Лекция 11b

Обработка на изключения

Основни теми

- **Обработка на изключения.**
- **Класът *Exception*.**
- ***Try* блок и приложението му.**
- **Прихващане и предаване на изключение.**
- **Потребителски дефинирани изключения с използване на наследственост.**

11b.1 Въведение

Изключение (*exception*) – указва за проблем, възникнал извън очакваната логика на изпълнение на програмата, при което програмата абортира изпълнение

Названието "*exception*“ (изключение) подсказва за проблем, който възниква рядко, извън “*правилото*" за очакваното изпълнение на програмата.

Например, в процеса на изпълнение се изчерпва наличната памет или друг изчислителен ресурс е недостъпен

11b.1 Въведение

Обработка на изключения– прихващане на изключенията и дефиниране на команди, позволяващи продължаване на работата или осигуряване на нормален изход на програмата

Позволява писане на програми, устойчиви на грешки

Изключението се описва като Java клас и се подчинява на същите правила, както и всеки друг клас

11b.1 Въведение

Примери

- `ArrayIndexOutOfBoundsException` – изключение породено при опит за достъп на елемент от масив извън дефинирания размер на масива
- `ClassCastException` – изключение породено при опит за преобразяване на тип на обект, който не е производен на типа, указан с оператора на оператора за явно преобразуване
- `NullPointerException` – изключение породено при използване на `null` референция, вместо очаквана референция към обект

11b.2 Обработка на изключения-обзор

Програмите често използват условен преход за управление на логиката на програмата

```
Изпълни стъпка N
If има грешка
    Perform error processing

Изпълни стъпка N + 1
If има грешка
    Perform error processing
…
```

11b.2 Обработка на изключения-обзор

Смесване на команди за изпълнение на логиката на програмата с команди за обработка на грешки води до програми трудни за четене, промяна, поддръжка и проверка (*дебъг*)

Обработката на изключения позволява командите за обработка на грешки да се извадят от основния контекст на програмата

Подобрява се яснотата на програмиране

Подобрява се възможността на реагиране на грешки на потребителя.

Съвет за по- добро качество

Когато потенциален проблем възниква рядко, смесването на програмен код с код за обработка на грешки може да влоши качеството на изпълнение на програмата, поради необходимост от чести проверки за коректното изпълнение на програмата.

11b.3 Пример: Divide By Zero без изключения

Демонстрираме възникване на изключение при липса на код за тяхната обработка

- **`Figure 11b.1` извежда промпт на потребителя да въведе `2` `цели числа`, които се предават като аргументи на метод `quotient` на частното от тези числа и връща `int` резултат**
- **В този пример се вижда как се "`хвърля`" изключение (`throw`) при откриване на ситуация, която програмата не може да реши с посредством съществуващия код – *целочислено деление на нула***

11b.3 Пример: Divide By Zero без изключения

При *хвърляне* на изключение, което не се обработва, програмата абортира изпълнението с извеждане на стандартно съобщение, указващо името на изключението - описание на "следата на стек-а"

Stack trace "*следата на стек-а*" включва

- **Името на изключението (клас)**
- **Пълен списък на извиканите методи, породили хвърлянето на изключението ("*call chain*")**

Помага за намирането на грешка

11b.3 Пример: Divide By Zero без изключения

`ArithmeticException` – **поражда се от различни грешки при аритметични операции, например,** ***целочислено делене на нула***

Място за хвърляне на изключение– най- горния ред във ***веригата от извиквания (chain call)*** **при разпечатване на следата на стек-а** ***(stack trace)***

`InputMismatchException` – **възниква ,например, когато методът** `nextInt` **на клас** `Scanner` **прочита низ, който не се свежда до цяло число**

Резюме

Изключение при целочислено делене

```
1  // Fig. 12a.1: DivideByZeroNoExceptionHandling.java
2  // An application that attempts to divide by zero.
3  import java.util.Scanner;
4
5  public class DivideByZeroNoExceptionHandling
6  {
7     // demonstrates throwing an exception when a divide-by-zero occurs
8     public static int quotient( int numerator, int denominator )
9     {
10       return numerator / denominator; // possible division by zero
11    } // end method quotient
12
13    public static void main( String args[] )
14    {
15       Scanner scanner = new Scanner( System.in ); // scanner for input
16
17       System.out.print( "Please enter an integer numerator: " );
18       int numerator = scanner.nextInt();
19       System.out.print( "Please enter an integer denominator: " );
20       int denominator = scanner.nextInt();
21
22       int result = quotient( numerator, denominator );
23       System.out.printf(
24          "\nResult: %d / %d = %d\n", numerator, denominator, result );
25    } // end main
26 } // end class DivideByZeroNoExceptionHandling
```

Опит за делене; denominator **може да е нула**

Read input; **exception occurs if input is not a valid integer**

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

Резюме

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at
DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoExceptionHandling.java:10)
        at
DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:22)
```

Примери за

Arithmetic exception

Означено е целочислено делене с нула

/by zero

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello
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
DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:20)
```

InputMismatchException

Unknown source

JVM няма достъп до кода, където е хвърлено изключение

11b.4 Пример: Обработка на изключения `ArithmeticException` и `InputMismatchException`

При обработка на изкючения, изключенията се прихващат и се дефинира действия за преодоляването им

В следващия пример се реализира прихващане на грешките, демонстрирани в фигура 11b.1 (*нула за знаменател, или неправилно въведено цяло число*)

Заграждане на команди в `try` блок

`try` блок– обхваща команди, които може да хвърлят изключение и команди, които не трябва да се изпълнят при наличие на изключение

Съставен е от ключовата дума `try` следвана от блок от код , означен с фигурни скоби {*отваряща ... затваряща*}

Software Engineering факти

Изключенията могат да се проявят в `try` блок, при извикване на вложени методи в `try` block или чрез Java Virtual Machine при изпълнението на компилиран Java байткод(class file).

Прихващане на изключения

- `catch` блок – прихваща(т.е., получава) и обработва изключение, включва:
  - Започва с ключовата дума `catch`
  - Име на клас от тип изключение за аргумент– указва какъв тип изключение се прихваща и обработва вв `catch` блока
  - Блок от код който се изпълнява при прихващане на указаното като аргумент изключение

Съпоставяне с аргумента на `catch` блока– типът на прихванатото изключение да е точно от типа на аргумента на `catch` блока или да е производен клас

След всеки try блок може да има един или повече `catch` блока

"*изпуснато*" изключение – "*хвърлено*" изключение, несъпоставимо с аргумент на никой от `catch` блоковете

- Прекратява текущата нишка в изпълнението на програмата

Обичайна грешка при програмиране

Синтактична грешка е да се поставят изпълними команди между `try` блока и съответните му `catch` блокове.

Обичайна грешка при програмиране

Всеки `catch` блок има единствен аргумент - задаване на списък от аргументи е синтактична грешка.

Пълен модел за обработка на изключения

При възникване на изключение:

- **Изпълнението на `try` блока приключва незабавно**
- **Управлението се предава на първия `catch` блок**
- **Ако името на изключението не се съпоставя с аргумента на първия `catch` блок се прави съпоставка с името на следващите `catch` блокове**
- **При успешно съпоставяне на името на изключението се изпълнява блокът от команди на съответния `catch` блок – изключението се обработва**
- **Когато никой `catch` блок не води до успешно съпоставяне с името на изключението посочено като аргумент на блока, програмата се абортира (прекратява)**

Пълен модел за обработка на изключения

След обработката на изключението:

- **Модел на завършена обработка- Управлението на логиката на изпълнение не се връща на мястото "хвърлило" изключението, понеже `try` блокът е недостъпен след излизането от него; Управлението се предава на първата команда след `catch` блока**
- **Модел на възстановена обработка- Управлението се предава на първата команда след тази "хвърлила" изключението**

Моделът на възстановена обработка **води до логически грешки и не се използва в Java**

- **`try` команда– състои се от `try` блок,съответни `catch` блокове и (или) `finally` блок**

Съвет за програмиране11b.2

При обработка на изключения програмата може да продължи изпълнение (*вместо да завърши*), след обработка на изключението ..

Съвет за добро програмиране

Използване на подходящо име за аргумент на изключение в `catch` блок допринася за по- добро четене и подръжка на програмата.

Използване на `throws` описание

`throws` описание- задава списък с всички изключенията, които даден метод може да хвърля за обработка

- **Изписва се след списъка с аргументи и преди тялото на метода**
- **Списъкът съдържа имена на изключения, разделени със запетая**
- **Изключенията в този списък може да се "*хвърлят*" от команди от тялото на метода или други методи, извиквани от тялото на метод**
- **Изключенията "*хвърляни*" от даден метод могат да са само списъка на `throws` описанието или техни производни класове**

Съвет за добро програмиране

Прочетете в API документацията дали даден метод хвърля и какви изключения. Това позволява да реализирате подходяща обработка на тези изключения във програмата си.

Резюме

```
1  // Fig. 12a.2: DivideByZeroWithExceptionHandling.java
2  // An exception-handling example that checks for divide-by-zero.
3  import java.util.InputMismatchException;
4  import java.util.Scanner;
5
6  public class DivideByZeroWithExceptionHandling
7  {
8     // demonstrates throwing an exception when a divide-by-zero occurs
9     public static int quotient( int numerator, int denominator )
10       throws ArithmeticException
11    {
12       return numerator / denominator; // possible division by zero
13    } // end method quotient
14
15    public static void main( String args[] )
16    {
17       Scanner scanner = new Scanner( System.in ); // scanner for input
18       boolean continueLoop = true; // determines if more input is needed
19
20       do
21       {
22          try // read two numbers and calculate quotient
23          {
24             System.out.print( "Please enter an integer numerator: " );
25             int numerator = scanner.nextInt();
26             System.out.print( "Please enter an integer denominator: " );
27             int denominator = scanner.nextInt();
28
```

`throws` описание на метод `quotient` , че този мето хвърля `ArithmeticException`

Цикълът се изпълнява, докато `try` блокът не завърши успешно

`try` блок

Четене на входни данни; `InputMismatchException` се хвърля при грешка в данните

Резюме

```
29            int result = quotient( numerator, denominator );
30            System.out.printf( "\nResult: %d / %d = %d\n", numerator,
31               denominator, result );
32            continueLoop = false; // input successful; end looping
33         } // end try
34         catch ( InputMismatchException inputMismatchException )
35         {
36            System.err.printf( "\nException: %s\n",
37               inputMismatchException );
38            scanner.nextLine(); // discard input so user can try again
39            System.out.println(
40               "You must enter integers. Please try again.\n" );
41         } // end catch
42         catch ( ArithmeticException arithmeticException )
43         {
44            System.err.printf( "\nException: %s\n", arithmeticException );
45            System.out.println(
46               "Zero is an invalid denominator. Please try again.\n" );
47         } // end catch
48      } while ( continueLoop ); // end do...while
49   } // end main
50 } // end class DivideByZeroWithExceptionHandling
```

При достигане до тук `denominator` е ненулев и пресмятането е успешно

Изключения- аргументи

Игнорира останалия вход

Извежда съобщение за грешка

Прихваща `ArithmeticException` (въведена е нула за знаменателя)

Ако ред 32 не се изпълни, цикълът се повтаря и потребителят наново въвежда данните

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Exception: java.lang.ArithmeticException: / by zero
Zero is an invalid denominator. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

Резюме

Примерна за обработка на изключения

Извежда се описание на изключението и изпълнението продължава с цел поправяне на грешката

11b.5 Кога да се използва обработка на изключения

Използва се за обработка на <u>синхронни грешки</u> (*грешни индекси на елементи на масив, препълване на паметта при целочислено делене, грешен тип на аргументи на метод и пр.*)

- **Синхронни грешки– възникват при изпълнение на команда**
- **Асинхронни грешки– възникват паралелно и независимо от изпълнението на програмата (*работа с мишка, клавиатура и пр.*)**

Software Engineering факт

Избягвайте да използвате обработката на изключения като алтернатива на условен преход в управлението на логиката на програмата. Всеки try..catch блок изисква допълнителни ресурси и забавя изпълнението на програмата.

11b.6 Йерархия от Java изключения

Всички изключения произхождат от директно или индиректно от class `Exception`

Йерархията от изключения може да се използва за създаване на производни класове

class `Throwable`, е базов клас на class `Exception`

- **Единствено `Throwable` могат да се използват за обработка на грешки**
- **Има два производни класа: `Exception` и `Error`**
  - **class Exception и производните му класове описват ситуации на изключения, които могат да се прихванат и обработятс цел възстановяване на работата на програма**
  - **class Error и производните му класове описват ситуации на изключения, които възникват в JVM и програмата прекъсва изпълнение (*фатални грешки*)**

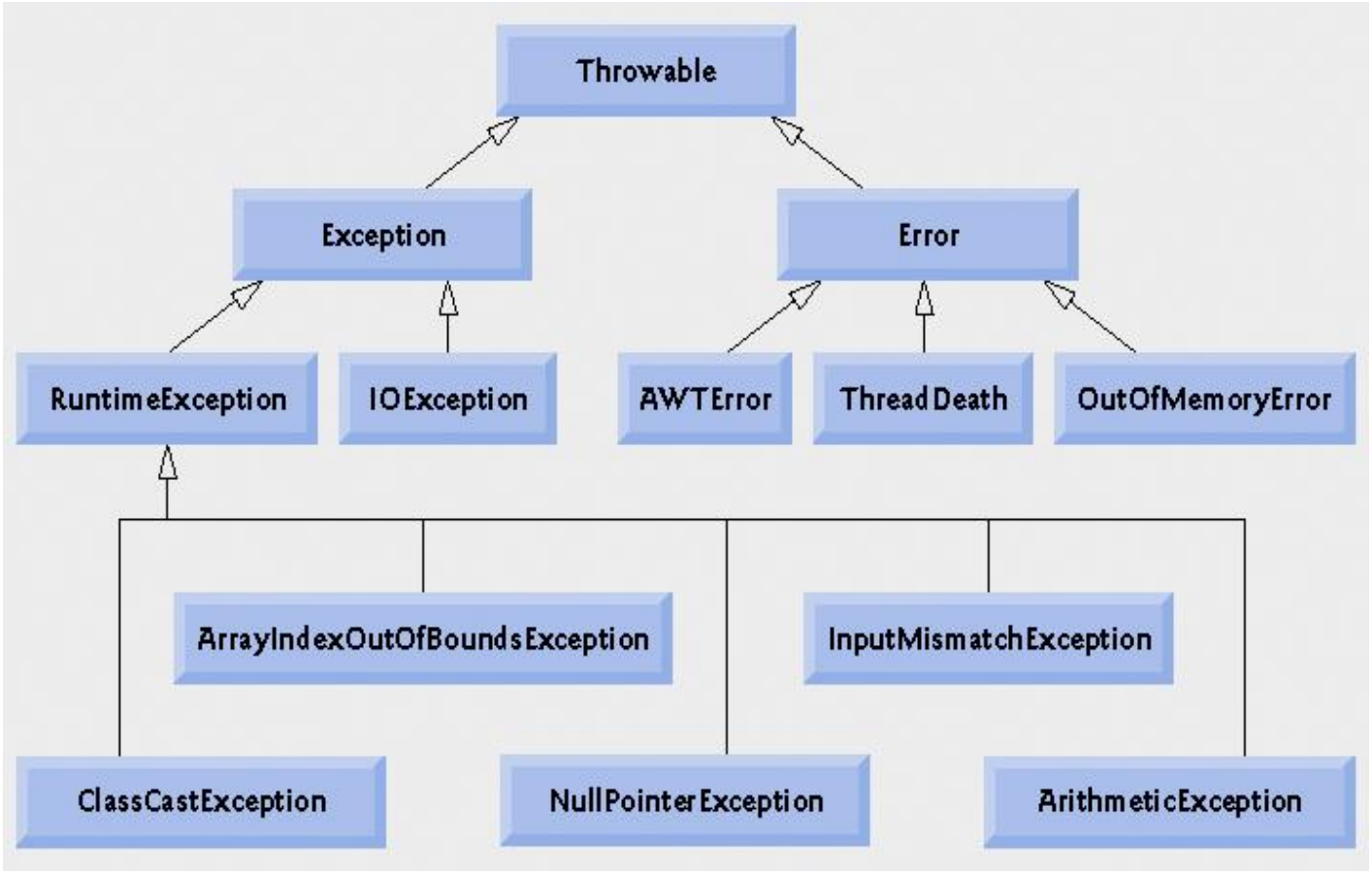

Fig. 11b.3 | Част от производните класове на *class Throwable*.

11b.6 Йерархия от Java изключения

Съществуват два типа изключения: *checked* и *unchecked*

- ***checked* изключения**
  - **Производни на class *Exception*, но не и от class *RuntimeException***
  - **Компилаторът налага "*обработи или опиши*" изключение**
  - **Компилаторът проверява при всяко извикване дали се обработва описаното изключение в списъка *throws* на този метод. Ако изключението не е обработено, то се изисква да бъде открито в списъка *throws* на викащия метод.**
- ***unchecked* изключения**
  - **Производни на m class *RuntimeException* или class *Error***
  - **Компилаторът не проверява дали изключението е описано в списъка *throws***
  - **Примери за *unchecked изключение* - *ArrayIndexOutOfBoundsException*, *ArithmeticException***
  - **Преодолява се с правилна логика на програмиране**

Software Engineering факти

Програмистите се занимават с `checked` изключения. Това води до по- добре работещи програми.

Обичайна грешка при програмиране

Хвърляне на изключение от метод, за което липсва обработка и описание `throws` в този метод води до синтактична грешка.

Обичайна грешка при програмиране

Ако метод в производен клас предефинира метод от базов клас, то се получава синтактична грешка при добавяне на повече изключения в `throws` описанието, отколкото метода има в базовия клас. Обаче, едно `throws` описание на метод в производен клас може да е подмножество на изключенията описани с `throws` списък в метода от базовия клас.

Software Engineering факт

Ако метод извиква други методи, които явно хвърлят *checked* изключения, то тези изключения трябва или да се обработят или да се опишат с *throws*.

Software Engineering факти

Дори когато компилаторът не налага "*прихвани или опиши*" изискване за *unchecked* изключения, използвайте обработка на тези изключения, когато е известно, че има възможност за появяването им . Например, програмата трябва да обработва *NumberFormatException* на метод *parseInt* от клас *Integer*, дори когато *NumberFormatException* (производен клас на *RuntimeException*) е от *unchecked* тип изключение.

11b.6 Йерархия от Java изключения

- `catch` блокът прихваща всички изключения от типа на аргумента или негови производни класове
- При наличие на няколко `catch` блока за същото изключение, само първия `catch` блок успешно съпоставил типа на изключението го обработва
- Има смисъл да се използва базов клас за аргумент на `catch` блок, когато всеки `catch` блок за производните на това изключение водят до същата обработка на изключения

Обичайна грешка при програмиране

Поставяне на `catch` блок за базов тип изключение преди `catch` блокове на производните на това изключение класове води до грешка при компилация, понеже "скрива" тези блокове за изпълнение.

11b.7 Блок от код `finally`

След използване на определени ресурси програмите трябва явно да освободят тези ресурси

- **`finally` блок**
  - **Състои се то ключовата дума `finally` следвана от блок от код**
  - **Не е задължителен в `try` командата**
  - **Поставя се след последния `catch` блок**
  - **Изпълнява се независимо дали е имало или не изключение в `try` блока или някой от `catch` блоковоте му**
  - **Не се изпълнява при преждемременно прекратяване на програмата вътре в `try` блока с метод `System.exit`**
  - **Обикновено съдържа код за освобождаване на ресурси(затваряне на файлове, връзки към бази данни или мрежи и пр.)**

Съвет за предпазване от грешки

Java не освобождава ресурсите, заемани от обект докато има референции към този обект. Така, неправилното използване на поредица от ненужни обекти също води до разхищаване ресурси от памет.

```
try
{
    statements
    resource-acquisition statements
} // end try
catch ( AKindOfException  exception1 )
{
    exception-handling statements
} // end catch
.
.
.
catch ( AnotherKindOfException  exception2 )
{
    exception-handling statements
} // end catch
finally
{
    statements
    resource-release statements
} // end finally
```

Fig. 12a.4 | finally блокът е след последния catch блок на try командата .

Резюме

11b.7 Блок от код finally

Когато няма изключения, catch блоковете се пропускат и изпълнението продължава с finally блока.

След изпълнението на finally блока изпълнението продължава с първата команда след finally блока.

Ако има изключение в try блока, програмата прекратява изпълнението на try блока . Първият catch блок с аргумент, съпоставим с типа на изключението се изпълнява и после се изпълнява finally блока. Ако няма catch блокове със съпоставими аргументи по тип на хвърленото изключение, управлението се предава на finally блока. След изпълнението на finally блока, управлението се предава на първата команда след try блока.

Ако catch блок хвърля изключение, finally блока пак се изпълнява.

Съвет за качество на програмиране

Винаги освобождавайте ресурсите явно и в първия момент, когато те не са повече нужни.

Съвет за предпазване от грешки

Понеже `finally` блокът винаге се изпълнява след `try` блок независимо дали е имало изключение, то този блок е идеалното място за освобождаване на ресурси.

11b.7 Блок от код `finally`

Стандартни потоци

- `System.out` – стандартен изход
- `System.err` – стандартна грешка

- `System.err` може да се използва за отделяне на съобщения за грешки от стандартния изход
- `System.err.println` и `System.out.println` извеждат по подразбиране на стандартен изход

Резюме

Пример за прихващане на изключения

```
1  // Fig. 12a.5: UsingExceptions.java
2  // Demonstration of the try...catch...finally exception handling
3  // mechanism.
4
5  public class UsingExceptions
6  {
7     public static void main( String args[] )
8     {
9        try
10       {
11          throwException(); // call method throwException
12       } // end try
13       catch ( Exception exception ) // exception thrown by throwException
14       {
15          System.err.println( "Exception handled in main" );
16       } // end catch
17
18       doesNotThrowException();
19    } // end main
20
```

Изпълнява метод , хвърлящ изключение

Резюме

```
21  // demonstrate try...catch...finally
22  public static void throwException() throws Exception
23  {
24     try // throw an exception and immediately catch it
25     {
26        System.out.println( "Method throwException" );
27        throw new Exception(); // generate exception
28     } // end try
29     catch ( Exception exception ) // catch exception thrown in try
30     {
31        System.err.println(
32           "Exception handled in method throwException" );
33        throw exception; // rethrow for further processing
34
35        // any code here would not be reached
36
37     } // end catch
38     finally // executes regardless of what occurs in try...catch
39     {
40        System.err.println( "Finally executed in throwException" );
41     } // end finally
42
43     // any code here would not be reached, exception rethrown in catch
44
```

Създава `Exception` обект и го "хвърля"

Повторно хвърля по- рано създаден `Exception`

`finally` блокът се изпълнява след `catch` блока

Резюме

```
45    } // end method throwException
46
47    // demonstrate finally when no exception occurs
48    public static void doesNotThrowException()
49    {
50       try // try block does not throw an exception
51       {
52          System.out.println( "Method doesNotThrowException" );
53       } // end try
54       catch ( Exception exception ) // does not execute
55       {
56          System.err.println( exception );
57       } // end catch
58       finally // executes regardless of what occurs in try...catch
59       {
60          System.err.println(
61             "Finally executed in doesNotThrowException" );
62       } // end finally
63
64       System.out.println( "End of method doesNotThrowException" );
65    } // end method doesNotThrowException
66 } // end class UsingExceptions
```

```
Method throwException
Exception handled in method throwException
Finally executed in throwException
Exception handled in main
Method doesNotThrowException
Finally executed in doesNotThrowException
End of method doesNotThrowException
```

`finally` блокът се изпълнява, дори и без да има изключение

Хвърляне на изключения с командата throw

- `throw` команда– използва се за хвърляне на изключения

Програмистите могат да хвърлят изключения при нужда да сигнализират за нещо нередно в изпълнението

- `throw` командата се състои от ключовата дума `throw` последвана от обект производен на Exception

Software Engineering факти

При изпълнение на `toString` към `Throwable` обект, се изписва низът подаден към конструктора на този обект или просто името на изключението.

Software Engineering факти

Изключения могат да се хвърлят от конструкторите. При установяване на грешка в създаване на обект е по- добре да се хвърли изключение, отколкото да се създаде неправилно формиран обект.

Повторно хвърляне на изключения

Изключения се хвърлят повторно когато `catch` блок решава, че няма достатъчно информация да възстанови работата на програмата или не може да обработи изключението

Обработката на изключението се отлага за по-горен слой от програмата

Изключение се хвърля повторно с командата `throw` следвана от референция към обекта на изключението

Обичайна грешка при програмиране

Ако изключение не е прихванато преди изпълнението на `finally` блока и `finally` блока хвърля изключение не прихванато в `finally` блока, то първото изключение се загубва , а изключението от `finally` блока се предава на викащия метод за понататъшна обработка.

Съвет за предпазване от грешки

Избягвайте хвърляне на изключение от `finally` блок . При необходимост от това вмъкнете `try` команда вътре във `finally` блока за предпазване от загуба на изключение.

11b.11 Дефиниране на потребителски изключения

Може да се създават потребителски дефинирани изключения

Производни на съществуващ клас изключение

Потребителското изключение съдържа само конструктори

- **Конструктор по подразбиране, предава текст по подразбиране на базовия конструктор**
- **Конструктор за общо ползване , предава потребителски дефинирано описание на изключението се предава н абазовия конструктор**

Software Engineering факти11b.13

При възможност използвайте предефинираните класове изключения налични в Java библиотеките.

Software Engineering факти

Новият клас на потребитески дефинирано изключение да е от групата checked изключения (т.е., *производно на Exception но не и на RuntimeException*) ако е необходима обработка на изключението.
Потребителското приложение трябва да може да възстанови работа при обработката на изключението.
Новият клас изключение може да е RuntimeException клиентския код може да игнорира изключението(*т.е., изключението е от групата на unchecked изключения*).

Съвет за добро програмиране

Прието е имената на всички потребителски дефинирани изключения да завършват на думата `Exception`.

11b.11 Дефиниране на потребителски изключения

Дефинират се три конструктора

- **По подразбиране**
- **Конструктор с аргумент String**
- **Конструктор с аргумент String и аргумент Exception**

Резюме

NegativeNumberException.java

```
1   // Fig 13:11: NegativeNumberException.java
2   // NegativeNumberException represents exceptions caused by illegal
3   // operations performed on negative numbers
4
5
6
7   // NegativeNumberException represents exceptions caused by
8   // illegal operations performed on negative numbers
9   class NegativeNumberException extends Exception
10  {
11     // default constructor
12     public NegativeNumberException()
13     {
14        super( "Illegal operation for a negative number" );
15     }
16
17     // constructor for customizing error message
18     public NegativeNumberException( String message )
19     {
20        super( message );
21     }
22
23     // constructor for customizing error message and
24     // specifying inner exception object
25     public NegativeNumberException(
26        String message, Exception inner )
27     {
28        super( message, inner )
29     }
30
31  } // end class NegativeNumberException
```

class **NegativeNumberException** производен на **Exception**

Конструктор по подразбиране

Конструктор с аргумент String

Конструктор с аргументи String и **Exception**

Задачи

Задача 1.

Напишете *JavaFX* приложение, което въвежда изминати километри и литри гориво изразходвани за пресмятане на разхода на гориво за километър. Използвайте обработка на изключението *NumberFormatException* което се хвърля при преобразуване на низове от *TextField* до *double*. При въвеждане на грешен формат в *TextField* , използвайте *диалогов прозорец* за издаване на съобщение до потребителя и повтаряне на въвеждането на данните.

Задачи

Задача 2.

Дефинирайте производен клас на *Exception* наречен *InvalidPasswordException* с два конструктора. Конструкторът по подразбиране да издава съобщение *"ERROR: invalid password"* при извикване на метода му *getMessage*(). Вторият конструктор да има *String* аргумент. Изключенията хвърляни с този конструктор да извеждат с методът му *getMessage*() аргументът , използван с този конструктор.

Напишете *JavaFX* приложение, което въвежда потребителско име и парола. При натискане на *Button* се сравняват въведените данни в *TextField* полетата с низовете “*username*” и “*password*”, съответно за потребителско име и парола. При липса на съвпадение да се хвърля *InvalidPasswordException* и да се извежда съобщение в диалогов прозорец за повтаряне на входа.