Iznimke u Javi

Sadržaj

Uvod u iznimke

- Iznimke (engl. *exceptions*) predstavljaju problem koji nastaje tijekom izvođenja programa
- Upravljanje iznimkama omogućava razvoj robusnih aplikacija koje mogu nastaviti rad i nakon nastanka problema
- Iznimke su u Javi predstavljene klasama koje izravno ili neizravno nasljeđuju klasu **java.lang.Throwable**
- Iznimke, odnosno objekte klasa koje predstavljaju iznimke moguće je hvatati (engl. *catch*) i bacati (engl. *throw*)
- Moguće je i kreirati vlastite klase koje predstavljaju iznimke nasljeđivanjem određenih klasa koje predstavljaju jednu od skupina iznimaka

Primjer programa bez obrade iznimke

- Neka je zadan sljedeći programski isječak:

```
public class BezObradeIznimke {

   private static int podijeli(int djeljenik, int djelitelj) {
      return djeljenik / djelitelj;
   }

   public static void main(String[] args) {
      Scanner unos = new Scanner(System.in);
      System.out.print("Unesite dva broja: ");
      int prvi = unos.nextInt();
      int drugi = unos.nextInt();
      int rezultat = podijeli(prvi, drugi);
      System.out.println("Rezultat dijeljenja je: " + rezultat);
      unos.close();
   }
}
```

Primjer programa bez obrade iznimke

- Ako se za drugi broj unese vrijednost „0”, dijeljenje nije moguće i događa se pogreška u programu opisana sljedećim opisom:

```
Unesite dva broja: 5 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at dijeljenje.BezObradeIznimke.podijeli(BezObradeIznimke.java:8)
at dijeljenje.BezObradeIznimke.main(BezObradeIznimke.java:18)
```

- Navedeni opis predstavlja stazu stoga (engl. *stack trace*) i uključuje:
  - Naziv iznimke: `java.lang.ArithmeticException`
  - Opisnu poruku uz iznimku: `/ by zero`
  - Lokacija poziva programskog koda s iznimkom:

```
at dijeljenje.BezObradeIznimke.podijeli(BezObradeIznimke.java:8)
```

Primjer programa bez obrade iznimke

- U slučaju upisa tekstualnih vrijednosti u program (umjesto brojčanih), aplikacija javlja sljedeću pogrešku:

```
Unesite dva broja: prvi drugi
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknown Source)
at java.util.Scanner.next(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at dijeljenje.BezObradeIznimke.main(BezObradeIznimke.java:15)
```

- Kako metoda „nextInt” iz objekta klase „Scanner” očekuje unos cjelobrojne vrijednosti, a upisan je tekst, nije moguće tu vrijednost pretvoriti u cjelobrojnu i dojavljuje se pogreška

Završetak rada programa nakon pojavljivanja iznimke

- U slučaju pojavljivanja iznimaka `java.lang.ArithmeticException` i `java.util.InputMismatchException` kao u primjeru, program završava s izvođenjem
- Ponekad aplikacija može nastaviti s radom (kao u slučaju aplikacije s grafičkim sučeljem), ali rezultati mogu biti nepredvidivi
- Svako pojavljivanje iznimke u programu može se i obraditi te time osigurati nastavak rada programa do očekivanog završetka
- Informacije o iznimci često se zapisuju u posebne ***log*** datoteke kako bi se naknadno mogle analizirati

Primjer programa s obradom iznimke

- U slučaju da se prošli primjer programa proširi blokovima za obradu iznimke, program može nastaviti s izvođenjem usprkos pogreškama kod unosa:

```
…
boolean nastaviPetlju = false;
do {
   try {
      System.out.print("Unesite dva broja: ");
      int prvi = unos.nextInt();
      int drugi = unos.nextInt();
      int rezultat = podijeli(prvi, drugi);
      System.out.println("Rezultat dijeljenja je: " + rezultat);
      nastaviPetlju = false;
   }
   catch(InputMismatchException ex1) {
      System.out.println("Morate unijeti brojčane vrijednosti.");
      unos.nextLine();
      nastaviPetlju = true;
   }
   catch(ArithmeticException ex2) {
      System.out.println("Unesite ispravne vrijednosti za dijeljenje.");
      unos.nextLine();
      nastaviPetlju = true;
   }
} while(nastaviPetlju);
…
```

Primjer programa s obradom iznimke

- Obrada iznimaka obavlja se korištenjem blokova ***try*** i ***catch***
- Blok „try” sadržava programski kod u kojem postoji mogućnost bacanja iznimke
- Blok „catch” sadržava programski kod koji se izvodi u slučaju kad se određena iznimka dogodi
- Blok „try” se izvodi sve do trenutka kad se baci iznimka i nakon toga se počne izvoditi programski kod u „catch” bloku (ako se dogodi iznimka)
- Ako se cijeli programski kod stavi u „do-while” petlju, program se izvršava bez obzira na iznimke i traži ponovni unos od korisnika u slučaju pogrešaka
- U slučaju da se dogodi iznimka za koju ne postoji „catch” blok, radi se o „neuhvaćenoj iznimci” koja će prekinuti izvođenje programa

Multi-catch blok

- Od Jave verzije 7 uvedena je mogućnost korištenja i „multi-catch” bloka
- On omogućava da se više različitih iznimaka hvata unutar jednog „catch” bloka, a ima sljedeći način označavanja:

catch (PrviTipIznimke **|** DrugiTipIznimke **|** TreciTipIznimke **ex**)

- Navedeni „catch” blok omogućava obrađivanje svih navedenih iznimaka (ili njihovih podklasa)
- Koristi se u slučaju da obrada više različitih iznimaka izvodi isti programski kod
- Može sadržavati neograničen broj tipova iznimaka

Finally blok

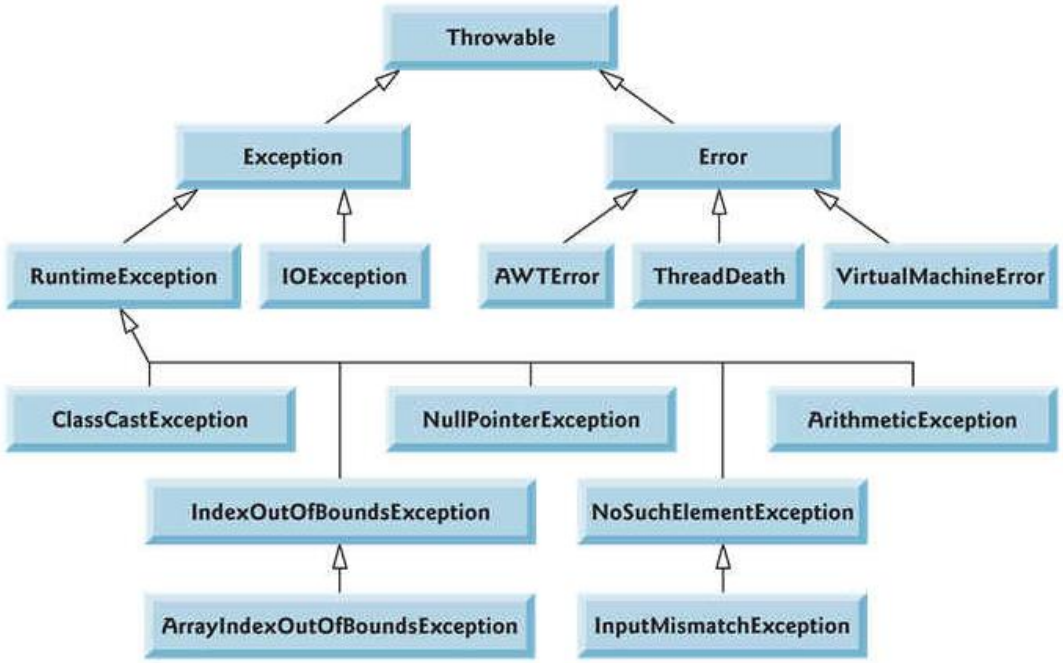
- Osim blokova „try” i „catch” postoji još jedna vrsta bloka naredbi povezanog s obradom iznimaka – „finally” blok
- Blok „finally” se postavlja nakon „catch” bloka (ili „try” bloka, ako „catch” ne postoji) i izvodi se **uvijek**, bez obzira je li došlo do iznimke ili ne
- Primjer:

```
try { … }

catch { … } //neobavezan

finally { … }
```

- Blok „finally” koristi se u slučaju kad je nakon izvođenja „try” bloka potrebno obaviti radnje poput zatvaranja datoteke ili veze s bazom podataka, koje su potrebne bez obzira je li došlo do iznimke ili ne

Hijerarhija iznimaka u Javi

- Sve klase u Javi izravno ili neizravno nasljeđuju klasu **Exception** koja ima nadklasu **Throwable**, što je prikazano na sljedećoj slici:



Hijerarhija iznimaka u Javi

- Kod obrađivanja iznimaka je moguće koristiti samo jedan „catch” blok koji hvata sve iznimke u Javi tako da mu se postavi tip iznimke „Throwable” – time se hvataju i svi podtipovi klase „Throwable”, odnosno sve iznimke u Javi
- Klasa „Exception” i njene podklase predstavljaju iznimke koje se događaju tijekom izvođenja programa i moraju se obrađivati
- Klasa „Error” i njene podklase predstavljaju ozbiljne pogreške unutar JVM-a i ne smiju se obrađivati – od nekih pogrešaka se program ne može oporaviti i najbolje da se prekine njegovo izvođenje

Označene i neoznačene iznimke u Javi

- U Javi postoje dvije vrste iznimaka: **označene** (engl. *checked*) i **neoznačene** (engl. *unchecked*)
- Java kompajler postavlja posebne zahtjeve na obrađivanje označenih iznimaka
- Vrsta iznimke određuje se klasom koju ta iznimka nasljeđuje: **Exception** ili **RuntimeException**
- Klase iznimaka koje izravno ili neizravno nasljeđuju klasu RuntimeException spadaju u skupinu neoznačenih iznimaka
- Obično označavaju **pogreške u programima**
- Primjer iznimaka: ArrayIndexOutOfBoundsException, ArithmeticException, ClassCastException, NullPointerException itd.

Označene i neoznačene iznimke u Javi

- U skupinu neoznačenih iznimaka spadaju i iznimke koje izravno ili neizravno nasljeđuju klasu Error
- Najvažnija karakteristika neoznačenih iznimaka je ta što se takve iznimke **ne moraju obrađivati** (kompajler ne javlja pogrešku ako se ne navede „catch” blok)
- Sve iznimke koje nasljeđuju klasu Exception (ako ne izravno i klasu RuntimeException) nazivaju se označenim iznimkama
- Označavaju uvjete kod izvođenja programa koje nisu pod izravnom kontrolom programa, na primjer kad ne postoji datoteka koju program otvara i slično
- Označene iznimke moraju se obrađivati, u suprotnom kompajler javlja pogrešku npr. „Unhandled exception type FileNotFoundException”

Bacanje iznimaka

- Iznimke, odnosno objekti klasa koje predstavljaju iznimke mogu se baciti korištenjem ključne riječi **throw**
- Na primjer, ako je potrebno baciti neoznačenu iznimku, kreiranje i bacanje iznimke moguće je napisati unutar jedne naredbe:

throw new RuntimeException(”Pogreška u programu!”);

- Na sličan način moguće je baciti i označenu iznimku, samo što tu iznimku mora obraditi programski kod u kojem se ona može dogoditi
- Ako se ne obrađuje označena iznimka na mjestu gdje se može dogoditi, moguće je kod metode koja uključuje taj programski kod navesti da ona prosljeđuje „odgovornost” obrađivanja iznimke kodu koji poziva tu metodu korištenjem ključne riječi **throws**

Bacanje iznimaka

- Na primjer, ako metoda „provjera” sadrži programski kod koji baca označenu iznimku „IOException”, ne mora je obrađivati ako je metoda označena ključnom riječju „throws” iza koje slijedi popis iznimaka (može ih biti više):
- public void provjera() **throws IOException** { … }
- Također, ako metoda baca iznimku korištenjem ključne riječi „throw”, **nema smisla** da se ta iznimka odmah i obrađuje (iako često IDE okruženje nudi tu opciju), već je potrebno koristiti ključnu riječ „throws”, npr.:

```
try {
   throw new IOException(”Pogreška u radu s datotekom!”);
}
catch(IOException ex) {…}
```

Kreiranje vlastitih klasa koje predstavljaju iznimke

- U praksi je često potrebno kreirati nove klase koje predstavljaju iznimke
- Vrstu iznimke (označena ili neoznačena) moguće je postaviti odabirom nadklase Exception ili RuntimeException
- Osim samog tipa iznimke preporuča se pisanje vlastitih konstruktora koji omogućavaju kreiranje objekta iznimke
- Preporuča se korištenje konstruktora bez parametara, konstruktora koji prima poruku o pogrešci, konstruktora koji prima uzročnu iznimku i konstruktora koji prima oba parametra

Kreiranje vlastitih klasa koje predstavljaju iznimke

```
public class MyException extends Exception {
    public MyException() {
        super("Dogodila se pogreška u radu programa!");
    }

    public MyException(String message) {
        super(message);
    }

    public MyException(String message, Throwable cause) {
        super(message, cause);
    }

    public MyException(Throwable cause) {
        super(cause);
    }
}
```

Pitanja?