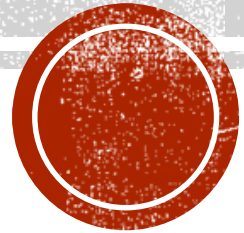
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА – 1

Изузеци и тврдње



ИЗУЗЕЦИ И ТВРДЊЕ

- Један од кључних изазова у програмирању је писање програма без грешака у фази извршавања или тзв. багова (енг. bugs).
- Такође је пожељно да се програми не гасе, посебно у ситуацијама када људски животи зависе од извршавања програма (нпр. аутопилоти).
- Уколико би за написани програм постојао формални доказ (верификација) да никад не може да погреши, онда би једини ризици били спољни фактори.
  - Формална верификација програма је тешка, скупа, а често и немогућа у императивним програмским језицима.
- Јава прибегава употреби две корисне технике за контролу грешака:
  - Изузеци;
  - Тврдње.

ИЗУЗЕЦИ

- Приликом извршавања програма, ако се појави грешка, настаје изузетна ситуација и стога се овај догађај назива изузетак.
- Две кључне користи од употребе изузетака су:
  - раздвајање кода који обрађује неочекиване ситуације од кода који се извршава када је све очекивано;
  - присиљавање програмера да размотри и реагује на одређене врсте грешака.
- Не треба све грешке у програмима третирати као изузетке – довољно је само неуобичајене или катастрофалне.
  - На пример, ако корисник не унесе исправан улазни податак, за то не треба користити изузетке. Рационалније је обавестити корисника о лошем уносу, дати шансу за нови унос.

ИЗУЗЕЦИ (2)

- Руковање изузецима укључује много додатног процесирања и самим тим успоравање извршавања програма.
- Унутар Јава програма изузетак се моделује као објекат који у себи носи информацију о непредвиђеној ситуацији која се десила.
- Када се деси непредвиђена ситуација унутар тела неког метода, она ја праћена наредбом за тзв. избацивање (креирање) изузетка.
- Надаље, изузетак (објекат) може бити:
  - прослеђен (пропагиран) над-методу, тј. методу који позива актуелни метод;
  - обрађен (или “ухваћен”) унутар неког од метода који претходи позиву метода у којем се десио.

ПРИМЕР 1

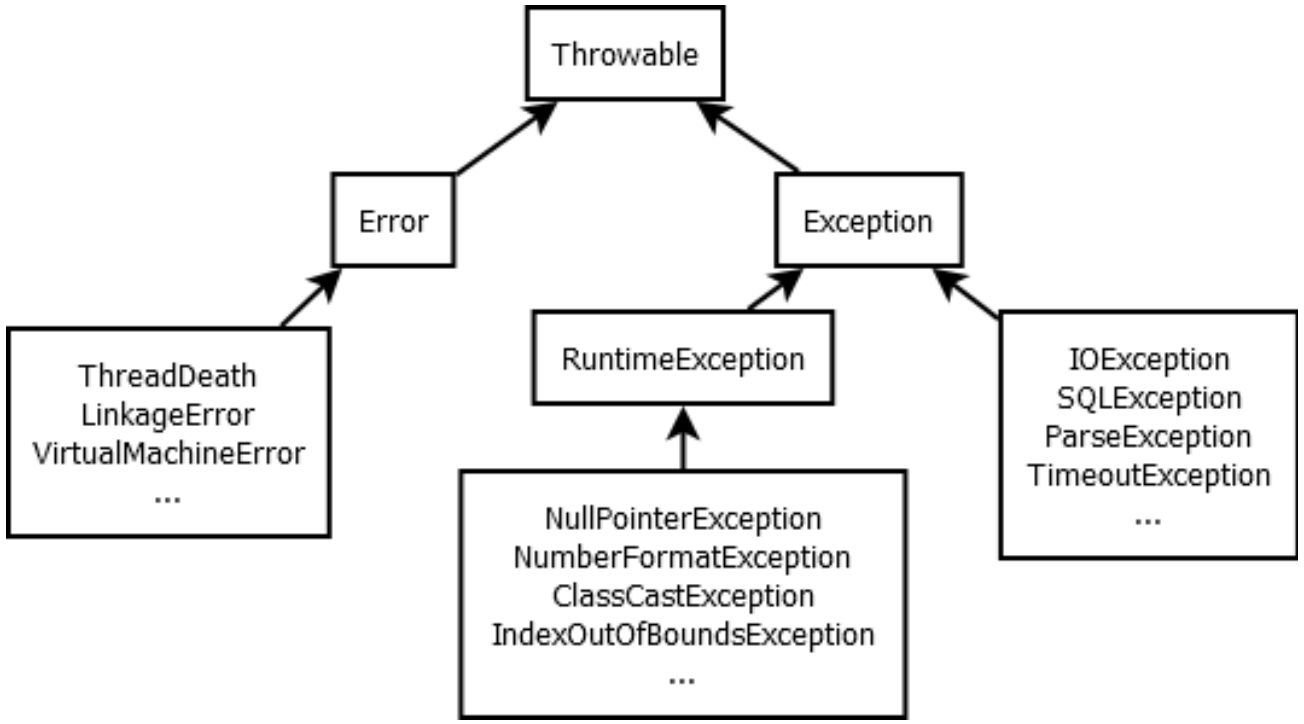
- Написати Јава програм који покушава да приступи елементу низа ван дозвољених граница.
- Ухватити изузетак и обрадити га унутар **main()** метода.

```
public class IndeksVanGranica {

    public static void main(String[] args) {
      int a[] = new int[2];
      System.out.println("Приступам елементу:" + a[3]);
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3
out of bounds for length 2
    at
rs.math.oop.g11.p01.izuzeciIndeksVanGranica.IndeksVanGranica.main(IndeksVanGra
nica.java:7)
```

ТИПОВИ ИЗУЗЕТАКА



ИЗУЗЕЦИ ТИПА ERROR

- За изузетке класе **Error** и њених потклаcа
  од програмера се не очекује да предузима неку акцију, тј. не очекује се да их обрађује.
- Ове врсте грешака се ретко јављају, али у Јави се и оне контролишу.
- Класа Error има неколико директних потклаcа, а неке значајније су:
  - ThreadDeath – избацује изузетак када се нит, која се извршава, намерно заустави.
  - LinkageError – указује на озбиљне проблеме са класама у програму.
  - VirtualMachineError –избацују се када се деси катастрофални пад JVM.
- Изузеци класа изведених из LinkageError и VirtualMachineError су резултат катастрофалних догађаја и услова.
  - У таквим ситуацијама програмер само може да констатује поруку о грешци и евентуално да је запамти у датотеци, бази података или слично.
  - На основу поруке треба да покуша да схвати шта је у написаном коду могло да изазове такав проблем како би тај проблем отклонио.

ПРИМЕР 2

- Написати Јава програм који демонстрира рад нетачне рекурзивне имплементације метода за рачунање факторијела (нема изласка из рекурзије).

```
public class FaktorijelBezUslovaIzlaska {

    static int faktorijelBezUslovaIzlaska(int n) {
      return n * faktorijelBezUslovaIzlaska(n - 1);
    }

    public static void main(String[] args) {
      System.out.println(faktorijelBezUslovaIzlaska(10));
    }
}
```

```
Exception in thread "main" java.lang.StackOverflowError
    at
rs.math.oop.g11.p02.izuzeciPrekoracenjeStekMemorije.FaktorijelBezUslovaIzlaska.fak
torijelBezUslovaIzlaska(FaktorijelBezUslovaIzlaska.java:6)
    at
```

ИЗУЗЕЦИ ТИПА RUNTIMEEXCEPTION

- За скоро све изузетке представљене потокласама класе **Exception** може се у програм укључити код који ће их обрађивати, осим за објекте класе **RuntimeException**.
- Преводилац допушта да програмер игнорише ове изузетке, јер су они најчешће последица логичких грешака у написаном програмском коду.
- Тиме се имплицитно препоручује измена програма.
- На пример, приступ елементу чији индекс је ван граница низа се сигнализира изузетком класе која је потокласа **RuntimeException**.
  - Очигледно је да овај изузетак може да се спречи одговарајућим кодом па Јава не стимулише његово обрађивање.

ПРИМЕР 3

- Написати Јава програм у којем се спречава појава референсирања на елемент чији индекс је ван граница индекса низа, тј. спречава се појава изузетка класе **ArrayIndexOutOfBoundsException.**

```
int a[] = new int[] { 1, 2, 3, 4, 5 };
System.out.println("Унесите индекс елемента којем приступате:");
Scanner skener = null;
skener = new Scanner(System.in);
int i = skener.nextInt();
if (i < 0 || i >= a.length)
        System.err.println("Индекс ван граница.");
else
        System.out.println("Број на траженом индексу је " + a[i]);
skener.close();
```

ПОТКЛАСЕ RUNTIMEEXCEPTION

- Постоји велики број класа које су директне поткласе класе **RuntimeException**, а неке од познатијих су:
  - **ArithmeticException** — недозвољена ситуација у примени аритметичких операција, нпр. дељење нулом.
  - **IndexOutOfBoundsException** — индекс низа којем се приступа није валидан.
  - **NegativeArraySizeException** — алокација низа са негативном димензијом.
  - **NullPointerException** — покушај приступа методу или пољу инстанцне променљиве која има специјалну вредност **null**.
  - **ClassCastException** — покушај конверзије инстанцне променљиве у недозвољени тип.

ИЗУЗЕЦИ ТИПА EXCEPTION

- Већ смо истакли да програмер не може да обрађује изузетке класе **Error.**
- Са друге стране, када се деси **RuntimeException**, могућа је поправка, али то треба да буде само привремена мера, а да се прави проблем реши у новој верзији програма.
- Када су у питању сви остали типови изузетака из класе **Exception**, они нису последица лошег програмирања попут изузетка класе **RuntimeException** нити су непоправљиве попут изузетака **Error.**
- Стога ће Јава компајлер, у тим ситуацијама, проверити да ли је примењен један од наредна два механизма за обраду изузетака:
  - хватање изузетка (try-catch блок);
  - прослеђивање изузетка (наредба throws).
- Уколико није урађено ни једно ни друго, програм неће бити преведен.

ПОТКЛАСЕ EXCEPTION

- Постоји велики број директних потклаcа класе **Exception** (изузев **RuntimeException**), а неке од њих су:
  - **IOException** — општи изузетак у вези са улазно/излазним операцијама. Нпр. класа **FileNotFoundException**, указује на непостојање тражене датотеке.
  - **ParseException** — указује на проблем у парсирању текста.
  - **SQLException** — општи проблем при повезивању са базом података или при прављењу **SQL** упита, обради резултата упита и слично.
  - **TimeoutException** — изузетак који се избацује када истекне време за чекање неке блокирајуће операције — најчешће се појављује у вишенитном програмирању.

РУКОВАЊЕ ИЗУЗЕЦИМА

- Претпоставимо да метод **f()** позива други метод **g** који може избацити изузетак.
  - Притом изузетак није **RuntimeException** нити **Error** типа.
    Нека је изузетак, на пример, типа **IOException.**

```
double f(){
   ...
   g(); // избацује IOException
   ...
}
```

- Један начин реаговања на изузетак може бити додавање специјалне наредбе **throws** у оквиру декларације метода, тј. прослеђивање изузетка методу позиваоцу.
  - Та наредба је праћена типом изузетка који може бити избачен унутар метода.

```
double f() throws IOException { ... }
```

РУКОВАЊЕ ИЗУЗЕЦИМА (2)

- Ако постоји више изузетака, онда се формира листа типова раздвојених зарезом.

```
double f(){
   ...
   g(); // избацује IOException
   ...
   h(); // избацује FileNotFoundException или SQLException
   ...
}
```

- У том случају би декларација метода f() могла да буде нека од следећих:

```
double f() throws IOException, FileNotFoundException, SQLException {...}
double f() throws IOException, SQLException {...}
double f() throws Exception {...}
```

РУКОВАЊЕ ИЗУЗЕЦИМА (3)

- Други начин реаговања је обрада изузетака у оквиру метода од интереса.
- За те сврхе потребно је укључити три блок-наредбе при чему је трећа опциона:
  - **try** блок – код који може да избаци један или више изузетака мора бити унутар **try** блока.
  - **catch** блок – обухвата код који је намењен руковању изузецима одређеног типа, који могу бити избачени у придруженом **try** блоку.
  - **finally** блок – увек се извршава пре него се метод заврши, без обзира да ли је било који изузетак избачен у **try** блоку или не.

```
double f(){
  try{
    g(); // избацује IOException
  }catch(IOException ex){
    ...
  }
  try{
    h(); // избацује FileNotFoundException или SQLException
  }catch(Exception ex){
    ...
  }
}
```

РУКОВАЊЕ ИЗУЗЕЦИМА (4)

- Могуће је и комбиновати поменута два начина,
  тј. неке изузетке само проследити даље, а неке хватати.
- На пример, претходно описан метод **f()** је могао да обради само **SQLException**,
  а остале да проследи помоћу **throws** наредбе.

```
double f() throws IOException, FileNotFoundException{
  g(); // избацује IOException
  try{
    h(); // избацује FileNotFoundException или SQLException
  }catch(SQLException ex){
    ...
  }
}
```

- Уколико се за изузетак (који није **RuntimeException** или **Error**)
  не уради ни једно ни друго, доћи ће до грешке приликом компилације.

TRY БЛОК

- Када треба да се ухвати изузетак, код метода који може избацити изузетак мора бити обухваћен **try** блоком.

```
try {
  // код који може избацити један или више изузетака
}
```

- Блок **try** је неопходан и када желимо да хватамо изузетке типа **Error** или **RuntimeException.**
- У оквиру **try** блока се могу наћи и наредбе које не производе изузетак, што суштински значи да се читав код метода може обухватити **try** блоком.
  - Ипак, препорука је да се ово не ради, већ да се **try** блоком обухвата минимална секвенца наредби која производи изузетак или изузетке од интереса.

CATCH БЛОК

- Код за руковање изузетком одређеног типа треба да буде ограђен витичастим заградама у **catch** блоку.
- Блок **catch** се мора налазити непосредно иза **try**.
- Блок **catch** се састоји од кључне речи **catch** праћене једним параметром унутар облих заграда за идентификацију типа изузетка којим блок рукује.
- Ово прати код за руковање изузетком који се налази унутар витичастих заграда.

```
try {
  // код који може избацити један или више изузетака
}catch(FileNotFoundException e) {
  // код за руковање изузетком типа FileNotFoundException
}
// извршавање се наставља овде...
```

ВИШЕСТРУКИ CATCH БЛОК

- У претходном примеру, у **catch** блоку се руковало само изузецима типа **FileNotFoundException.**
- Ако могу бити избачени и други (изузев **Error** и **RuntimeException**), претходни код се неће успешно превести.

```
try {
  // код који може избацити један или више изузетака
}catch(FileNotFoundException e) {
  // код за руковање изузетком типа FileNotFoundException
}catch(IOException e) {
  // код за руковање изузетком типа IOException
}
// извршавање се наставља овде...
```

- У навођењу **catch** блокова потребно је водити рачуна о хијерархији изузетака и увек прво наводити оне специфичније па потом оне општије.

FINALLY БЛОК

- Природа изузетака је таква да се извршавање **try** блока прекида по избацивању изузетка, без обзира на значај кода који следи тачку у којој је изузетак избачен.
  - Тиме се ствара могућност да изузетак изазове неконзистентно стање делова програма. На пример, може се догодити да се отвори датотека, и да се не стигне до њеног затварања.
- **finally** блок обезбеђује решење овог или сличних проблема.
  - Овај блок се извршава увек, без обзира да ли се десио изузетак.
- Ако нема **catch** блокова, **finally** блок се смешта непосредно након **try** блока.

```
try{
  // код који може избацити изузетке...
}catch(ExceptionType1 e) {
  // ...
}catch(ExceptionType2 e) {
  // ...
}finally{
  // код који се увек извршава након try-блока
}
// извршавање се наставља овде...
```

ПРИМЕР 5

- Написати Јава програм који у петљи прихвата текст са стандардног улаза.
- У свакој итерацији петље покушати парсирање унетог текста у датум формата “dd.MM.yyyy”.
- Корисник треба да уноси текст све док га не унесе у валидном формату — након тога прекинути извршавање петље, односно програма.

ПРИМЕР 5 (2)

```
LocalDate datum = null;
Scanner skener = null;
DateTimeFormatter datumFormat = DateTimeFormatter.ofPattern("dd.MM.yyyy");
Boolean unetValidanFormat = false;
try {
    skener = new Scanner(System.in);
    while (!unetValidanFormat) {
                try {
                  System.out.println("Датум dd.MM.yyyy:");
                  String datumString = skener.next();
                  datum = LocalDate.parse(datumString, datumFormat);
                  System.out.println("Валидан датум: " + datum);
                  unetValidanFormat = true;
                  } catch (DateTimeParseException e) {
                  System.out.println("Погрешан формат датума!");
                  }
      }

} finally {
    skener.close();
}
```

```
Датум dd.MM.yyyy:
10-12-2022
Погрешан формат датума!
Датум dd.MM.yyyy:
10.12.2022
Валидан датум: 2022-12-10
```

ПРОСЛЕЂИВАЊЕ (ПРОПАГИРАЊЕ) ИЗУЗЕТАКА

- У ситуацијама када се у неком методу појави изузетак, програмер може да одлучи да тај изузетак не хвата, већ да га проследи (пропагира) у позивајући метод.
- Мотивација за такву одлуку може бити боље познавање околности под којима је изузетак настао у позивајућем методу.
- Као што је раније објашњено, прослеђивање изузетка у позивајући метод се реализује помоћу кључне речи **throws**, иза које следи листа изузетака.

ПРИМЕР 6

- Написати Јава програм за решавање проблема из примера 5, притом је парсирање потребно издвојити као засебни статички метод.
- (Напомена: с обзиром да овај метод неће имати контролу над главном петљом у којој се уноси текст од стране корисника, пропагација изузетка у **main()** метод има више смисла, него реаговање на изузетак унутар њега.)

ПРИМЕР 6 (2)

```
public class ParsiranjeDatumaPropagiranjeIzuzetka {

    final static DateTimeFormatter datumFormat =
          DateTimeFormatter.ofPattern("dd.MM.yyyy");

    static LocalDate parsirajDatum(String datumString) throws DateTimeParseException{
      LocalDate datum = LocalDate.parse(datumString, datumFormat);
      return datum;
    }

    …
```

ПРИМЕР 6 (3)

```
    public static void main(String[] args) {
      LocalDate datum = null;
      Scanner skener = null;

      Boolean unetValidanFormat = false;
      try {
          skener = new Scanner(System.in);
          while (!unetValidanFormat) {
              try {
                  System.out.println("Datum dd.MM.yyyy:");
                  String datumString = skener.next();
                  datum = parsirajDatum(datumString);
                  System.out.println("Валидан датум: " + datum);
                  unetValidanFormat = true;
              } catch (DateTimeParseException e) {
                  System.out.println("Погрешан формат датума!");
              }
          }
      } finally {
          skener.close();
      }
    }
}
```

ИЗБАЦИВАЊЕ ИЗУЗЕТАКА

- Избацивање изузетка се врши наредбом **throw.**
- На пример, метод за дељење, скраћено записан као оператор “/”, може у својој реализацији имати проверу да ли је делилац нула.
- Ако јесте, применом наредбе **throw** се креира објекат класе **ArithmeticException** са одговарајућом поруком да је у питању дељење нулом.

ПРИМЕР 7

- Написати Јава програм који демонстрира сабирање одговарајућих елемената два дводимензионална низа. (Елементи су одговарајући ако имају исте индексе.).
- Уколико се сабирање односи на два низа који немају идентичне димензије, потребно је избацити нови тип изузетка.

ПРИМЕР 7 (2)

```
public class Niz2DIzuzetak extends Exception {

    public Niz2DIzuzetak(String poruka) {
      super(poruka);
    }

    static double[][] saberi2DNizove(double[][] a, double[][] b) throws Niz2DIzuzetak {
      if (a.length != b.length)
          throw new Niz2DIzuzetak("Лоше спољне димензије.");
      double[][] c = new double[a.length][];
      for (int i = 0; i < a.length; i++) {
          if (a[i].length != b[i].length)
              throw new Niz2DIzuzetak("Лоше унутрашње димензије.");
          c[i] = new double[a[i].length];
          for (int j = 0; j < c[i].length; j++)
              c[i][j] = a[i][j] + b[i][j];
      }
      return c;
    }
```

ПРИМЕР 7 (3)

```
public static void main(String[] args) {
      double[][] m1 = new double[][] { { 1, 2, 3 }, { 4, 5, 6 } };
      double[][] m2 = new double[][] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
      double[][] m3 = new double[][] { { 1, 1, 1 }, { 1, 1, 1 } };
      try {
          double[][] m4 = saberi2DNizove(m1, m3);
          System.out.println("Прво сабирање успело.");
          double[][] m5 = saberi2DNizove(m1, m2);
          System.out.println("Друго сабирање успело.");
      } catch (Niz2DIzuzetak e) {
          System.err.println(e);
      }
    }
}
```

ПРЕПОРУКЕ ЗА РАД СА ИЗУЗЕЦИМА

- **finally** блок треба користити за затварање отворених ресурса, нпр. Scanner објекта.
- изузетке из потkласа класе **RuntimeException** као и класе **Error** не треба обрађивати суштински, већ их само треба документовати у фази употребе програма (продукцији)
- преферирати што специфичније изузетке који боље описују проблематику;
- унутар вишеструких **catch** блокова најпре хватати специфичне;
- не треба хватати најопштији могући изузетак из класе **Throwable**;
- у фази програмирања не игнорисати изузетке, посебно оне изведене из класе RuntimeException. Они често указују на проблеме који ће у фази употребе програма постати још израженији;
- не претеривати са употребом изузетака будући да њихова пропагација и хватање може изазвати одређено успорење рада програма.

ТВРДЊЕ

- Тврдња је оператор који омогућава проверу испуњености неког услова у програму.
  - На пример, може се проверавати:
    ненегативност индекса елемента у низу,
    да ли је површина круга позитивна,
    да ли је брзина кретања честице у симулацији мања од брзине светлости и слично.
- Уколико услов није испуњен програм аутоматски избацује грешку и прекида се.
- Овај ригорозни приступ проверавању испуњености услова помаже програмеру током програмирања у разрешавању грешака и повећавању поузданости програма.

НАРЕДБА ASSERT

- Тврдње се реализују помоћу оператора **assert** и класе **java.lang.AssertionError.**
- Тврдња започиње кључном речи **assert** након чега следи логички израз.

```
assert <логички израз>;
```

- Ако је израз тачан, ништа се не дешава, само се наставља са наредном наредбом.
- У супротном се креира објекат класе **AssertionError.**
- Програм проверава, у току извршавања, тврдње само ако су оне активиране.
  - Што се постиже задавањем аргумента -ea (енг. enable assertions) виртуалној машини.
  - На овај начин се једноставно, без измене програмског кода,
    може прелазити из режима употребе у режим програмирања и отклањања грешака.

ПРИМЕР 8

- Написати Јава програм који помоћу тврдње проверава да ли је број ненегативан.

```
public class NenegativanBrojTvrdnja {
	public static void main(String[] args)
	{
		int x = -1;
		assert x >= 0;
	}
}
```

```
Exception in thread "main" java.lang.AssertionError
    at
rs.math.g10.p08.tvrdnjeNenegativanBroj.NenegativanBrojTvrdnja.main(Nenegativan
BrojTvrdnja.java:8)
```

ПРИМЕР 9

- Написати Јава програм који ради исто што и програм за пример 8 са том разликом што помоћу тврдње треба да се испише и опис грешке.

```
public class NenegativanBrojTvrdnjaSaPorukom {

	public static void main(String[] args)
	{
		int x = -1;
		assert x >= 0 : "x < 0";
	}
}
```

```
Exception in thread "main" java.lang.AssertionError: x < 0
    at
rs.math.g10.p09.tvrdnjeNenegativanBrojSaPorukom.NenegativanBrojTvrdnjaSaPoruko
m.main(NenegativanBrojTvrdnjaSaPorukom.java:8)
```

ПРИМЕР 10

- Честа употреба тврдњи је у проверавању такозваних предуслова или постуслова, односно стања пре или после извршавања неког метода.
- Написати Јава програм који помоћу тврдње проверава да ли метод за сортирање низа заиста сортира низ (постуслов).

ПРИМЕР 10 (2)

```
private static boolean jeSortiran(int[] x) {
    for (int i = 0; i < x.length - 1; i++)
        if (x[i] > x[i + 1])
            return false;
    return true;
}

private static void sortiraj(int[] x) {
    int j, a;
    for (int i = 1; i < x.length; i++) {
        a = x[i];
        j = i;
        while (j > 0 && x[j - 1] > a) {
            x[j] = x[j - 1];
            j--;
        }
        x[j] = a;
    }
}
```

ПРИМЕР 10 (3)

```
public static void main(String[] args) {
    int[] niz = { 20, 91, -6, 16, 0, 7, 51, 42, 3, 1 };
    sortiraj(niz);
    assert jeSortiran(niz) : "низ није сортиран";
    for (int e : niz)
        System.out.printf("%d ", e);
    System.out.println();
}
```

```
-6 0 1 3 7 16 20 42 51 91
```

- Ако би позив метода за сортирање био коментарисан, добио би се следећи испис.

```
Exception in thread "main" java.lang.AssertionError: низ није сортиран
    at
rs.math.oop.g11.p10.tvrdnjeSortiranjePostuslov.SortiranjeSaPostuslovom.main(So
rtiranjeSaPostuslovom.java:9)
```

ПРЕПОРУКЕ ЗА РАД СА ТВРДЊАМА

- Битно је уочити разлику између намене тврдњи и намене изузетака.
- Тврдње се користе да би програм указао на неприхватљиве околности.
  - Стога је потребно да програмер поправи програм и избегне такве околности.
  - Овим се не гарантује да програм ради оно што се очекује, али добром "покривеношћу" тврдњама смањује се шанса да се деси нека неприхватљива околност у фази употребе.
- Изузеци се користе када је реч о грешкама које су зависне и од других фактора, а не само од програмерске логике.
  - Примери оваквих грешака су недозвољена/неочекивана стања система датотека, неадекватни уноси од стране корисника и слично.
- У фази употребе програма (продукционом окружењу) тврдње се обично онеспособљавају.

ПИТАЊА И ЗАДАЦИ

- Шта су изузеци и које су предности употребе изузетака?
- Илустровати примером када је у програму потребно проблем обрадити употребом изузетака, а када је довољно исписати само информативну поруку кориснику.
- Објаснити хијерархију изузетака у програмском језику Јава.
- Објаснити и илустровати примером ситуацију када долази до изузетка типа **Error**, изузетка типа **RuntimeException**, а када до изузетка типа **Exception**.
- Који су могући начини руковања изузецима? Илустровати примером.
- Објаснити и илустровати примером ситуацију када се користи вишеструки **catch** блок. О чему посебно треба водити рачуна при употреби вишеструког **catch** блока?

ПИТАЊА И ЗАДАЦИ (2)

- Када је корисно корситити **finally** блок? Илустровати примером.
- Написати Јава програм којим се са стандардног улаза уносе ниске све док се не унесе празна ниска. За сваку унесену ниску на екрану исписати подниску састављену од карактера који почињу на позицији 3 и дужине су 5, у којем су сва мала слова претворена у одговарајућа велика слова.
- Шта се подразумева под прослеђивањем изузетка,
  а шта под избацивањем изузетка?
- Решити проблем дат у примеру 8, с тим што се издвајање подниске и претварање слова одвија у посебној функцији чији су аргументи позиција индекса од којег почиње подниска и дужина подниске.
- Када се користи кључна реч **throw**, а када кључна реч **throws**?

ПИТАЊА И ЗАДАЦИ (3)

- Истражити класу **ArithmeticException**, који методи се налазе у тој класи? Примером илустровати употребу неких метода класе **ArithmeticException.**
- Написати Јава програм који демонстрира множење два дводимензионална низа целих бројева. Уколико димензије низова који се множе не одговарају правилима за множење, потребно је избацити нови тип изузетка.
- Које су најважније препоруке за коришћење изузетака?
- Шта су тврдње, како се реализују и када се користе?
- Упоредити и примером илустровати ситуације када се за контролу грешака у току рада програма користе изузеци, а када тврдње.