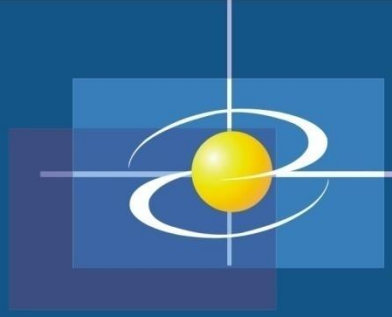


Programski jezik JAVA
PREDAVANJE 4

Prezentacija kreirana na osnovu sljedeće literature :
Dejan Živković: Osnove Java programiranja; Bruce Eckel: Misliti na Javi

Izuzeci (exceptions) i ulaz/izlaz

- Greške i izuzetne situacije mogu se desiti u izvršavanju svake aplikacije. Radi se o prekidu normalnog izvršavanja programa.
- To mogu biti greške u logici programiranja, kao npr:
  - Dijeljenje sa nulom
  - Pristup elementima polja izvan specificiranih granica.
- Ovo su **izuzetne** situacije u programu koje nastaju izvršavanjem izraza koji narušavaju semantiku jezika.
- Postoje i drugi tipovi **izuzetnih situacija**, kao što su:
  - Nedostupnost sistemskog resursa poput Interneta.
  - Nepostojanje lokalne datoteke na disku.
  - Nedovoljne količine memorije.
  - Netačan unos podataka od strane korisnika.
- Aplikacije pisane u raznim programskim jezicima mogu reagovati na ovakve situacije na različite, često nepredvidljive načine.
- Mehanizam za rad u izuzetnim situacijama (**error handling**).

Izuzeci (exceptions) i ulaz/izlaz

- Primjer: Program učitava string i pretvara ga u integer:

```
import java.util.*;
public class izuzeci {
      public static void main(String[] args) {
                Scanner x = new Scanner(System.in);
                 int s = Integer.parseInt(x.nextLine());
                 System.out.println("s= " +s);
      }
   }
```

- Ako unesemo 1 program će ispisati s = 1 na standardni izlaz (output).
- Ako unesemo **a** – imamo izuzetnu situaciju tokom izvršavanja programa. Normalan tok se prekida i umjesto ispisa `System.out.println("s= " +s);` imamo:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:449)
    at java.lang.Integer.parseInt(Integer.java:499)
    at izuzeci.main(izuzeci.java:6)
```

- radno okruženje nije reagovalo nepredvidljivo, već je u liniji koda **`int s=Integer.parseInt(x.nextLine());`** u nemogućnosti konvertovanja stringa u integer prekinulo normalno izvršavanje programa i kontrola je prenijeta na drugo mjesto izvršavnja gdje se ispisala gornja poruka.

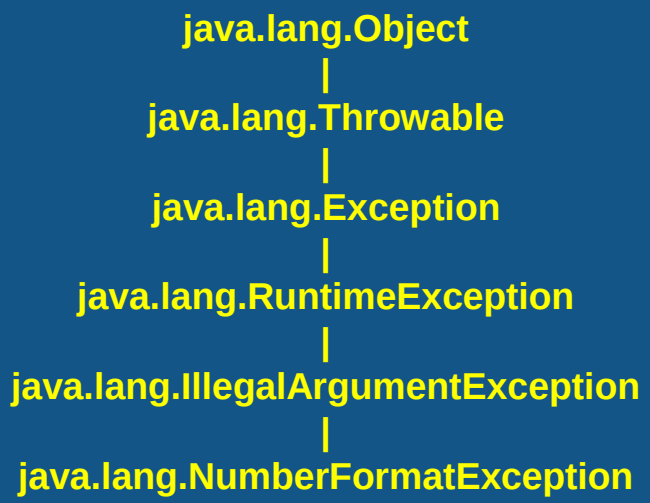
Izuzeci (exceptions) i ulaz/izlaz

- Šta se desilo? Odgovor leži u deklaraciji metode parseInt:

```
public static int parseInt(String s) throws NumberFormatException
```

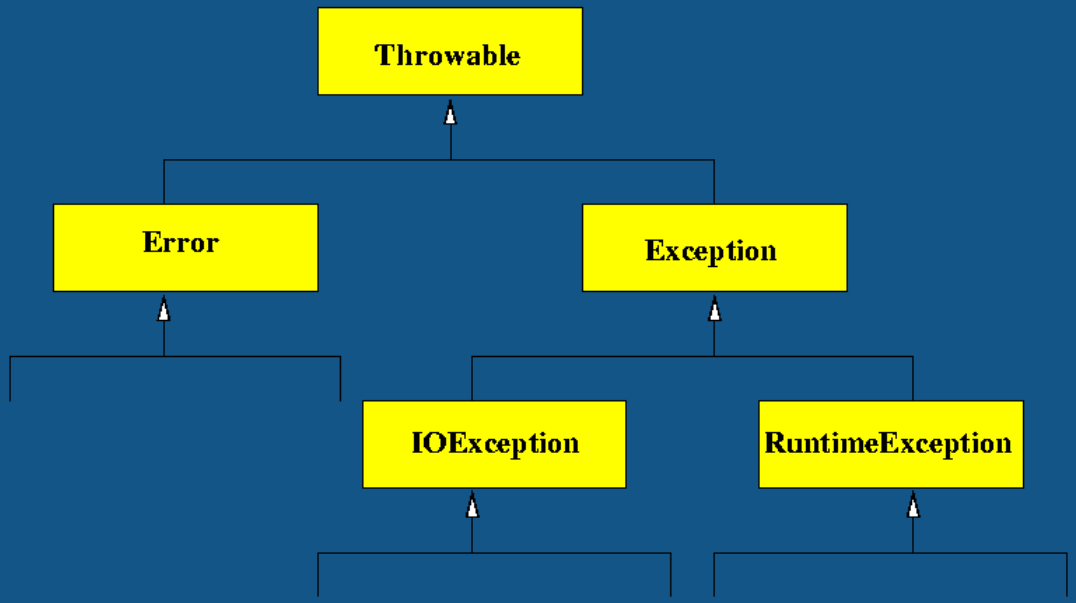
- U ovom slučaju metoda parseInt, kada naiđe na argument koji se ne da konvertovati u integer, uzrokuje da radno okruženje kreira objekt iz klase java.lang.NumberFormatException i preda ga radnom okruženju. Na taj način radno okruženje reaguje na izuzetnu situaciju.
- Da bi vidjeli šta se dalje dešava, najprije pogledajmo koje informacije sadrži takav objekt. Pogledajmo zato hijerahiju klasa u kojoj se nalazi java.lang.NumberFormatException:
- Throwable je natklasa za sve

  greške (errors) i izuzetke (exeptions).



Podjela na greške i izuzetke

- Ovakva kontrola izuzetaka nije zadovoljavajuća, jer smo možda trebali uraditi još nešto prije završetka programa. Prema tome, moramo riješiti sljedeće:
  - **Treba nam način da procesiramo izuzetak**
  - **Da znamo koji izuzetak da procesiramo**
- U slučaju izuzetne situacije (greške) Java izbacuje (*throws*) objekat koji enkapsulira informaciju o grešci koja se javila.
- Svaki takav objekat (izuzetak) pripada nekoj klasi koja proširuje klasu Throwable (`java.lang.Throwable`)



Izuzeci tipa ERROR

- Predstavljaju izuzetke za koje se ne očekuje da ih hvata korisnik
- Tri direktne podklase klase **Error**:
  - ThreadDeath: pojavljujea se kada se namjerno zaustavi nit u izvršavanju. Kada se ne uhvati (catch), završava se nit, a ne program
  - LinkageError: ozbiljne greške unutar klasa u programu (nekompatibilnosti među klasama, pokušaj kreiranja objekta nepostojeće klase)
  - VirtualMachineError –JVM greška

Izuzeci tipa RunTimeException

- Za skoro sve izuzetke koji su obuhvaćeni potklasama klase **Exception** potrebno je uključiti kod za njihovu obradu ili program neće proći kompajliranje
- RunTimeException izuzeci se tretiraju drugačije. Prevodilac dozvoljava njihovo ignorisanje.
- Razlog: ovi izuzeci se najčešće pojavljuju kao posljedica ozbiljnijih grešaka u kodu, čija obrada ne bi mogla ništa značajno promeniti.
- Neke njene podklase:
- ArithmeticException: neispravan rezultat aritmetičke operacije poput dijeljenja sa nulom
- IndexOutOfBoundsException: indeks koji je izvan dozvoljenih granica za objekat poput niza, stringa ili vektora
- NegativeArraySizeException: upotreba negativnog indeksa niza
- NullPointerException: poziv metoda ili pristup podatku članu null objekta
- ArrayStoreException: pokušaj dodeljivana reference pogrešnog tipa elementu niza
- ClassCastException: pokušaj kastovanja objekta neodgovarajućeg tipa
- SecurityException: pokušaj narušavanja sigurnosnih pravila (security manager).

Try-catch-finally

- Java dijeli izuzetke na provjeravane (*checked exceptions*) i neprovjeravane (*unchecked exceptions*):
  - Provjeravani izuzeci su oni za koje prevodilac provjerava da li ih program procesira. To su svi izuzeci u `java.lang.Exception` osim `java.lang.RunTimeException`.
  - Neprovjeravani izuzeci su oni za koje prevodilac ne provjerava da li smo ih procesirali. To su sve podklase od `java.lang.Error` i `java.lang.RunTimeException`.
- Java koristi try-catch-finally šemu za procesiranje izuzetaka.

```
try{
        /*
          u try bloku dolazi ono što pokušavamo izvršiti
          -ako uspijemo idemo na finally blok
          -ako ne na catch blok
        */
.................
 }//kraj try bloka
 catch (NumberFormatException e) {
.................
  }//kraj catch bloka

  finally{
 /*
  ovaj dio nije obavezen
  -prevodioc se ne buni ako ga nema
 } //kraj finally bloka
```

Primjer

```
1  import java.util.*
2  public class NoviA {
3  public static void main(String[] args) {
4      try{
5
6        Scanner x = new Scanner(System.in);
7                int s = Integer.parseInt(x.nextLine());
8          System.out.println("s= " +s);
9
10      }//kraj try bloka
11      catch (NumberFormatException e) {
12
13      System.out.println("Izuzetak ispisan na standardni output:
" + e.getMessage());
14       }//kraj catch bloka
15
16      finally{
17
18        System.out.println("Ovo se uvijek izvrsi!");
19      } //kraj finally bloka
20
21            System.out.println("Program nastavi s
izvrsavanjem.");
22            int s=Integer.parseInt("1234");
23            System.out.println("s= " +s);
24      }
```

Rezultat izvršavanja

- Izvršavanjem java NoviA za unijetu vrijednost aaa imamo ovakav ispis:

```
Izuzetak ispisan na standardni uotput: For input string: "aaa"
Ovo se uvijek izvrsi!
Program nastavi s izvrsavanjem.
s= 1234
```

- Try blok može slijediti više catch blokova.

```
try{
...

  }//kraj try bloka
catch (NumberFormatException e) {
...
}//kraj catch bloka
catch (ArrayIndexOutOfBoundsException e) {
...
}//kraj catch bloka
catch (Exception e) {//uhvatimo sve ostale
...
}//kraj catch bloka
```

try-catch

- Dio koda koji može izbaciti izuzetak koji želimo procesirati smjestimo unutar `try` bloka. Ako izuzetak bude izbačen preskače se preostali dio `try` bloka i prelazi se na izvršavanje `catch` bloka.
- Unutar `catch` bloka procesiramo izuzetak koji se javlja kao argument u `catch` naredbi. Catch blokova može biti više: u svakom procesiramo jedan mogući izuzetak.
- Ako je potrebno dodajemo `finally` blok koji se izvršava u oba slučaja: kad je izuzetak izbačen i kad nije. `finally` će se izvršiti i onda kada izuzetak nije uhvaćen sa `catch`. U taj blok obično stavljamo oslobađanje resursa koje je potrebno izvršiti u svakom slučaju.
- Kod može eksplicitno izbaciti izuzetak pomoću `throw` naredbe.
- Ako neka metoda može izbaciti *provjeravani izuzetak*, a sama ga ne procesira, onda taj izuzetak mora biti deklarisan u samoj metodi.

Primjer..

- Ilustrirajmo ovo poslednje pravilo. Na primjer, konstruktor klase **java.io.FileReader** može izbaciti provjeravani izuzetak **FileNotFoundException**. Treba procesirati u catch bloku kao ovdje:

```
// Verzija s procesiranjem izuzetka

public void method()
{
        try{

            FileReader fr = new FileReader("moj.file");
            .........
        }
        catch(FileNotFoundException e){
             e.printStackTrace();
             System.exit(0);
        }
}
```

Nastavak...

- Ako ne znamo kako procesirati **FileNotFoundException** možemo ga ignorisati, ali ga moramo deklarisati na sljedeći način:

```
// Verzija bez procesiranja izuzetka

public void method() throws FileNotFoundException
{

        FileReader fr = new FileReader("moj.file");
        .........

}
```

- Sada će **FileNotFoundException** biti proslijeđen pozivnoj metodi, sve dok se ne nađe odgovarajući `catch` blok. Ako takvog nema, radno okruženje procesira izuzetak zaustavljanjem programa i ispisom sadržaja steka.