Тема 6. Параметризирани типове. Изключения.

доц. д-р инж. Владимир Николов

17.3.2020 г.

Съдържание

- Параметризирани типове ;
- Изключенията в езика Java;
  - Дефиниции - обикновено състояние и изключително състояние-изключение;
  - Действие на системата при изключение;
  - Действие на механизъма за предизвикване на изключенията;
  - Разлики между възврат от метод и предизвикване на изключение;
  - Обработчици на изключения;
  - Клауза **finally;**
  - Прихващане на всички изключения;
  - Прекратяване и продължаване;
  - Търсене на обработка;
  - Стандартни изключения-йерархия на класовете;
  - Създаване на собствени изключения;
  - Изключения при файлов достъп.

Параметризирани типове

Дефиниция:

Параметризираните типове *дават възможност от една и съща имплементация да се създава код за различни типове на аргументите (шаблонна спецификация).*

- Това са известните от езика C++ шаблонни типове.

Параметризирани типове

Базовият вариант на езика Java няма еквивалент на шаблоните в C++.

Използване на параметризираните типове:

- За създаване на функции с различни аргументи (типове на данните);
- За създаване на контейнери и алгоритми;

Параметризирани типове при създаване на функции с различни аргументи

Приложение на параметризирани типове:

- За създаване само на подобни функции с различни типове аргументи. Липсата в езика Java може да предизвика създаване на копия на един и същ код за различните аргументи;
- За автоматично генериране на класове. Позволява да се генерират различните по тип на данните класове от обща шаблонна спецификация-това е проблем на езика Java, който не може да се пренебрегне;

Параметризирани типове създаване на контейнери и алгоритми

- Друга необходимост от шаблоните е използването им в Стандартната темплейтна библиотека STL. За да могат да се използват контейнерите и алгоритмите, те се параметризират по тип на съхранявания елемент.

Параметризирани типове създаване на контейнери и алгоритми

След версия 5.0 са отстранени недостатъците от липсата на шаблони при дефиниране на контейнери и алгоритми.

```
import java.util.*;
import java.io.*;
public class mainapp {
    public static void test(Map jm) {
          System.out.println("Testing " +
          jm.getClass().getName());
    }
    public static void main(String[] args)  {
      test(new HashMap<String,Number>());
      test(new TreeMap<String,Number>());
    }
}
```

Параметризирани типове създаване на контейнери и алгоритми

Темплейтизиране на колекциите (след версия 5.0 - обозначена като 5.0+). Добавяне на шаблонна променлива E.

Примерна спецификация на колекцията List в версия 5.0+:

```
public class List< E > { ...
  public void add( E element ) { ... }
  public E get( int i ) { ... }
}
```

Параметризирани типове
създаване на контейнери и алгоритми

Използване на шаблонната променлива:

- При деклариране на променливи на инстанциите на класа;
- За аргументите на методите;
- Връщаните типове от методите.

Например: E е аргумент на метода add( ), връщан тип на метода get( ).

Параметризирани типове създаване на контейнери и алгоритми

Примери за използване:

```
List<String> listOfStrings;
List<Date> listOfDates;
List<java.math.BigDecimal> listOfDecimals;
List<Foo> foos;
```

Резултат:

```
public class List< String > { ...
   public void add( String element ) { ... }
   public String get( int i ) { ... }
}
```

Параметризирани колекции (Generics)

Например, колекция за тип Date:

```
List<Date> dates = new ArrayList<Date>(  );
dates.add( new Date(  ) );
dates.add( "foo" ) // продуцира грешка - не е Date!!
```

ArrayList е един от типовете на интерфейса List:
Декларацията на променливата dates от тип List<Date>, означава:

- Колекцията с тип ArrayList е за съхраняване на специализиращ тип - **дати** (Date);
- Тъй като се използва параметризация, методът за добавяне трябва да има за параметър дата:
  - add(Date date) и не допуска друго, освен дата.

Параметризирани колекции (Generics)

Създаване на шаблонен клас:

```
public class Base < T > {
    T var ;
    public void setVar ( T var) {
        this.var = var ;
    }
    public T getVar () {
        return var;
    }
}
```

Параметризирани колекции (Generics)

Използване на шаблонен клас:

```
// class Derivative1 { }
 class Derivative2 { }

 Base<Derivative1> obj1 = new Base<Derivative1>();
obj1.setVar( new Derivative1());
Derivative1 obj2 = obj1.getVar();
```

Изключенията в езика Java

Изключенията в езика C++:

- Изключенията и прихващането им е допълнение към възможностите на езика;
- Много от програмистите на C++ не ги използват и не знаят как работят.
- Те се проявяват при грешки извън нормалното изпълнение на програмата;
- При незащитена от грешки програма се проявява изключение и програмата се прекъсва на системно ниво.

Изключенията в езика Java

В Java, изключенията **са част от езика**. Следствия:

- Сигнатурата на методите включва информация за изключението;
- Езиковият процесор изисква да се програмира в стил на извикване на методи, които:
  - Продуцират изключения;
  - Задължително се проверява за наличието им;
  - Задължително трябва да се обработват;
  - При програмирането на Java трябва да се отчитат генерираните от библиотеките изключения, защото методите на класовете им ги предизвикват.

Обработването на изключенията не е сложно, но е необходимо да се предвидят. Ако бъдат пропуснати, компилаторът **дава грешка**.

Изключенията в езика Java

Дефиниции:

Изключително състояние е програмно състояние, който не позволява да се продължи изпълнението на **текущия метод** или обхват.

Има съществена разлика между обикновения проблем и изключителното състояние. Ако програмният ход има достатъчно информация ***в текущия контекст***, която позволява да се продължи с действието на програмата, да се поправи и възстанови нейното състояние, това е ***обикновен проблем***.

Изключенията в езика Java

В **изключителното състояние**, *в текущия контекст*, не може да се направи нищо поради липса на информация. Това, което може да се направи е да се излезе от текущия контекст и да се разгледа проблема в по-висшия контекст. Това се осъществява като се изхвърли изключение.

Изключенията в езика Java са класове на езика, обекти от който се създават в изключителни състояния.

Изключенията в езика Java

Пример - делението на нула.

То може да се избегне с проверка на делителя и промяна на хода на програмата в резултат от проверката, без да се извършва самото деление.

Но какво следва, ако делителят е нула? Варианти:

- Ако е известно, в контекста на конкретния метод, какво да се прави в такъв случай, може да се проверява делителят и да се извърши обработка на ситуацията;
- Не е допустима стойността – изхвърля се изключение без да се прави проверка.

Текущият контекст (метод или обхват) определя дали състоянието е изключително или обикновено.

Изключенията в езика Java

Действие на системата при изключение:

Когато се изхвърли изключение се извършва следното:

- Създава се обект на изключението, както се създава всеки друг обект в Java- в динамичната памет, с оператор **new**.
- Текущия ход на изпълнение (този, който не може да продължи) се спира и от текущия контекст се изхвърля референция към обекта на изключението. В този момент механизма за обслужване на изключенията влиза в действие и започва да търси подходящо място, където програмата да продължи. Това подходящо място е прихващача на изключения (*exception handler*), чиято задача е да се справи с проблема така, че програмата да може да предаде управлението в друга част от кода или да продължи.

Изключенията в езика Java

Пример за изхвърляне на изключение по нулева референция към обект, например **testRef**. Възможно е да се подаде референция, която не е инициализирана, което може да се провери преди подаването на референцията като аргумент. Може да предаде информация за грешката в по-определен контекст чрез създаване на обект представящ нужната информация и "изхвърлянето му" извън текущия контекст-*изхвърляне на изключение*:

```
if(testRef == null) {
    throw new NullPointerException();
}
```

Това изхвърля изключение **без аргументи**, като позволява – в текущия контекст – да се прехвърли отговорността за проблема на по-високо ниво. Той се обработва от прихващача на изключението.

Изключенията в езика Java

Добавяне на аргументи на изключението

Изключението има два конструктора за всички стандартни изключения:

- подразбиращ се;
- Експлицитен със стринг за аргумент. Дава възможност да се добави информация за изключението:

```
if(testRef == null) {
   throw new NullPointerException("testRef is null");
}
```

Изключенията в езика Java

Действие на механизъма за предизвикване на изключенията:

- Изпълнява се оператора **new** за да се създаде обект, който не се извиква при нормалното изпълнение на програмата и се изпълнява конструктора за обекта;
- Обектът се "връща" от метода, **макар и типът на резултата му да не е като този**, за който е проектиран метода. Механизмът на предизвикване на изключенията **е друг начин на връщане на управлението от метод**.

Изключенията в езика Java

Разлики между възврат от метод и предизвикване на изключение:

- Може да се излезе от най-вътрешното (от произволно) ниво на вложеност;
- Връща се стойност, независимо дали методът или обхватът имат връщана стойност, след изключението те завършват изпълнението си;
- Мястото *където* се връща изпълнението на програмата може да е напълно различно от това при нормално завършване на метода;
- Може да конструира и изхвърли произволен обект, наследник на класа **Throwable**. Типично се изхвърля различен тип изключение за различни видове грешки. Идеята е да се съхрани информация в обекта на изключението чрез типа на избрания обект, така че прихващането да разбере какво да прави с изключението. Често единствената информация е типа на обекта на изключението, а нищо съществено не се съхранява в обекта.

Изключенията в езика Java

Прихващане на всички изключения

- Възможно е да се създаде обработчик, който хваща всякакъв тип изключение. Това се прави чрез хващане на базовия тип Exception. Има и други наследници на Exception, които могат да се използват вместо него.

Например:

```
catch(Exception e) {
    System.out.println("caught an exception");
}
```

Примерът ще хване всякакво изключение, така че трябва да се постави накрая на списъка от обработчици, за да се избегне писането на обработчиците, които иначе биха се намирали след него.

Изключенията в езика Java

Обработчици на изключения

Изхвърленото изключения завършва с *exception handler*. За всеки тип изключение има по един обработващ блок (catch). Обработчиците на изключения непосредствено следват аналитичния блок и са означени с ключовата дума catch:

```
try {
// код, който би могъл да генерира изключение
} catch(Type1 id1) {
// обработка на изключение от клас Type1
} catch(Type2 id2) {
// обработка на изключение от клас Type2
} catch(Type3 id3) {
// обработка на изключение от клас Type3
}
```

Изключенията в езика Java
Прекратяване и продължаване

Два основни модела в теорията на обработката на изключения:

- ***Прекратяването*** (което се поддържа в Java и C++) - грешката е толкова критична, че не може нищо да се направи на мястото на възникване. Който е изхвърлил изключението е преценил, че не може да се спаси положението и *не иска* да се връща управлението;

Изключенията в езика Java
Прекратяване и продължаване

- ***Продължаване*** **-** обработчикът на изключения се оставя да свърши нещо за поправяне на ситуацията, а после методът, в който е възникнало изключението се повтаря, с предполагаем успех. В този случай изключението е подобно на извикване на метод , който да оправи грешките. Алтернативно, може да се постави **try** в цикъл **while**, който продължава да влиза пак в **try**, докато резултатът стане удовлетворителен.

Изключенията в езика Java

Прихващане на изключение чрез базовите класове

Тъй като класът Exception е базов за всичките изключения, които са важни за програмиста, от него не се получава много информация за изключението. За целта може да се извикат методите на неговия базов тип **Throwable**:

String getMessage( ) -Връща детайли на съобщението.

String toString( ) - Връща кратко описание на Throwable, включително детайлно съобщение, ако има такова.

void printStackTrace( )

void printStackTrace(PrintStream)-Извежда Throwable и трасирания стек на извикванията на Throwable. Стекът на извикванията показва веригата от викания на методи, която е довела до мястото, където е изхвърлено изключението.

Изключенията в езика Java

Могат да се използват и методи от базовия тип на Throwable — **Object** (базовия тип на всички класове):

- getClass( );
- getName( );
- toString( );
- както и останалите възможности на класа, които са разгледани - чрез Class обект.

Изключенията в езика Java

Пример с използването на методите на Exception:

```
public class ExceptionMethods {
 public static void main(String[] args) {
  try {
    throw new Exception("Here's my Exception");
  } catch(Exception e) {
    System.out.println("In catch Exception e...");
    System.out.println("e.getMessage(): " + e.getMessage());
    System.out.println("e.toString(): " + e.toString());
    System.out.println("e.printStackTrace():");
    e.printStackTrace();
  }
 }
}
```

Изключенията в езика Java

Резултат:

In catch Exception e...

e.getMessage(): Here's my Exception

e.toString(): java.lang.Exception: Here's my Exception

e.printStackTrace():

java.lang.Exception: Here's my Exception

at ExceptionMethods.main

Изключенията в езика Java

Клаузата **finally**

Предназначение

- Ако има нужда от общ код, който трябва да се изпълни в блока **try**, независимо дали има или не изключение. Това обикновено е свързано с операции, различни от освобождаването на памет. За да се реализира това, в края на всички обработчици на изключения се добавя клаузата **finally**.

Изключенията в езика Java

Пример:

```
try {
  // Анализиран кодов блок:
 // който продуцира изключения A, B или C
 } catch (A a1) {
  // Обработка на A
 } catch (B b1) {
  // Обработка на B
 } catch (C c1) {
  // Обработка на C
 } finally {
  // Код, който трябва да се изпълни винаги
 }
```

Изключенията в езика Java

Пример за илюстрация на finally:

```
public class FinallyWorks {
  static int count = 0;
  public static void main(String[] args) {
    while(true) {
      try {
        if(count++ == 0)
          throw new Exception();
        System.out.println("No exception");
      } catch(Exception e) {
        System.out.println("Exception thrown");
      } finally {
        System.out.println("in finally clause");
        if(count == 2) break; // прекратяване на "while"
      }
    }
  }
}
```

Изключенията в езика Java

Изход:

Exception thrown

in finally clause

No exception

in finally clause

Изключенията в езика Java

Търсене на обработка

Принцип на търсене на обработващ блок:

- Когато е изхвърлено изключение функционалността, обслужваща изключенията преглежда “най-близките” обработчици в реда, в който са написани. Когато намери съвпадение, изключението се счита обработено и по-нататъшно търсене не се прави.
- Съвпадането не изисква точно съответствие между изключението и обработчика. **Обект от наследен клас може да се използва за обработчик, предназначен за базовия клас**.

Изключенията в езика Java

Пример:

```
class BaseClass extends Exception {}
class DerivativeClass extends BaseClass {}
public class TestClass {
public static void main(String[] args)
{
  try {
    throw new DerivativeClass();
  } catch(DerivativeClass s) {
    System.out.println("Caught DerivativeClass");
  } catch(BaseClass a) {
    System.out.println("Caught BaseClass");
  }
 }
}
```

Изключенията в езика Java

Изключението **DerivativeClass** ще се хване в първата **catch** клауза за която съвпада, като в случая е първата такава. Ако обаче се премахне първата клауза:

```
try {
  throw new DerivativeClass();
} catch(BaseClass a) {
System.out.println("Caught BaseClass");
}
```

catch(BaseClass e) ще хване **BaseClass *и всеки клас наследник***

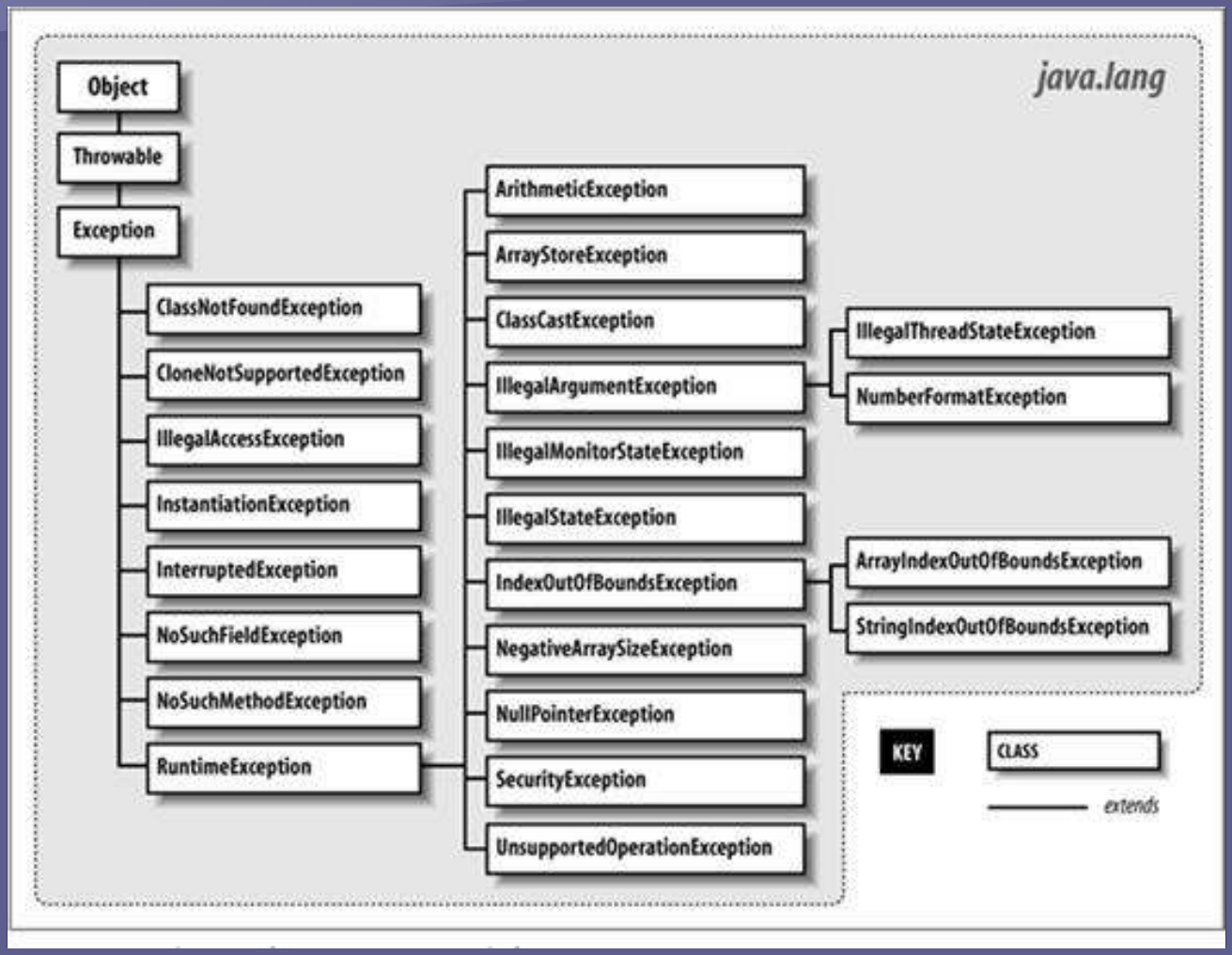
Изключенията в езика Java

Ако се разменят клаузата на базовия клас и наследника, например:

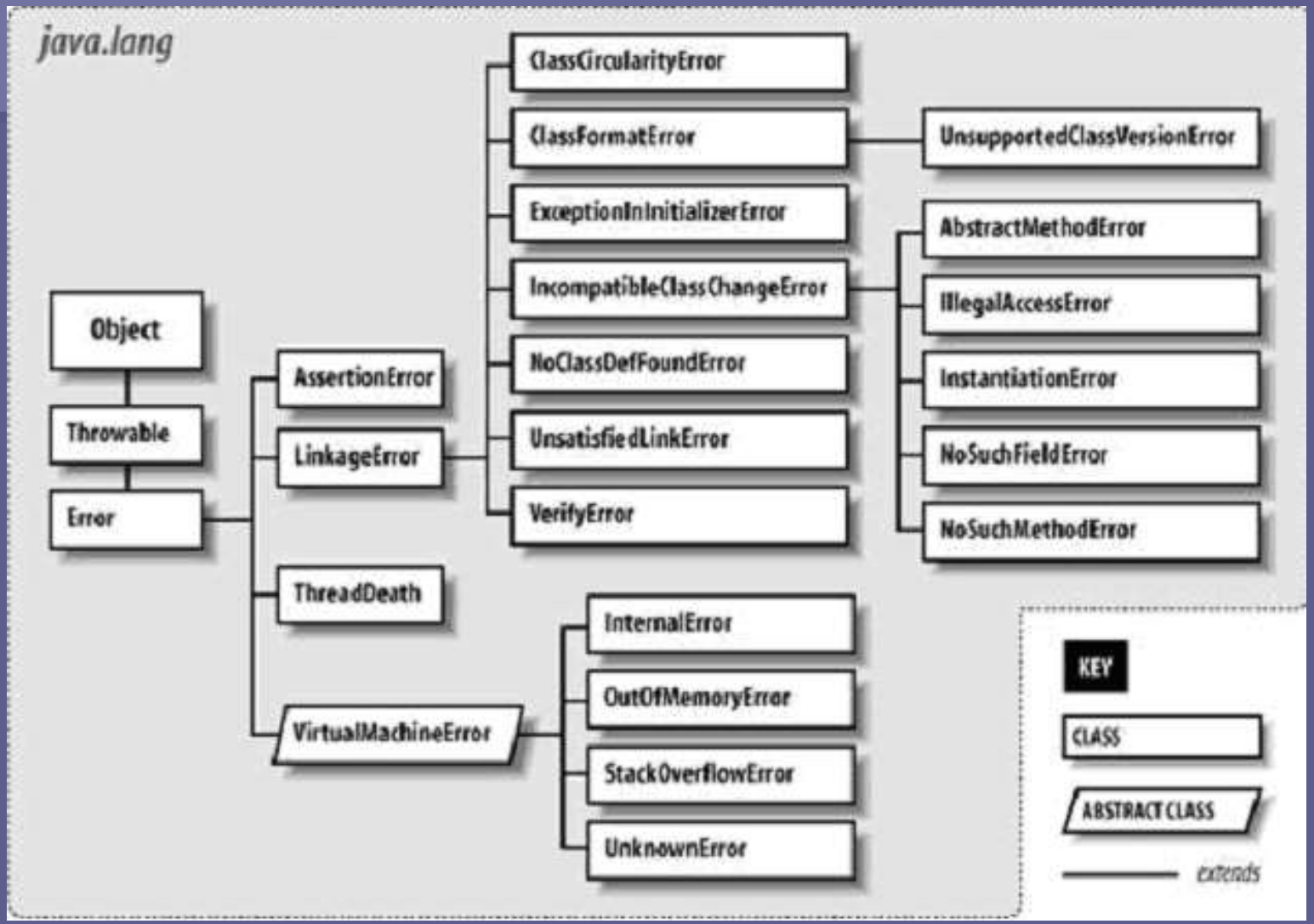
```
try {
  throw new DerivativeClass();
} catch(BaseClass a) {
  System.out.println("Caught BaseClass");
} catch(DerivativeClass s) {
  System.out.println("Caught DerivativeClass");
}
```

компилаторът ще издаде съобщение за грешка, понеже catch-клаузата на **DerivativeClass** не може никога да бъде достигната.

Стандартни изключения java.lang.Exception



Стандартни грешки java.lang.Error



Създаване на собствени класове изключения

- Няма ограничения да се използват само класовете изключния на езика Java.
- Кода може да се осигури срещу грешки, които създадената библиотека може да предизвика, но не са могли да бъдат предвидени, когато се е създавала йерархията от класове-изключения на Java.
- За да се създаде **собствен клас за изключение**, се наследява съществуващ клас изключение най-близък по значение.

Създаване на собствени класове изключения

```
class MyException extends Exception {
  public MyException() {}
  public MyException(String msg) {
    super(msg);
  }
}
public class Inheriting {
  public static void f() throws MyException {
    System.out.println("Throwing MyException from f()");
    throw new MyException();
  }
```

(продължава)

Създаване на собствени класове изключения

```
public static void g() throws MyException {
  System.out.println("Throwing MyException from g()");
  throw new MyException("Originated in g()");
}
public static void main(String[] args) {
  try {
    f();
  } catch(MyException e) {
    e.printStackTrace();
  }
  try {
    g();
  } catch(MyException e) {
    e.printStackTrace();
  }
 }
}
```

Създаване на собствени класове изключения

Изход – примерен стек на извикванията:

// от извикването на f():

Throwing MyException from f()
MyException
at Inheriting.f(Inheriting.java:16)
at Inheriting.main(Inheriting.java:24)
// от извикването на g():

Throwing MyException from g()
MyException: Originated in g()
at Inheriting.g(Inheriting.java:20)
at Inheriting.main(Inheriting.java:29)

Изключенията в езика Java

```
import java.io.*;           // System & File IO
public class FileTest {
    private void fileRead()
    {
    String str;
       try {
           RandomAccessFile fin =
                    new RandomAccessFile "<path>\\FileTest.java","r");
//пример за <path> C:\\oop_2
          while((str = fin.readLine()) != null ) {
             System.out.println(str);
          }
          fin.close();
        }
```

(продължава)

Изключенията в езика Java

```
        catch(EOFException e) { // стандартни изключения
            System.out.println("End of file encountered");
        }
        catch(FileNotFoundException e) {
            System.out.println("FileNotFound");
        }
        catch(IOException e) {
            System.out.println("IOException encountered");
        }
    }
    public static void main(String[] args) {
        FileTest oFile = new FileTest();
        oFile.fileRead();
    }
}
```

ВЪПРОСИ ?