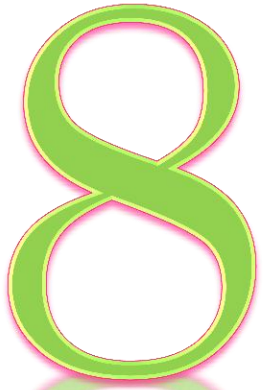

Objektno orijentisano programiranje
vežbe

Staša Vujičić Stanković

Izuzeci

- Mehanizmom izuzetaka vrši se signalizacija ozbiljnih problema pri izvršavanju programa.

- Problemi:
  - greške
  - neki neobični događaji koji zahtevaju posebnu pažnju.

- Pozitivni aspekti upotrebe izuzetaka:
  - Razdvaja se kôd kojim se obrađuju greške od kôda koji se izvršava kada je sve u redu.
  - Obezbeđuju način primoravanja da se reaguje na konkretne greške.

- Ne treba na sve greške u programu reagovati pomoću izuzetaka,
  već samo na neuobičajene ili katastrofalne situacije.
  Razlog je što upotreba izuzetaka zahteva dodatno procesiranje, što može dovesti do usporenja programa.

KLASIFIKACIJA IZUZETAKA

- Mogu se klasifikovati u 4 kategorije:
  - **Greške u kodu i podacima**
    - nekorektno kastovanje nekog objekta
    - upotreba indeksa niza koji je van dozvoljenih granica
    - deljenje nulom u okviru celobrojnog aritmetičkog izraza…
  - **Izuzeci standardnih metoda**
    - metod substring() klase String izbacuje StringIndexOutOfBoundsException…
  - **Izuzeci koje kreira sam korisnik**
  - **Java greške**
    - mogu da se odnose na greške prilikom pokretanja Java Virtualne mašine, ali obično nastaju kao posledica neke greške u programu

Definicija izuzetaka

- Izuzetak u Javi je objekat koji se kreira kada dođe do abnormalne situacije u programu.
- Ovaj objekat sadrži polje koje čuva informaciju o prirodi problema.
- Izuzetak je objekat neke potklase standardne klase **Throwable** – ovo važi i za standardne izuzetke i za izuzetke koje korisnik definiše.
- Potklase Error i Exception klase Throwable pokrivaju sve standardne izuzetke.

- Za izuzetak kažemo da je **bačen**.
- Objekat se prosleđuje kao argument određenom delu programa koji se bavi obradom problema tog tipa.

Izuzeci

- Objekat tipa Throwable
  sadrži dve informacije o izuzetku:

  - poruku o grešci

  - zapis o steku izvršavanja u trenutku kreiranja objekta.

    Stek izvršavanja čuva informacije o svim metodima koji su u stanju izvršavanja u određenom trenutku.

Izuzeci

- Metode:
  - **getMessage()** – vraća sadržaj poruke koja opisuje trenutni izuzetak
  - **printStackTrace()** – ispisuje poruku i odziv steka na standardni izlaz za greške (ekran u slučaju konzolnog programa)
  - **printStackTrace(PrintStream s)** – isto kao prethodni metod, osim što je izlazni tok dat kao argument.

Izuzeci klase Error

- Predstavljaju stanja za koja se od korisnika ne očekuje da išta uradi, tj. izuzeci ovog tipa se ne hvataju.
- Klasa Error ima tri direktne potklase:
  - **ThreadDeath** – nastaje kada se nit koja se izvršava namerno stopira
  - **LinkageError** – ozbiljni problemi sa klasama u programu kao što su nekompatibinost među klasama ili pokušaj kreiranja objekta nepostojećeg klasnog tipa
  - **VirtualMachineError** – pad Java VM.
- U ovakvim situacijama jedino što možemo da uradimo je da pročitamo poruku o grešci koja se generiše kada se izbaci izuzetak.

RuntimeException izuzeci

- Za skoro sve izuzetke koji su predstavljeni nekom od potklasa klase Exception moramo da uključimo kôd u naš program koji će se baviti njima u slučaju da su bačeni.
- Grupa izuzetaka izvedenih iz potklase RuntimeException se drugačije tretira. Naime, kompajler dozvoljava da se ovakvi izuzeci ignorišu, jer se pojavljuju usled ozbiljnih grešaka u kodu. U većini slučajeva malo se može uraditi na oporavku od greške.

RuntimeException izuzeci

- Neke bitnije potklase klase *RuntimeException*:
  - **ArithmeticException** – npr. deljenje nulom
  - **IndexOutOfBoundsException** – indeks je van dozvoljenih granica objekta, koji može biti niz, String ili Vector.
  - **NegativeArraySizeException** – pokušaj definicije niza sa negativnom dimenzijom
  - **NullPointerException** – upotreba promenljive koja sadrži null umesto da referiše na konkretan objekat
  - **ClassCastException** – pokušaj kastovanja objekta u nekorektni tip
  - **IllegalArgumentException** – argument prosleđen metodu nije korektnog tipa

Druge potklase klase Exception

- Za sve ostale potklase klase Exception kompajler će izvršiti proveru
  - da li je izvršena indikacija da metod može da baci tu vrstu izuzetka
  - da li je izvršena obrada izuzetka u metodu u kom može biti izbačen
- Ako nešto od ovoga nije ispunjeno, kôd neće biti iskompajliran.

INDIKACIJA DA METOD MOŽE DA BACI IZUZETAK

- Da bi se naglasilo da neki metod izbacuje izuzetak odgovarajućeg tipa, deklaraciji metoda se dodaje klauzula **throws**.

```
double mojMetod() throws
lista_izuzetaka koje metod izbacuje
{ ... }
```

- Ako u nekom drugom metodu pozivamo ovaj metod, onda on ili mora da izbaci datu listu izuzetaka ili mora da ih obradi.

Obrada izuzetaka

- Obrada izuzetaka se sastoji iz 3 bloka:
  - **try blok** – označava deo koda gde se može javiti izuzetak
  - **catch blok** – obuhvata kôd koji je namenjen obradi izuzetka određenog tipa koji može biti bačen u odgovarajućem try bloku. Argument catch bloka je izuzetak odgovarajućeg tipa i ovaj blok se navodi odmah iza try bloka.
  - **finally blok** – uvek se izvršava pre nego se metod završi, bez obzira da li je izuzetak izbačen u try bloku.

Obrada izuzetaka

- Kada se izbaci izuzetak,
  kontrola se prebacuje na prvu naredbu u catch bloku.
- Try i catch blok su povezani,
  ne smeju se odvajati nekim drugim kodom.
- Na primer, ako petlja sadrži try blok,
  onda i catch blok mora biti deo petlje.
- U slučaju da želimo da se petlja završi ako je ispaljen neki izuzetak,
  onda celu petlju treba staviti u try blok.
- primeri: TestTryCatchPetlja.java i TestTryCatchPetlja1.java

Obrada izuzetaka

- Ako *try* blok može da izbaci više različitih vrsta izuzetaka, možemo staviti za svaki tip izuzetka po jedan *catch* blok za obradu.

```
try
{ ... }
catch(ArithmeticException e) {... }
catch(IndexOutOfBoundsException e) { ... }
// izvršavanje se nastavlja ovde
```

...

- Izuzetak se hvata prvim catch blokom čiji je argument istog tipa kao i izuzetak ili je tipa koji je natklasa klase izuzetka. Samo će se kôd tog catch bloka izvršiti. Nakon toga izvršavanje programa se nastavlja počevši od naredbe iza poslednjeg catch bloka.

Obrada izuzetaka

- Ukoliko je argument catch bloka tipa Exception, takav blok može da hvata bilo koji izuzetak tog tipa ili tipa koji je izveden iz ove klase.
- Ukoliko imamo više catch blokova koji obrađuju izuzetke koji su u hijerarhiji, onda redosled blokova mora biti takav da
  najpre idu blokovi sa najizvedenijim tipom, a na kraju sa najosnovnijim tipom.
  U suprotnom, kôd neće biti preveden.
- Razlog: ukoliko se catch blok sa odgovarajućim tipom izuzetka navede pre catch bloka sa tipom izuzetka koji je izveden iz prethodnog, drugi catch blok nikad neće biti izvršen, što će kompajler da detektuje.

Obrada izuzetaka

- Ako hoćemo da predamo izuzetak koji smo uhvatili u metodu pozivajućem programu, možemo ga izbaciti unutar catch bloka. Na primer:

```
try {
   ...
}  catch(ArithmeticException e) {
   throw e;
}
```

Try blok

- Try blok se ne razlikuje od bilo kog drugog bloka između vitičastih zagrada po pitanju opsega promenljivih.
- Promenljive deklarisane unutar try bloka vidljive su samo u bloku.
- Catch blok ima svoj zaseban opseg promenljivih u odnosu na try blok.
- Ukoliko je potrebno ispisati vrednosti promenljivih ili objekata koje su postavljene u try bloku, treba deklarisati promenljive van try bloka.
- primer: TestTryCatch.java

Finally blok

- Usled pojave izuzetka i prekidanja daljeg izvršavanja try bloka usled prelaska na odgovarajući catch blok, postoji mogućnost da neki delovi ostanu u nekorektnom stanju.
- Na primer, ako nakon otvaranja fajla dođe do pojave izuzetka, neće se izvršiti kod za zatvaranje fajla, što dovodi do nekorektnog stanja.
- Finally blok omogućava "čišćenje" na kraju izvršavanja try bloka.

Finally blok

- Finally blok se uvek izvršava, bez obzira da li je došlo do ispaljivanja izuzetka u odgovarajućem try bloku.
- Ukoliko je potrebno, na primer, zatvoriti fajl, ili osloboditi neke kritične resurse, kôd koji to treba da odradi treba smestiti u finally blok.

Finally blok

- Kao i catch blok, i finally blok je pridružen odgovarajućem try bloku i mora neposredno da sledi iza poslednjeg catch bloka.
  Ako nema catch bloka, finally blok dolazi odmah nakon try bloka.
- U suprotnom, program neće biti preveden.
- Ukoliko finally blok vraća neku vrednost, return naredba finally bloka preklapa bilo koju return naredbu try bloka.

Zadatak fje.java

- Napisati program koji čita podatke sa standardnog ulaza i na osnovu učitane vrednosti računa vrednosti funkcija arccos, arcsin, koren(x), 1/x, log(a,b) ili ispisuje poruku o grešci na standardni izlaz.
  Na primer, ako se unesu vrednosti:

  0.5 0.7 0.8 10 -10 1 2 3 -3 - 4

  tada treba da se ispiše sledeći tekst:

- arccos(0.5) = 1.0472
- arcsin(0.5) = 0.523599
- koren(0.5) = 0.707107
- JedanKrozIks(0.5) = 2
- log(0.5,0.7) = 0.514573
- arccos(0.8) = 0.643501
- arcsin(0.8) = 0.927295
- koren(0.8) = 0.894427
- JedanKrozIks(0.8) = 1.25
- log(0.8,10) = -10.3189
- Greska za x = -10: arccos: argument nije u intervalu [-1 1].
- Greska za x = -10: arcsin: argument nije u intervalu [-1 1].
- Greska za x = -10: koren: potkorena velicina mora biti nenegativna!!!
- JedanKrozIks(-10) = -0.1
- Greska: log: osnova mora biti pozitivna i razlicita od jedan.
- Greska za x = 2: arccos: argument nije u intervalu [-1 1].
- Greska za x = 2: arcsin: argument nije u intervalu [-1 1].
- koren(2) = 1.41421
- JedanKrozIks(2) = 0.5
- log(2,3) = 1.58496
- Greska za x = -3: arccos: argument nije u intervalu [-1 1].
- Greska za x = -3: arcsin: argument nije u intervalu [-1 1].
- Greska za x = -3: koren: potkorena velicina mora biti nenegativna!!!
- JedanKrozIks(-3) = -0.333333
- Greska: Nema dovoljno argumenata u datoteci za log!

- Napisti odgovarajuće metode za obradu izuzetaka:
  - **acos** – izbacuje izuzetak ukoliko argument nije između -1 i 1, inače ispisuje vrednost acos(arg).
  - **asin** – izbacuje izuzetak ukoliko argument nije između -1 i 1, inače ispisuje vrednost acos(arg).
  - **koren** – izbacuje izuzetak ukoliko argument nije veći od 0, inače ispisuje vrednost sqrt(arg).
  - **JedanKrozIks** – izbacuje izuzetak ukoliko argument nije različit od 0, inače ispisuje vrednost 1 / arg.
  - **log** – izbacuje izuzetak ukoliko prvi argument nije različit od 1 i veći ili jednak 0, a drugi argument veći od 0 inače ispisuje vrednost log(arg1, arg2).

- Napisati klasu *MojiIzuzeci* koja nasledjuje klasu *ArithmeticException*, a sadrži:
  - konstruktor bez argumenata i
  - konstruktor sa argumentom koji je tekst odgovarajuće poruke prilikom pojavljivanja izuzetka.