

Išimtys (*Exceptions*)

11/2/2006

R.Vaicekuskas VU MIF: OP/OP2

1

Motyvacija

- Programose pasitaiko klaidų.
- Į klaidas reikia reaguoti
- Įvairiose programavimo sistemose bei kalbose naudojamos skirtingos klaidų apdorojimo priemonės
- Dažniausiai naudojamas klaidos požymio grąžinimas kviečiančiajam

2

Tipiška išskirtinė situacija, susijusi su klaidomis

```
readFile {  
  open the file;  
  determine its size;  
  allocate that much memory;  
  read the file into memory;  
  close the file;  
}
```

3

Tipiškas sprendimas (C kalba)

```
errorCodeType readFile {  
  initialize errorCode = 0;  
  open the file;  
  if (theFileIsOpen) {  
    determine the length of the file;  
    if (gotTheFileLength) {  
      allocate that much memory;  
      if (gotEnoughMemory) {  
        read the file into memory;  
        if (readFailed) {  
          errorCode = -1; }  
        } else { errorCode = -2; }  
      } else { errorCode = -3; }  
    }  
  }
```

```
close the file;  
  if (theFileDidntClose  
    && errorCode == 0) {  
    errorCode = -4; }  
  else {  
    errorCode =  
      errorCode & -4; }  
  } else {  
    errorCode = -5;  
  }  
  return errorCode;  
}
```

4

Ypatybės

- “Normalios” programos eigos kodas paskęsta klaidų apdorojimo kode
- Todėl tampa nesuprantama programos fragmento logika
- Lieka neaišku ar iš tiesų reaguojama į visas klaidas, ar atlaisvinami visi resursai
- Klaidų apdorojimą sunku modifikuoti

5

Java sprendimas

- Naudoti išskirtinių situacijų (*exception* - išimtis) mechanizmą
- Programa saugi – “nelūžta”
- “Normalus” kodas atskiriamas nuo klaidų apdorojimo kodo
- Objektinis sprendimas, “įvyniojant” klaidą į klaidos objektą
- Įstoriškai – “paveldėtos” C++ priemonės.

6

Išimtys įgyvendinamos

- Java kalbos sakiniai **try-catch-finally, throw**
- metodų išimčių deklaravimu **throws** konstrukcija
- *Throwable* klasių šeima

7

Išskirtinės situacijos atsiradimas

- Kai vykdoma operaciją / metodo kvietimas, kurio sėkminga baigmė nejmanoma
- Išreikštiniu būdu iššaukiant situaciją Java sakiniu:
throw *ExceptionObjectReference*;
- Žr. Java dokumentacijoje:
ArithmeticException, NullPointerException, IOException, ArrayIndexOutOfBoundsException

8


Išimties vystymasis

- Programos normali vykdymo tėkmė nutraukiama, sukuriama išimtį apibūdinantis objektas (*exception*)
- Jeigu nesimama priemonių ("pagauti" išimtį *try-catch* sakiniu) programos darbas užbaigiamas išvedant diagnostinį pranešimą.

9

Sukelti išimtį

```
class Exc1 {  
    public static void main (String[] args) {  
        for (;;)   
            System.out.println( calc() );  
    }  
  
    static int calc() {  
        int i = enterValue ();  
        int j = enterValue ();  
        int result = i / j;  
        return result;  
    }  
  
    static int enterValue () {  
        // Here user enters value  
        return 0;  
    }  
}  
/*  
"java Exc1" outputs  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at Exc1.calc(Exc1.java:10)  
    at Exc1.main(Exc1.java:4)  
*/
```



Try - catch – finally sakiny

```
try { Sakiniai  
}  
  
catch (ExceptionType1 var){ Sakiniai }  
catch (ExceptionType2 var){ Sakiniai }  
.....  
catch (ExceptionTypeN var){ Sakiniai }  
finally { Sakiniai  
}
```

11

Try sakinio vykdymas

- Susidarius išimčiai programos valdymas grįžta į artimiausiąjį metodų iškvietimų steką gaubiantįjį **try** sakinį, kuris geba apdoroti (pagauti) išimtį.
- Vykdomas pirmasis, tinkantis pagal situacijos tipą, **catch** blokas (išimtis apdorojama).
- Besąlygiškai - klaidos ir normaliu atveju - vykdomas **finally** blokas.
- Išimtis apdorota – vykdomas sakiny po

12

Išimties “gaudymas”

The diagram illustrates the flow of an exception in Java. It starts with a code snippet for `Exc2` that has a `try-catch` block for `ArithmeticException`. Annotations (1) through (4) point to specific parts of the code: (1) the `try` block, (2) the `enterValue` method call, (3) the `catch` block, and (4) the `exc.printStackTrace()` call. Below the code, the stack trace output is shown, with arrows indicating how it corresponds to the annotations: (1) points to the first line of the stack trace, (2) points to the line where the exception was thrown, (3) points to the line where the exception was caught, and (4) points to the line where the stack trace was printed.

```

class Exc2{
public static void main (String[] args) {
    try {
        for (;;)
            System.out.println( calc() );
    }
    catch (ArithmeticException exc) {
        System.out.println("Sorry: " + exc);
        System.out.println("Stack trace:");
        exc.printStackTrace();
    }
}

static int calc() {
    int i = enterValue ();
    int j = enterValue ();
    return i / j ;
}

static int enterValue () {return 0;}
}
/*
"Java Exc2" outputs:
Sorry: java.lang.ArithmeticException: / by zero
Stack trace:
java.lang.ArithmeticException: / by zero
    at Exc2.calc(Exc2.java:16)
    at Exc2.main(Exc2.java:5)*
*/

```

(1) Iš metodo kreipinio
neįžįstama normaliu
keliu

(2) Išimties
kilimo vieta

(3) Išimties
apdoravimo
blokas

(4) Išimtis
apdorota

13

Įdėtieji išimčių sakiniai

```
try {           // Išorinis try blokas
    ...        // A
    try {       // Vidinis try blokas
        ...     // B
    } catch(...) {...}
    ...        // C
} catch(...) {...}
...          // D
```

14

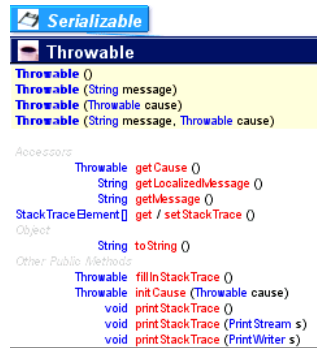
Išimties objektas

- Neša **informaciją** apie klaidą
- **Išvestas** iš klasės *java.lang.Throwable*
- Turi **konstruktorius**, **metodus** išgauti klaidos vietą, pranešimo tekstą, priežastį...
- Išvestinės klasės **specializuoja** klaidas bei gali suteikti papildomą informaciją
- Patiems iššaukiant situacijas naudinga naudoti **savo** apibrėžtas išimčių klases.

15

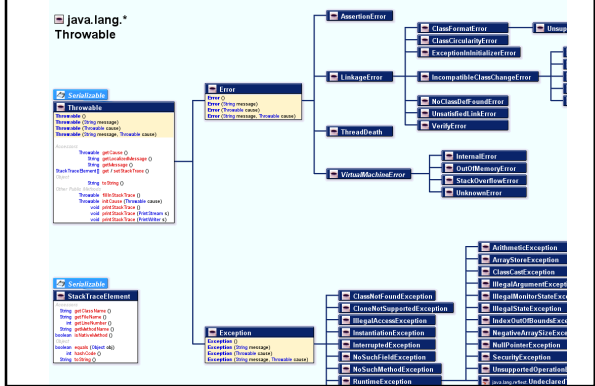
Throwable

(<http://www.falkhausen.de/en/diagram/html/java.lang.Throwable.html>)



16

Throwable šeima



Svarbiausios išimčių klasės

- **Throwable** - išimčių superklasė
- **Error** – išreiškia fatalias klaidas – paprastai neapdorojamos
- **Exception** – klaidos, kurios privalo būti deklaruojamos ir apdorojamos (*checked*).
- **RuntimeException** – kompiliatoriaus nekontroliuojamos klaidos – gali įvykti bet kada. Apdorojamos, esant reikalui

18

Catch bloką tvarka

- Nusako, išimčių tinkamumo tikrinimo eiliškumą
- Blokas tinka ("pagauna" išimtį) jeigu išimties objekto ref. gali būti priskirta **catch** dalyje nurodytam tipui.
- Kompiliatorius traktuoja kaip klaidingą **catch** bloką išsidėstymą, jei superklasės išimtinė situacija "gaudoma" anksčiau negu subklasės.

19

Tinkamos tvarkos iliustracija

```
try {  
    // some call  
}  
catch (IOException exc) {} // Child of Exception  
catch (ArithmeticException exc) {} // child  
catch (RuntimeException exc) {} // parent
```

20

Tipiniai veiksmai, apdorojant (pagautą) išimtį

- Naivusis - išvesti diagnostinį pranešimą ir nutraukti programos vykdymą.
- Optimistinis – "ištaisyti" situaciją ir toliau tęsti programą.
- Realistinis - jeigu klaida "nepataisoma", **catch** bloke sakiniu **throw** pakartotinai iššaukti išimtį, naudojant tą patį situacijos objektą, ar sukuriant naują, tikintis apdorojimo apgaubtyse.

21

Finally – resursų atlaisvinimo sekcija

```
Stream input = null;  
try{  
    input = new Stream(file); //--→  
    while (!input.eof())  
        process(input.next()); //--→  
}  
finally {  
    if (input != null) input.close();  
} // Close file in any case
```

22

Finally ypatybės

Jeigu iš **finally** bloko išeinama **break** / **return** / **throw** sakiniu, "pirminė" priežastis, dėl kurios paliekamas **try** blokas – "pamirštama":

```
try {  
    //...  
    return 1; // Forgotten  
}  
finally { return 2; } // →
```

23

Išimties iššaukimas: **throw**

```
// Average calculation  
int sum = 0, n = 0;  
//...  
try {  
    if (n == 0)  
        throw new Exception("No values for averaging"); //→  
    int avg = sum / n;  
}  
catch (Exception e) {  
    System.out.println("Averaging failed: " + e);  
}
```

24

Išimties klasės apibrėžimas

- Reikalingas, kai norime **specializuoti** išimtį, kad klientinė dalis galėtų išskirti ją iš kitų situacijų ir atitinkamai reaguotų.
- Kai norime papildyti **Exception** klasę **specifine** informacija.
- Paprastai su konkrečia biblioteka / susijusia klasių aibe susiejame vieną bazinę situacijos klasę, bei jos “vadas”.

25

Išimtis – “kliento sąskaita tuščia”

```
class AccountException extends Exception {  
    // Account specific part  
    private int balance = 0;  
    public int getBalance(){return balance;}  
  
    // No-args constructor  
    public AccountException(){}  
  
    // Constructor with message text  
    public AccountException(String msg){ super(msg); }  
  
    // Constructor with message text and additional info  
    public AccountException(String msg, int balance){  
        super(msg);  
        this.balance = balance;  
    }  
}  
  
class TestAccount  
{  
    // Test AccountException  
    public static void main(String[] s) {  
        try {  
            //...  
            throw new AccountException ("Sąskaitoje nepakanka pinigų", -100);  
            //...  
        } catch (AccountException e) {  
            System.out.println(" " + e + " ". Balansas: " + e.getBalance());  
            //AccountException: Sąskaitoje nepakanka pinigų. Balansas: -100  
        }  
    }  
}
```

26

Išimčių kontrolė: throws

- Išimtys, nesančios **Error**, **RuntimeException**, yra kontroliuojamos (*checked*)
- tokios išimtys privalo būti arba pagautos metodo kode arba deklaruotos metodo apibrėžime.
- *pavidalas*
... method (parameters)
throws ExcType1,...,ExcTypeN { ... }
- Išvestinės klasės užklotyje gali tik susiaurinti / specializuoti deklaruojamų išimčių aibę.

27

Pavyzdys

```
//Pavyzdys:  
class Account {  
    private balance;  
  
    public void withdraw(int amount)  
        throws AccountException {  
        if (balance - amount >= 0)  
            balance -= amount;  
        else  
            throw AccountException("Stop", balance - amount);  
    }  
  
    public static void main(String[] args) {  
        try {  
            Account a = new Account();  
            a.withdraw(100);  
        } catch (AccountException exc) {  
            //...  
        }  
    }  
}  
  
class CreditAccount extends Account {  
    // No exception declared  
    public void withdraw(int amount){  
        //...  
    }  
}
```

Kada nenaudoti situacijų

Neteisinga naudoti situacijas įprastai (normaliai) programos tėkmei valdyti

```
// Antipattern  
try {  
    for (;;) {  
        process(stream.next());  
    }  
} catch (StreamEndException e){  
    stream.close();  
}
```

29

Pabaiga



30