, kjer se je zgodila (`try/catch` blok)

  – pustimo, da izjemo obravnava tisti, ki kliče metodo, v kateri je do izjeme prišlo (metoda, v kateri se lahko pojavi izjema, to napove z rezervirano besedo `throws`).

Naloga

Obravnavanje izjem



izjeme/VrziAliNeVrzi.java

Napiši metodo, ki sproži izjemo. Metodo nato kliči iz dveh različnih metod: v prvi obravnavaj izjemo, v drugi pa izjemo vrži naprej.

Blok `finally`

- Polna struktura `try-catch-finally` bloka:

```
try {
// koda, ki lahko sproži izjemo
} catch ( ...) {
    // koda, ki se izvede, če do izjeme res pride
} finally {
   // koda, ki se izvede v vsakem primeru
}
```

- ukazi v bloku `finally` se izvršijo v vsakem primeru (ne glede na potek dogodkov v `try-catch bloku`).
- v `finally` bloku sprostimo vse sistemske vire, zapremo datoteke in opravimo ostale "čistilne" ukrepe

Blok `finally` – primer

- Kaj izpiše spodnji program?

```
public static void main(String[] args) {
  System.out.println("A");
  try {
    if (args.length != 1) return;
    System.out.println("B");
  } finally {
      System.out.println("C");
  }
  System.out.println("D");
}
```

1 argument
A
B
C
D

brez argumentov
A
C

Nasledniki razreda `Throwable`

- če potrebujemo "svoj" razred za sporočanje izjem, ustvarimo podrazred razreda `Exception` (ali katerega od njegovih podrazredov)

- redefiniramo metodo `getMessage()`

```
class DeljenjeZNic extends ArithmeticException {
  public String getMessage() {
     return "Deljenje z nič ni dovoljeno";
  }
}
```

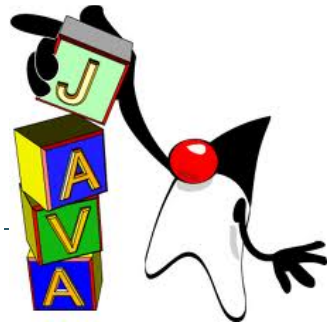
Sprožanje izjem

- V programu izjemo sprožimo z ukazom `throw`

```
public static double deli(int x, int y)  {
    if (y == 0) throw new DeljenjeZNic();
    return x / y;
}
```

Naloga

Izjema v metodi kotangens



izjeme/Cot.java

- Napiši podrazred razreda `ArithmeticException`, ki ga bomo uporabljali za sporočanje izjeme ob napačnem argumentu funkcije tangens (`NapacenArgument`)
- Napiši metodo `Tan(double x)` za računanje funkcije tangens

```
Tan(x) = Sin(x) / Cos(x)
```

Ob napačno podanem argumentu naj metoda vrže izjemo

- Metodo `Tan()` kliči na dva načina: z uporabo try-catch bloka in brez nje.