Crystal Talent Solutions

Leksioni 4

Exception handling with try-throw-catch-finaly constructs. The exceptions class.

Trajtimi i përjashtimeve është një nga veçoritë më të rëndësishme të programimit në java që na lejon të trajtojmë gabimet e ekzekutimit të shkaktuara nga përjashtimet. Në këtë leksion, ne do të mësojmë se:1. çfarë është një përjashtim 2. llojet e tij 3. klasat e përjashtimeve dhe 4. si të trajtojmë përjashtimet në java me shembuj.

1. Cfarë është nje perjashtim (exception)

Një përjashtim (exception) është një ngjarje e padëshiruar që ndërpret rrjedhën normale të programit. Kur ndodh një përjashtim, ekzekutimi i programit përfundon. Në raste të tilla ne marrim një mesazh gabimi të gjeneruar nga sistemi. E mira e përjashtimeve është se ato mund të trajtohen në Java. Duke trajtuar përjashtimet, ne mund t'i sigurojmë një mesazh kuptimplotë përdoruesit në lidhje me këtë çështje sesa një mesazh të krijuar nga sistemi, i cili mund të mos jetë i kuptueshëm për një përdorues.

2. Pse ndodh një përjashtim?

Mund të ketë disa arsye që mund të shkaktojnë përjashtimin e një programi. Për shembull: Hapja e një skedari jo-ekzistues në programin tuaj, problemi i lidhjes në rrjet, të dhëna të dobëta të cilat jepen nga përdoruesi etj. Kur një përjashtim nuk është trajtuar nga programuesi, bën që ekzekutimi i programit të përfundojë dhe afishon një mesazh gabimi i gjeneruar nga sistemi i cili tregohet përdoruesit. Për shembull le të shohim përjashtimin e gjeneruar të sistemit më poshtë:
Një përjashtim i gjeneruar nga sistemi jepet më poshtë:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero at
ExceptionDemo.main(ExceptionDemo.java:5)
 ExceptionDemo : The class name
 main : The method name
 ExceptionDemo.java : The filename
 java:5 : Line number
```

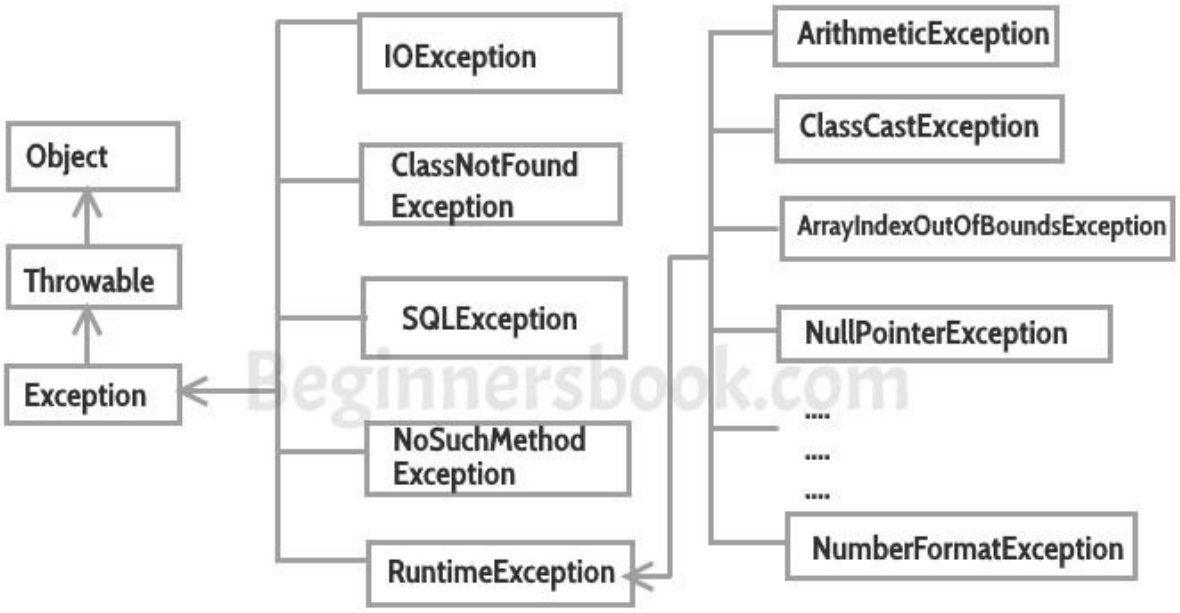
Ky mesazh nuk është miqësor për përdoruesit, kështu që një përdorues nuk do të jetë në gjendje të kuptojë se çfarë shkoi keq. Për t'i bërë të ditur arsyen dhe për tu kuptuar mesazhi që afishohet, ne trajtojmë përjashtimet dhe për më teper i perdorim gjate ekzekutimeve. Ne trajtojmë kushte të tilla dhe më pas shtypim një mesazh paralajmërues dhe të kuptueshëm për përdoruesit, i cili i lejon ata të korrigjojnë gabimin pasi në shumicën e rasteve ndodh përjashtimi për shkak të të dhënave të këqija të cilat jepen nga përdoruesi.

3. Përparësia e trajtimit të përjashtimeve

Trajtimi i përjashtimeve siguron që rrjedha e programit të mos prishet kur ndodh një përjashtim. Për shembull, nëse një program ka një mori deklaratash dhe një përjashtim ndodh në mes të rrugës pas ekzekutimit të deklaratave të caktuara, deklaratat pas përjashtimit nuk do të ekzekutohen dhe programi do të përfundojë papritmas.
Duke përdorur këto, sigurohemi që të gjitha deklaratat të ekzekutohen dhe rrjedha e programit të mos prishet.

4. Dallimi midis gabimit dhe përjashtimit

Gabimet tregojnë se diçka në kod ka shkuar keq, aplikacioni duhet të prishet në vend që të përpiqet të trajtojë gabimin.
Përjashtimet janë ngjarje që ndodhin në kod. Një programues mund të trajtojë kushte të tilla dhe të ndërmarrë veprimet e nevojshme korrigjuese. Pak shembuj:
NullPointerException - Kur përpiqeni të përdorni një referencë që tregon nul.
Përjashtimi aritmetik - Kur të dhënat e këqija sigurohen nga përdoruesi, për shembull, kur përpiqesh të pjestosh një numër me zero, ky përjashtim ndodh sepse pjesëtimi i një numri me zero është i papërcaktuar.
ArrayIndexOutOfBoundsException - Kur përpiqeni të përdorni elementet e një grupi jashtë kufijve të saj, për shembull madhësia e grupit është 5 (që do të thotë se ka pesë elemente) dhe ju po përpiqeni të përdorni elementin e 10-të.



5. Llojet e përjashtimeve

Ekzistojnë dy lloje të përjashtimeve në Java:
1) Përjashtime të kontrolluara
2) Përjashtime të pakontrolluara

Përjashtime të kontrolluara
Të gjitha përjashtimet përveç Runtime Exceptions njihen si përjashtime të kontrolluara pasi kompiluesi i kontrollon ato gjatë kompilimit për të parë nëse programuesi i ka trajtuar ato apo jo. Nëse këto përjashtime nuk trajtohen / deklarohen në program, do të merrni gabim përpilimi. Për shembull, SQLException, IOException, ClassNotFoundException etj.

Përjashtime të pakontrolluara
Runtime Exceptions njihen gjithashtu si Unchecked Exceptions. Këto përjashtime nuk kontrollohen në kohën e kompilimitt, kështu që kompiluesi nuk kontrollon nëse programuesi i ka trajtuar ato apo jo, por është përgjegjësia e programuesit të trajtojë këto përjashtime dhe të sigurojë

një afishim të sigurt. Për shembull, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etj.

Kompajleri nuk do t'ju detyrojë kurrë të ndërtni një përjashtim të tillë ose t'ju detyrojë ta deklaroni në metodën duke përdorur ndonjë fjalën kyçe.

4.1 Try Catch in Java – Exception handling

Në këtë pjesë do të shohim bllokun e provave të kapura që përdoret për trajtimin e përjashtimeve.

4.1.1 Provo bllokm (Try block)

Blloku try përmban një grup deklaratash ku mund të ndodhë një përjashtim. Një try block ndiqet gjithmonë nga një catch block, i cili merret me përjashtimin që ndodh në bllokun provues të shoqëruar. Një try block duhet të pasohet nga catch block ose në finally block ose të dyja.

Sintaksa e bllokut të provës

```
try{
  //statements that may cause an exception
}
```

Ndërsa shkruani një program, nëse mendoni se deklarata të caktuara në një program mund të sjellin një përjashtim, mbylli ato në bllokun e provës dhe trajto tek ai i përjashtimit.

4.1.2 Catch block

Një catch block është vendi ku ju trajtoni përjashtimet, ky bllok duhet të ndjekë bllokun e provës (try block) . Një bllok provë i vetëm mund të ketë disa blloqe kapëse të lidhura me të. Ju mund të kapni përjashtime të ndryshme në blloqe të ndryshme kapëse (catch block). Kur një përjashtim ndodh në bllokun e provës, blloku përkatës i kapjes që merret me atë përjashtim të veçantë ekzekuton. Për shembull nëse një përjashtim aritmetik ndodh në bllokun try atëherë ekzekutojnë deklaratat e mbyllura në bllokun e kapjes për përjashtim aritmetik.

```
try
{
   //statements that may cause an exception
}
catch (exception(type) e(object))
{
   //error handling code
}
```

Shembull: provoni catch block.

Nëse ndodh një përjashtim në bllokun e provës, atëherë kontrolli i ekzekutimit kalon në bllokun përkatës të kapjes. Një bllok provë i vetëm mund të ketë bllokime të shumta kapëse të shoqëruara me të, duhet t'i vendosni bllokimet e kapjeve në një mënyrë të tillë që blloku i kapjes së mbajtësve të përjashtimeve gjenerike të jetë i fundit (shih shembullin më poshtë). Përgjegjësi i përjashtimeve përgjithësuese mund të trajtojë të gjitha përjashtimet, por duhet të vendosni në fund, nëse e vendosni në të gjitha blloqet kapëse, atëherë ai do të shfaqë mesazhin gjenerik. Ju gjithmonë dëshironi t'i jepni përdoruesit një mesazh kuptimplotë për secilin lloj të përjashtimit, në vend të një mesazhi të përgjithshëm.

```
class Example1 {
  public static void main(String args[]) {
    int num1, num2;
    try {
      /* We suspect that this block of statement can throw
       * exception so we handled it by placing these statements
       * inside try and handled the exception in catch block
       */
      num1 = 0;
      num2 = 62 / num1;
      System.out.println(num2);
      System.out.println("Hey I'm at the end of try block");
    }
    catch (ArithmeticException e) {
      /* This block will only execute if any Arithmetic exception
       * occurs in try block
       */
      System.out.println("You should not divide a number by zero");
    }
    catch (Exception e) {
      /* This is a generic Exception handler which means it can handle
       * all the exceptions. This will execute if the exception is not
       * handled by previous catch blocks.
       */
      System.out.println("Exception occurred");
    }
    System.out.println("I'm out of try-catch block in Java.");
  }
}
```

Output:

```
You should not divide a number by zero
I'm out of try-catch block in Java.
```

4.1.3. Blloqe të shumta kapëse në Java

Shembulli që kemi parë më sipër është të kesh blloqe të shumta kapëse, le të shohim disa rregulla në lidhje me bllokime të shumta kapëse me ndihmën e shembujve. Për ta lexuar këtë në detaje, shihni kapjen e përjashtimeve të shumta në java.

1. Siç e përmenda më lart, një bllok provë i vetëm mund të ketë ndonjë numër blloku kapës.
2. Një bllok i përgjithshëm kapës mund të trajtojë të gjitha përjashtimet. Pavarësisht nëse është ArrayIndexOutOfBoundsException ose ArithmeticException ose NullPointerException ose ndonjë lloj tjetër i përjashtimit, kjo i trajton të gjitha. Për të parë shembujt e NullPointerException dhe ArrayIndexOutOfBoundsException, referojuni :

```
catch(Exception e){
  //This catch block catches all the exceptions
```

}

Nëse po pyesni pse na duhen mbajtës të tjerë kapësish kur kemi një bllok të përgjithshëm që mund të trajtojë të gjitha. Kjo sepse në mbajtësin e përjashtimeve gjenerike mund të shfaqësh një mesazh por nuk je i sigurt për llojin e përjashtimit që mund të shkaktojë, kështu që do të shfaqë të njëjtin mesazh për të gjitha përjashtimet dhe përdoruesi mund të mos jetë në gjendje të kuptojë se cili përjashtim ka ndodhur. Kjo është arsyeja që duhet të vendosni është në fund të të gjitha blloqeve një përjashtim specifik.

3. Nëse nuk ka përjashtim në bllokun e provës, blloqet e kapjes injorohen plotësisht.

4. Blloqet korresponduese të kapjes ekzekutojnë për atë lloj specifik të përjashtimit: catch (ArithmeticException e) është një bllok kapës që mund të mbajë ArithmeticException catch (NullPointerException e) është një bllok kapës që mund të trajtojë NullPointerException

5. Ju gjithashtu mund të ndertoni throw catch, i cili është një temë e përparuar e cila do të trajtohet më poshtë.

Example of Multiple catch blocks

```
class Example2{
  public static void main(String args[]){
    try{
        int a[]=new int[7];
        a[4]=30/0;
        System.out.println("First print statement in try block");
    }
    catch(ArithmeticException e){
      System.out.println("Warning: ArithmeticException");
    }
    catch(ArrayIndexOutOfBoundsException e){
      System.out.println("Warning: ArrayIndexOutOfBoundsException");
    }
    catch(Exception e){
      System.out.println("Warning: Some Other exception");
    }
  System.out.println("Out of try-catch block...");
 }
}
```

Output:

```
Warning: ArithmeticException
Out of try-catch block...
```

Në shembullin e mësipërm ka blloqe të shumëfishta kapëse dhe këto blloqe kapëse ekzekutohen në mënyrë sekuenciale kur një përjashtim ndodh në bllokun e provës. Që do të thotë nëse vendosni bllokun e fundit të kapjes (catch(Exception e)) në vendin e parë, menjëherë pasi të keni provuar

bllokun atëherë në rast të ndonjë përjashtimi ky bllok do të ekzekutohet pasi mund të trajtojë të gjitha përjashtimet. Ky bllok kapës duhet të vendoset më në fund për të shmangur situata të tilla.

4.1.4 Finally block

Tani për tani ju vetëm duhet të dini se ky bllok ekzekuton nëse ndodh një përjashtim apo jo. Ju duhet t'i vendosni ato deklarata në blloqet e fundit, të cilat duhet të ekzekutojnë nëse ndodh përjashtim ose jo.

4.2 Kontrolli i rrjedhës në blloqet try-catch-në fund

Në këtë manual do të shohim se si të përdorim blloqet try-catch-në fund për trajtimin e përjashtimeve. Le të shohim bllokun try-catch së pari dhe pastaj do të shohim se si të punojmë me try-catch-në fund.

4.2.1 Rrjedha e kontrollit në blloqet e provës / catch block:

Kur përjashtimi nuk ndodh:

Kur thëniet që janë të pranishme në bllokun try nuk bëjnë ndonjë përjashtim atëherë së pari, trupi i bllokut try ekzekutohet dhe pastaj kodi pas catch block. Në këtë rast blloku i kapjes nuk ekzekutohet kurrë pasi ato kanë për qëllim të ekzekutohen kur ndodh një përjashtim. Për shembull-

```
class Example1
{
  public static void main(String args[])
  {
            int x = 10;
            int y = 10;
            try{
              int num= x/y;
              System.out.println("next-statement: Inside try block");
            }
      catch(Exception ex)
            {
              System.out.println("Exception");
            }
            System.out.println("next-statement: Outside of try-catch");
  }
}
```

Output:

```
next-statement: Inside try block
next-statement: Outside of try-catch
```

Në shembullin e mësipërm, përjashtimi nuk ndodhi në try block, kështu që catch block nuk funksionoi.

Kur ndodh përjashtimi?

Së pari hidhni një vështrim në shembullin më poshtë dhe pastaj ne do të diskutojmë rezultatin:

```
class Example1
{
  public static void main(String args[])
  {
            int x = 0;
            int y = 10;
            try{
              int num= y/x;
              System.out.println("next-statement: Inside try block");
            }
      catch(Exception ex)
            {
              System.out.println("Exception Occurred");
            }
            System.out.println("next-statement: Outside of try-catch");
  }
}
```

Output:

```
Exception Occurred
next-statement: Outside of try-catch
```

Shënoni në shembullin e mësipërm: Ka dy deklarata të pranishme brenda bllokut të provës. Meqenëse përjashtimi ndodhi për shkak të deklaratës së parë, deklarata e dytë nuk ekzekutoi. Prandaj mund të konkludojmë se nëse ndodh një përjashtim, pjesa tjetër e try block nuk ekzekuton dhe kontrollet kalojnë për të kapur bllokun.

4.2.2 Rrjedha e kontrollit në blloqet try / catch / finally block:

1. Nëse ndodh përjashtim në trupin e bllokut të provës, atëherë kontrolli transferohet menjëherë (duke kapërcyer pjesën tjetër të deklaratave në bllokun e provës) në bllokun e kapjes. Sapo bllokimi i kapjes të përfundojë ekzekutimin, atëherë në fund bllokoni pjesën tjetër të programit.
2. Nëse nuk ka ndonjë përjashtim të ndodhur në kodin që është i pranishëm në bllokun e provës atëherë së pari, blloku i provës ekzekutohet plotësisht dhe pastaj kontrolli transferohet në bllokun e fundit (duke kapërcyer bllokimet e kapjes).
3. Nëse një deklaratë kthimi haset ose në bllokun e provës ose kapjes, në këtë rast më në fund ekzekutohet blloku. Kontrolloni kalimet e para për në fund dhe pastaj ajo u kthye përsëri në deklaratën e kthimit.

Le ta shohim këtë shembull për të kuptuar pikat e lartpërmendura:

```
class TestExceptions {
  static void myMethod(int testnum) throws Exception {
    System.out.println ("start - myMethod");
    if (testnum == 12)
```

```
        throw new Exception();
      System.out.println("end - myMethod");
      return;
    }
    public static void main(String  args[]) {
      int testnum = 12;
      try {
        System.out.println("try - first statement");
        myMethod(testnum);
        System.out.println("try - last statement");
      }
      catch ( Exception ex) {
        System.out.println("An Exception");
      }
      finally {
        System. out. println( "finally") ;
      }
      System.out.println("Out of try/catch/finally - statement");
    }
}
```

Output:

```
try - first statement
start - myMethod
An Exception
finally
Out of try/catch/finally – statement
```

4.3 Si të hedhim exceptions në java me shembull

Në Java kemi përcaktuar tashmë klasat e përjashtimeve si ArithmeticException, NullPointerException, ArrayIndexOutOfBounds etj. Këto përjashtime janë vendosur të shkaktojnë 2 kushte të ndryshme. Për shembull, kur ndajmë një numër me zero, kjo shkakton ArithmeticException, kur përpiqemi të aksesojmë elementin e grupit jashtë kufijve të tij, atëherë do të marrim ArrayIndexOutOfBoundsException.

Ne mund të përcaktojmë grupin tonë të kushteve ose rregullave dhe të hedhim një përjashtim në mënyrë të qartë duke përdorur fjalën kyçe throw. Për shembull, ne mund të hedhim ArithmeticException kur ndajmë numrin me 5, ose ndonjë numër tjetër, ajo që duhet të bëjmë është thjesht të vendosim kushtin dhe të hedhim ndonjë përjashtim duke përdorur fjalën kyçe të hedhjes. Fjala kyçe e hudhjes mund të përdoret gjithashtu për hedhjen e përjashtimeve të personalizuara,

Sintaksa e fjalës kyçe të hedhjes është si më poshtë:

```
throw new exception_class("error message");
```

për shembull:

```
throw new ArithmeticException("dividing a number by 5 is not allowed in this program");
```

4.3.1 Shembull i throw keyword

Le të themi se kemi një kërkesë ku duhet të regjistrojmë studentët vetëm kur mosha e tyre është jo më shumë se 12 dhe pesha është më pak se 40, nëse ndonjë nga kushtet nuk plotësohet, atëherë përdoruesi duhet të marrë një Përjashtim të Aritmetikës me mesazhin paralajmërues "Student nuk është i përshtatshëm për regjistrim ". Ne kemi zbatuar logjikën duke vendosur kodin në metodën që kontrollon pranueshmërinë e studentit nëse mosha dhe pesha e futur e studentit nuk i plotëson kriteret, atëherë ne hedhim përjashtimin duke përdorur fjalën kyçe hedh (throw).

```
/* In this program we are checking the Student age
 * if the student age<12 and weight <40 then our program
 * should return that the student is not eligible for registration.
 */
public class ThrowExample {
  static void checkEligibilty(int stuage, int stuweight){
    if(stuage<12 && stuweight<40) {
      throw new ArithmeticException("Student is not eligible for registration");
    }
    else {
      System.out.println("Student Entry is Valid!!");
    }
  }

  public static void main(String args[]){
    System.out.println("Welcome to the Registration process!!");
    checkEligibilty(10, 39);
    System.out.println("Have a nice day..");
 }
}
```

Output:

```
Welcome to the Registration process!!Exception in thread "main"
java.lang.ArithmeticException: Student is not eligible for registration
at beginnersbook.com.ThrowExample.checkEligibilty(ThrowExample.java:9)
at beginnersbook.com.ThrowExample.main(ThrowExample.java:18)
```

Në shembullin e mësipërm kemi hedhur një përjashtim të pakontrolluar, në të njëjtën mënyrë mund të hedhim gjithashtu një përjashtim të pakontrolluar dhe të përcaktuar nga përdoruesi.

4.4 Trajtimi i perjashtimit

Në këtë pjesëne do të shohim shembuj të disa përjashtimeve që përdoren shpesh.

Shembulli 1: Përjashtim aritmetik Klasa: Java.lang. AritmetikaPërjashtimi Kjo është një klasë e integruar e pranishme në paketën java.lang. Ky përjashtim ndodh kur një numër i plotë ndahet me zero.

```
class Example1
{
```

```
  public static void main(String args[])
  {
    try{
      int num1=30, num2=0;
      int output=num1/num2;
      System.out.println ("Result: "+output);
    }
    catch(ArithmeticException e){
      System.out.println ("You Shouldn't divide a number by zero");
    }
  }
}
```

Output of above program:

You Shouldn't divide a number by zero

Shpjegim: Në shembullin e mësipërm unë kam ndarë një numër të plotë me një zero dhe për shkak të kësaj ArithmeticException është hedhur.

Shembulli 2: ArrayIndexOutOfBounds Exception

Class: Java.lang.ArrayIndexOutOfBoundsException

Ky përjashtim ndodh kur përpiqeni të përdorni indeksin e grupit i cili nuk ekziston. Për shembull, Nëse array ka vetëm 5 elemente dhe ne po përpiqemi të shfaqim elementin e 7-të, atëherë do të hidhte këtë përjashtim.

```
class ExceptionDemo2
{
  public static void main(String args[])
  {
    try{
      int a[]=new int[10];
      //Array has only 10 elements
      a[11] = 9;
    }
    catch(ArrayIndexOutOfBoundsException e){
      System.out.println ("ArrayIndexOutOfBounds");
    }
  }
}
```

Output:

ArrayIndexOutOfBounds

Në shembullin e mësipërm, vargu inicializohet për të ruajtur vetëm 10 indekse elementesh 0 deri në 9. Meqenëse jemi duke u përpjekur të hyjmë në elementin e indeksit 11, programi po hedh këtë përjashtim.

Shembulli 3: NumberFormat Exception

Class: Java.lang.NumberFormatException

Ky përjashtim ndodh kur një varg analizohet në çdo ndryshore numerike. Për shembull, deklarata int num = Integer.parseInt ("XYZ"); do të hidhte NumberFormatException sepse Vargu "XYZ" nuk mund të analizohet në int.

```
class ExceptionDemo3
{
  public static void main(String args[])
  {
    try{
            int num=Integer.parseInt ("XYZ") ;
            System.out.println(num);
    }catch(NumberFormatException e){
             System.out.println("Number format exception occurred");
     }
  }
}
```

Output:
Number format exception occurred

Shembulli 4: StringIndexOutOfBound Exception

Class: Java.lang.StringIndexOutOfBoundsException

Një objekt i kësaj klase krijohet sa herë që një indeks thërritet në një varg, i cili nuk është në interval. Secili karakter i një objekti string ruhet në një indeks të veçantë duke filluar nga 0. Për të marrë një karakter të pranishëm në një indeks të veçantë të një vargu mund të përdorim një metodë charAt (int) të java.lang.String ku argumenti int është indeksi.

```
class ExceptionDemo4
{
  public static void main(String args[])
  {
    try{
            String str="beginnersbook";
            System.out.println(str.length());;
            char c = str.charAt(0);
            c = str.charAt(40);
            System.out.println(c);
    }catch(StringIndexOutOfBoundsException e){
             System.out.println("StringIndexOutOfBoundsException!!");
     }
  }
}
```

Output:
13
StringIndexOutOfBoundsException!!
Përjashtim ndodhi sepse indeksi i referuar nuk ishte i pranishëm në Varg.

Shembulli 5: NullPointer Exception

Class: Java.lang.NullPointer Exception

Një objekt i kësaj klase krijohet sa herë që një anëtar thërritet me një objekt "null".

```
class Exception2
{
  public static void main(String args[])
  {
          try{
                  String str=null;
                  System.out.println (str.length());
          }
     catch(NullPointerException e){
                  System.out.println("NullPointerException..");
          }
  }
}
```

Output:

NullPointerException..
Këtu, gjatësia () është funksioni, i cili duhet të përdoret në një objekt. Sidoqoftë, në shembullin e mësipërm String object str është nul kështu që nuk është objekt për shkak të të cilit ka ndodhur NullPointerException.

Shtojcë:

#2) ArrayIndexOutOfBoundsException: ArrayIndexOutOfBoundsException is thrown when an array element is accessed using an illegal index. The index used is either beyond the size of the array or is negative.
#3) ClassNotFoundException: If the class definition is not found then the ClassNotFoundException is raised.
#4) FileNotFoundException: FileNotFoundException is given when the file does not exist or does not open.
#5) IOException: IOException is thrown when the input-output operation fails or is interrupted.
#6) InterruptedException: Whenever a thread is doing processing or sleeping or waiting, then it is interrupted by throwing InterruptedException.
#7) NoSuchFieldException: If a class does not contain a specified field or variable, then it throws NoSuchFieldException.
#8) NoSuchMethodException: When the method being accessed is not found, then NoSuchMethodException is raised.

#9) NullPointerException: NullPointerException is raised when a null object is referred. This is the most important and most common exception in Java.
#10) NumberFormatException: This exception is raised when a method could not convert a string into a numeric format.
#11) RuntimeException: Any exception that occurs at runtime is a RuntimeException.
#12) StringIndexOutOfBoundsException: The StringIndexOutOfBoundsException is thrown by String class and indicates that the index is beyond the size of the String object or is negative.
#13) EOFException: EOFException is a part of the java.io package and is thrown when the end of file is reached and the file is being read.
#14) IllegalArgumentException: IllegalArgumentException is thrown when illegal or invalid arguments are passed to the method. **For example,** the wrong data format, null value when non-null is required or out of range arguments.