Visoka tehnička škola Niš

Studijski program:

Savremene računarske tehnologije

Internet programiranje

(11)

Obrada izuzetaka u Javi

Prof. dr Zoran Veličković, dipl. inž. el.

Decembar, 2018.

Izuzeci

- **NEUOBIČAJENA STANJA** koja se javljaju **U TRENUTKU IZVRŠENJA** programa se nazivaju **IZUZECI** (engl. exceptions).
- Dakle, **IZUZETAK** je **GREŠKA** koj se otkriva tek pri **IZVRŠENJU** programa!
- **JAVA** poseduje **POSEBAN MEHANIZAM** za **OBRADU** ovih grešaka-**IZUZETAKA**.
- Pored ugrađenog mehanizma za **OBRADU IZUZETAKA**, Java poseduje i mehanizme za **IZAZIVANJE** (kaže se i ispaljivanje) **izuzetaka**!
- **REZERVISANE REČI** koje se koriste za za obradu i izazivanje **IZUZETAKA** su: **try**, **catch** (uhvati - izuzetak) i **throw** (baci - izuzetak).
- U radu sa izuzecima koriste se još **throws** i **finally** naredbe.
- Na **POJAVU IZUZETKA**, Javino izvršno okruženje – **JVM** formira predefinisani **OBJEKT** (slično kao kod događaja u C#) koji se **PROSLEĐUJE METODI** koja je izazvala grešku na **OBRADU**.
- **KATEGORIZACIJA** izuzetaka u Javi je prikazana u tabeli na sledećem slajdu.

Kategorizacija izuzetaka u Javi

R.br.	TIP IZUZETKA	OBRAZLOŽENJE
1	GREŠKE U KODU ILI PODACIMA	▪ Pokušaj **neispravne** konverzije objekata; ▪ Pokušaj korišćenja indeksa **izvan granica** niza; ▪ **Nula** kao delilac.
2	IZUZECI STANDARDNIH METODA	Primer: Metoda **substring()** klase **String** može da ispali izuzetak - **StringIndexOutOfBoundsException**.
3	ISPALJIVANJE SOPSTVENIH IZUZETAKA	Mogu se kreirati i ispaliti **KORISNIČKI** kreirani izuzeci.
4	JAVINE GREŠKE	KORISNIČKE greške nastale u programu.

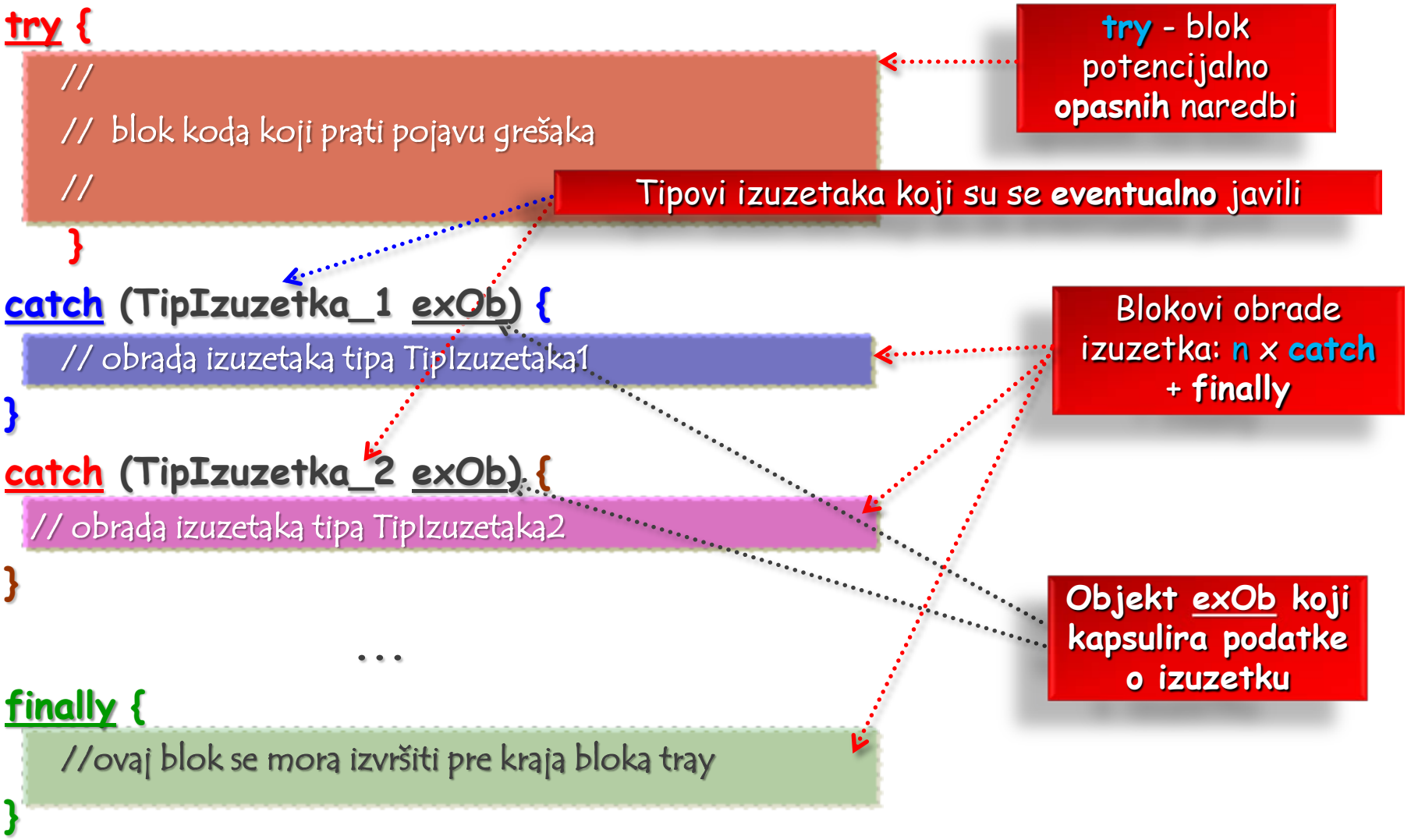
Obrada izuzetaka (1)

- Na pojavu **IZUZETKA**, **metoda** u kojoj je nastao, **POKUŠAVA** da razreši grešku, a ako u tome **NE USPE**, **PROSLEĐUJE GA DALJE** - **DRUGIM** metodama na obradu na **VIŠEM** hijerarhijskom nivou.
- **IZUZECI** se mogu **PROGRAMSKI GENERISATI** (kaže se "ručno" izazavni izuzeci) **ILI** ih **AUTOMATSKI** generiše sam **JAVIN IZVRŠNI SISTEM**.
- **JAVIN IZVRŠNI SISTEM** generiše izuzetke prilikom **narušavanja pravila jezika** ili **ograničenja** koja potiču od samog **izvršnog okruženja**.
- Uobičajeno je da se **RUČNO** (programsko) **ISPALJIVANJE IZUZETAKA** koristi za **PROSLEĐIVANJE stanje greške** metodi **pozivaocu**.
- Dobra programerska navika je da se **UNUTAR BLOKA try** smeste **SVE PROGRAMSKE NAREDBE** koje mogu izazavati izuzetak.
- Pomoću naredbe **catch** "**hvata se**" eventualno **NASTALI IZUZETAK** i potom se korisničkim **PROGRAMSKIM KODOM** pokušava da **REŠI NASTALI PROBLEM**.

Obrada izuzetaka (2)

- Već je rečeno, **Javin** izvršni sistem **AUTOMATSKI IZAZIVA** (kaže se "ispaljuje") **IZUZETKE** pri pojavi **SISTEMSKIH** grešaka.
- Sa druge strane, naredbom **throw**, izuzetak se može izazvati **PROGRAMSKI - RUČNO**.
- Međutim, ako se **NE ŽELI** obrada izuzetka metodom u kojoj je nastao, on se može **IZBACITI** iz procesa **obrade** (te metode) naredbom **throws**.
- Često je **NEOPHODNO IZVRŠITI** neki blok naredbi pre **POVRATKA IZ IZUZETKA**, tada taj blok treba označiti - deklarisati naredbom **finally**.
- Za **HVATANJE** ispaljenih izuzetaka, mogu se upotrebiti **VIŠE NAREDBI catch**, dakle omogućeno je da se uhvati **VIŠE TIPOVA** različitih izuzetaka.
- U Javi, blok naredbi **try**, može da bude **UGNEŽDEN**.
- Već je rečeno da se naredbom **throws** mogu **ISKLJUČITI IZUZECI** koji se mogu "uhvatiti" u **try - catch** bloku (iako se mogu ispaliti)!

Višestruki tray-catch-finally blokovi



Generička forma obrade izuzetaka

```
double doSomething( int aParam)
    throws ExceptionType1, ExceptionType2{

    //Code that does not throw exceptions

    //Set of try/catch/finally blocks...

    //Code that does not throw exceptions

    //Set of try/catch/finally blocks...

    //Code that does not throw exceptions

    //Set of try/catch/finally blocks...

    //Code that does not throw exceptions

    //...

}
```

Ovi izuzeci se NE HVATAJU (throws)

Tipična struktura: try-catch-finally

```
try{
        // Kod koji baca izuzetak
}
```

```
catch( MyException1 e){
    // Kod koji obrađuje izuzetak
}
```

```
catch( MyException2 e){
    // Kod koji obrađuje izuzetak
}
```

```
finally{
    // Kod koji se izvršava posle tray bloka
}
```

Ovi izuzeci se HVATAJU

Generalno se može imati onoliko **catch** blokova koliko Vam je potrebno (dakle, može biti ni jedan). Blok **finally** je **opcioni** kada se ima **catch** naredba

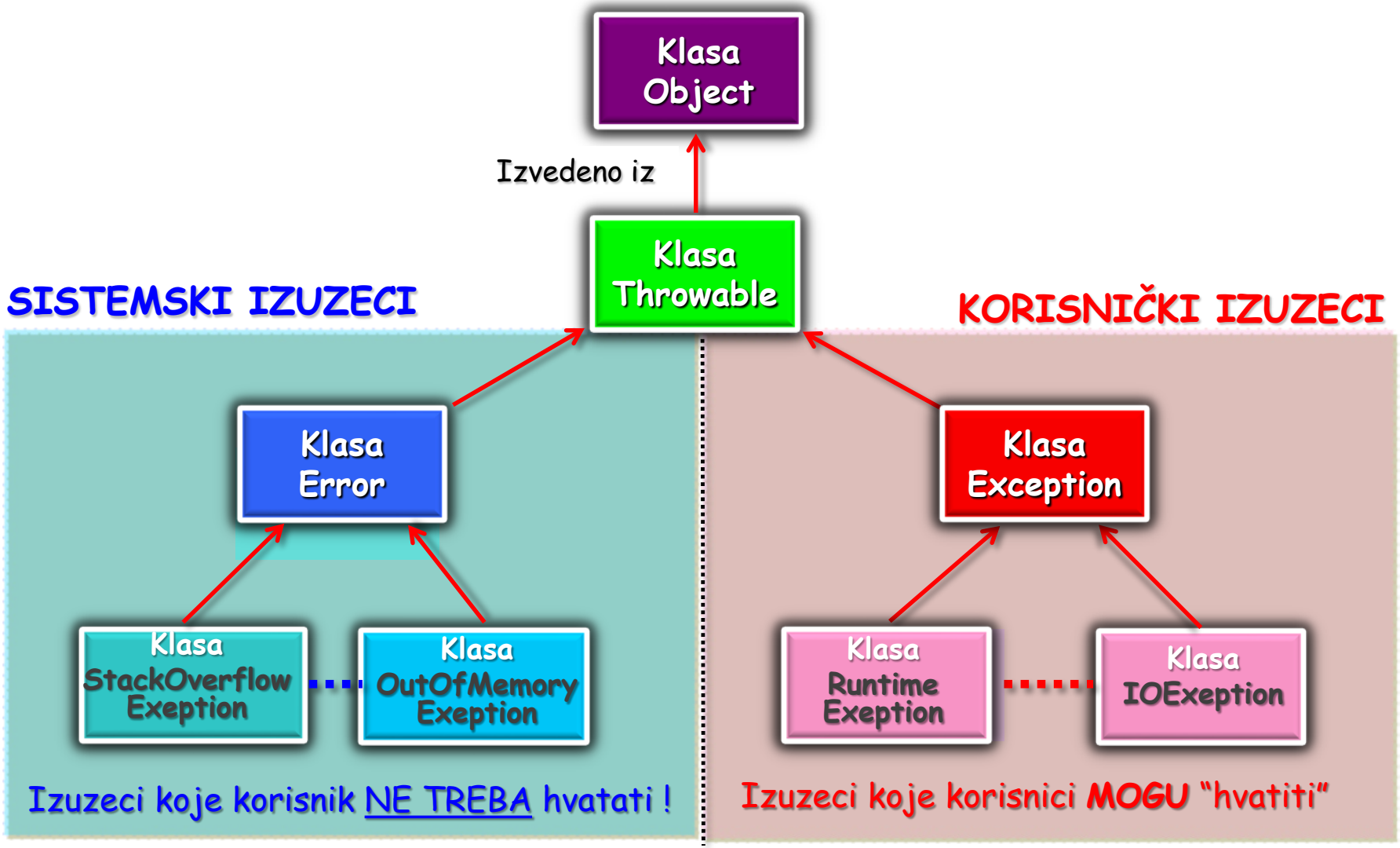
„Ručno“ bacanje izuzetaka

- Na pojavu izuzetka **UVEK** se formira **OBJEKT** neke od potklase klase **Throwable**.
- Dakle, **SVI** tipovi izuzetaka su **POTKLASE** ugrađene klase **Throwable**.
- Klasa **Exception** je **potklasa** klase **Throwable** i odnosi se na izuzetke koje treba da uhvate **KORISNIČKI PROGRAMI**
- **POTKLASE KLASE Exception** se koriste za hvatanje korisničkih **IZUZETAKA**.
- Bacanje **KORISNIČKOG IZUZETKA** se realizuje naredbom **throw**:

throw objekt_tipa_Throwable

- Jedna od često korišćenih **POTKLASA** klase **Exception** je **RuntimeException**.
- Izuzeci ovog tipa se **AUTOMATSKI DEFINIŠU** za programe koje piše **KORISNIK-PROGRAMER**.
- Sa druge strane, **JAVIN IZVRŠNI SISTEM** koristi **IZUZETAK** tipa **Error** da se označe izuzeci vezani za **JAVINO OKRUŽENJE** pri izvršenju programa.

Potklase klase Throwable



Klasa Error

- Izuzeci tipa **Error** nastaju kao posledica **FATALNIH GREŠAKA** i obično ih korisnički (Vaš) program **NE OBRAĐUJE** jer ih svakako ne može rešiti.
- Kada **Javin izvršni sistem OTKRIJE IZUZETAK** (npr. pokušaj deljenje nulom) prvo se **napravi NOVI OBJEKAT ZA TAJ IZUZETAK** a zatim se **BACA ODGOVARAJUĆI IZUZETAK**.
- Zapamtite, **SVAKI IZUZETAK** koji ne obradi Vaš program biće obrađen od strane **UGRAĐENOG** (standardnog) Javinog obrađivača.
- Od **UGRAĐENOG** obrađivača izuzetaka možete očekivati da ispiše tekst sa **opisom izuzetka** i **stanje PROGRAMSKOG STEKA** za metode koje su ga izazvale, a iza toga odmah **ZAVRŠAVA PROGRAM**!
- **CILJ** svakog programera je da probleme **NE REŠAVA SISTEM** (jer vam se to neće svideti!) već on **SAM** (programer).
- Već je napomenuto da **Java poseduje mehanizme** koje programer može koristiti za **HVATANJE** i **OBRADU** izuzetaka.

try-catch struktura

```
class MultiCatch {
  public static void main(String args[]) {
    try {
        int a = args.length;
        System.out.println("a = " + a);
        int b = 42 / a;
        int c[] = { 1 };
        c[42] = 99;
    } catch (ArithmeticException e) {
        System.out.println("Divide by 0: " + e);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Array index oob: " + e);
    }
    System.out.println("After try/catch blocks!.");
  }
}
```

Hvatanje DVA različita tipa izuzetaka

Parametri sa komandne linije

?

Više naredbi catch

Deljenje nulom posle catch bloka!

Bacanje izuzetka: throw naredba

```
class ThrowDemo {
  static void demoproc() {
    try {
          throw new NullPointerException ("demo");
        } catch (NullPointerException e) {
          System.out.println("uhvaćeno unutar procedure");
          throw e;                          // ponovno bacanje izuzetka !
    }
}
public static void main(String args[]) {
    try {
          demoproc();
        } catch (NullPointerException e) {
      System.out.println(„Ponovo uhvaćen: " + e);
   }
 }
}
```

Kreiranje objekta NullPointerException i bacanje izuzetka

Metoda demoproc() baca i hvata izuzetak tipa NullPointerException

Poziv metode za bacanje i hvatanje izuzetka

Hvatanje izuzetka tipa NullPointerException

2 puta hvatanje izuzetka tipa NullPointerException

Naredbe throws i finally

- Ako metoda **MOŽE DA IZAZOVE IZUZETAK**, a sama ga **NE OBRAĐUJE** treba ga **deklarisati** kao **throws**.
- U naredbi **throws** se **NABRAJAJU** svi tipovi izuzetaka koje metoda može da **BACI** (izuzimajući **Error** i **RuntimeExceptions**) ali ih **NE HVATA** pa samim tim i ne obrađuje.
- Naredbom **finally** treba formirati **BLOK NAREDBI** koje će se izvršavati **NEPOSREDNO PO ZAVRŠETKU** (odnosno izlasku) iz bloka **tray-catch**.
- Blok **finally** se izvršava **BEZ OBZIRA** na to da li je izuzetak ispaljen ili ne!
- Naredba **finally** je korisna prilikom **zatvaranja datoteka**, **otvorene mrežne konekcije** i **oslobađanja drugih resursa**.
- Za **SVAKU** naredbu **try** potrebna je **BAR PO JEDNA** odredba:
  - **catch** ili
  - **finally**!

Primer: finally naredba

```
// Demonstracija finally naredbe.

class FinallyDemo {
 // Through an exception out of the method.
  static void procA() {
    try {
            System.out.println("unutar procA");
               throw new RuntimeException ("demo");
    } finally {
        System.out.println("procA's finally");
    }
 }
```

Obrada ugrađenih izuzetaka (1)

- Već je pomenuto da Java poseduje **UGRAĐENE IZUZETKE**.
- Ovi izuzeci se odnose na **ARITMETIČKE OPERACIJE** (deljenje nulom), na **GRANICE NIZOVA**, **NEPOSTOJANJA PROMENLJIVIH**, ...
- U standardnom paketu **java.lang** definisano je **VIŠE KLASA IZUZETAKA**.
- Većina izuzetaka su **POTKLASE** standardnog tipa **RuntimeException**.
- Spisak **SVIH IZUZETAKA** klase **RuntimeException** i njihov opis je dat na sledećem slajdu.
- Sećate se, paket **java.lang** se **PODRAZUMEVANO UVOZI** u **SVAKI** Java program.
- Ove izuzetke **NE TREBA** nabrajati u listi iza **throws** (moraju se obraditi).
- Ovo su takozvani **NEPROVERAVANI IZUZECI** - ne proveravaju se u **VREME KOMPAJLIRANJA**.
- Lako je zaključiti da onda postoji lista i **PROVERAVANIH** izuzetaka.

Neproveravani izuzeci (1)

NEPROV. IZUZETAK	ZNAČENJE
ArithmeticException	Pojava nedozvoljenog **aritmetičkog stanja**, kao što je recimo deljenje celog broja nulom.
IndexOutOfBoundsException	Pokušaj korišćenja indeksa čija je vrednost **izvan granica** prihvatljivih za objekt.
NegativeArraySizeException	Pokušaj definisanja niza sa **negativnom** dimenzijom.
NullPointerException	Pokušaj korišćenja promenljive čija je vrednost **null**.
ArrayStoreException	Pokušaj smeštanja u niz objekata koji **ne odgovara** tipu.
ClassCastException	Pokušaj **konverzije objekta** u nedozvoljeni tip.
IllegalArgumentException	Prosleđivanje argumenata metodi koji **ne odgovaraju** po tipu.
SecurityException	Pokušaj izvršavanja **nedozvoljene operacije** - ugrožena bezbednost.

Neproveravani izuzeci (2)

NEPROV. IZUZETAK	ZNAČENJE
IllegalMonitorStateException	Pokušaj niti da **sačeka** na monitor koji joj ne pripada.
IllegalStateException	Pokušaj **poziva metode** u trenutku kada to nije dozvoljeno.
UnsupportedOperationException	Pokušaj izvođenja operacije koja **nije podržana**.

- U tabeli su prikazani tzv. **NEPROVERAVANI IZUZECI** jer **KOMPAJLER NE PROVERAVA** da li ih metoda ispaljije ili obrađuje.
- Provera ovih izuzetaka se obavlja tek u **VREME IZVRŠAVANJA** što predstavljati problem za **OTKRIVANJE GREŠAKA**.
- Dakle, ove greške se ne mogu otkriti kompajlerom.

Proveravani izuzeci

PROV. IZUZETAK	ZNAČENJE
ClassNotFoundException	Klasa **nije** nađena.
CloneNotSupportedException	Pokušaj **kloniranja** objekta bez intefejsa Cloneable.
IllefalAccesException	**Odbijen zahtev** za pristup klasi.
InstantionalException	Pokušaj da se napravi objekt **abstraktne klase** ili interfejsa.
InterruptedException	Jedna programska nit je **prekinula** drugu.
NotSuchFieldException	Zahtevano polje **ne postoji**.
NotSuchMethodException	Zahtevani metod **ne postoji**.

- Ovi izuzeci se proveravaju u vreme kompajliranja.
- Ovo su tzv. **PROVERAVANI IZUZECI** koji se **MORAJU UKLJUČITI** u listu **throws**.

Obrada ugrađenih izuzetaka(2)

```
class Exc2 {
  public static void main(String args[]) {
    int d, a;
    try {
          d = 0;
          a = 42 / d;
       System.out.println("Ovo se nikada neće printati.");
    } catch (ArithmeticException e) {      // hvata pokušaj deljenja nulom
       System.out.println("Deljenje nulom.");
    }
    System.out.println("Posle catch naredba.");
  }
}
```

Monitoriše blok koda

Zašto?

Kada se izvršava?

Hvata eventualnu neregularnost deljenja nulom (**divide-by-zero** error)

Obrada sopstvenih izuzetaka (1)

- Pravljenje **SOPSTVENIH IZUZETAKA** je neophodno za razrešavanje **SOPSTVENIH GREŠAKA**.
- Za ove potrebe koristi se potklasa **Exception** (ona je takođe potklasa klase **Throwable**).
- Klasa **Exception** **NE DEFINIŠE SOPSTVENE METODE**, već **NASLEĐUJE** metode svoje natklase **Throwable** i to:
  - **toString()**,
  - **printStackTrace()**,
  - **getMessage()**, ...
- Na ovaj način, **KORISNIČKIM IZUZECIMA**, stoje na raspolaganju **SVE METODE** **natklase** **Throwable**.
- Omogućeno je i **ULANČAVANJE IZUZETAKA**.
- Na sledećem slajdu je prikazana tabela sa **nekim** od raspoloživih metoda klase **Throwable**.

Deo javnih metoda klase Throwable

METOD	OPIS
String getMessage()	Vraća sadržaj poruke, opisujući time aktuelni izuzetak. Najčeće je to puno kvalifikovano ime klase izuzetaka i kratak opis izuzetka.
void printStackTrace()	Prikazuje poruku i evidenciju praćenja steka izvršavanja u standardnom izlaznom toku greške. Kod konzolnih programa to je ekran.
void printStackTrace(PrintStream s)	Identičan prethodnom, stim što se kao argument dostavlja **izlazni tok**. Primer: e.printStackTrace(**System.err**)
String toString()	Vraća objekt tipa String sa opisom izuzetaka
String getLocalizedMessage()	Vraća lokalizovan opis izuzetka

Obrada sopstvenih izuzetaka (2)

```
class MyException extends Exception {
  private int detail;
  MyException(int a) {
    detail = a;
  }
  public String toString() {
    return "MyException[" + detail + "]";
  }
} // end class

class ExceptionDemo {
  static void compute(int a) throws MyException {
    System.out.println("Called compute(" + a + ")");
    if(a > 10)
      throw new MyException(a);
    System.out.println("Normalan izlaz");
  } // end class
```

Pravi se nov tip izuzetka

Konstruktor **nove klase**

Redefinisanje metode **toString()**: prikazuje se kao vrednost izuzetka

Metoda **compute()** baca izuzetak **MyException** kada je a > 10

Ispaljuje izuzetak **MyException** ali ga sama metoda ne obrađuje

Obrada sopstvenih izuzetaka (3)

```
// Test, nastavak prethodnog primera

public static void main (String args[]) {
    try {
            compute(1);
            compute(20);
    } catch (MyException e) {
      System.out.println("Uhvaćen" + e);
    }
  }
}
```

2x se baca izuzetak sa različitim parametrima

Obrađivač izuzetaka

```
Called compute (1)
Normalan izlaz
Called compute (20)
Uhvaćen MyException
```