Išskirtinės situacijos
(Exceptions)

Motyvacija

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

Tipiška situacija, susijusi su klaidomis

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


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;
}
```


Ypatybės

- "Normalios" programos eigos kodas paskęsta klaidų apdorojimo kode
- Tampa nesuprantama programos fragmento logika
- Lieka neaišku ar iš tiesų reaguojama į visas klaidas, ar atlaisvinami visi resursai

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.

Išimtys įgyvendinamos

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

7

Išskirtinės situacijos atsiradimas

- Kai vykdoma operaciją / metodo kvietimas, kurio sėkmingas baigmė neįmanoma
- Išreikštiniu būdu iššaukiant situaciją Java sakiniu:

  throw *ExceptionObjectReference*;
- Žr. Java dokumentacijoje: **ArithmeticException, NullPointerException, IOException, ArrayIndexOutOfBoundsException**

8

Vystymasis

- Programos normali vykdymo tėkmė nutraukiama, sukuriamas išimtį apibudinantis objektas (*exception*)
- Jeigu nesimama priemonių ("pagauti" situaciją 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)
*/
```

Išimties kilimo vieta

Try - catch – finally sakinys

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 steke gaubiantįjį **try** sakinį, kuris geba išimtį apdoroti (pagauti).
- Vykdomas pirmasis, *tinkantis pagal situacijos tipą,* **catch** blokas (išimtis apdorojama).
- Besalygiškai - klaidos ir normaliu atveju - vykdomas **finally** blokas.
- Išimtis apdorota – vykdomas sakinys po **try** , priešingu atveju – taikomas gaubiantis **try** sak.

12

Išimties “gaudymas”

```
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 negrįžtama normaliu keliu

(3) Išimties apdorojimo blokas

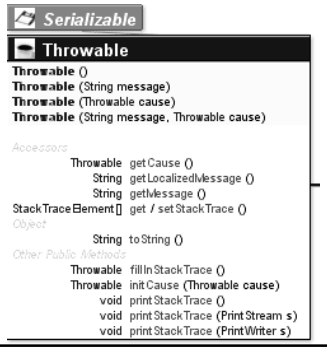
(4) Išimtis apdorota

(2) Išimties kilimo vieta

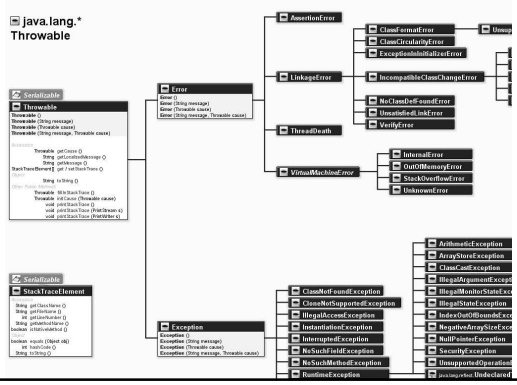
Situacijos objektas

- 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 situacijų klases.

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



Throwable šeima



Svarbiausios situacijų klasės

- **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

Catch blokų tvarka

- Nusako, išimčių tinkamumo tikrinimo eiliškumą
- Blokas tinka (“pagauna” išimtį) jeigu situacijos objekto ref. gali būti priskirta **catch** dalyje nurodytajam tipui.
- Kompiliatorius traktuoja kaip klaidingą **catch** blokų išsidėstymą, jei superklasės situacija “gaudoma” anksčiau negu subklasės.

Tinkamos tvarkos iliustracija

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

19

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.

20

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
```

21

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; } //→
```

22

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);}
```

23

Išimties klasės apibrėžimas

- Reikalingas, kai norime specializuoti išimtį, kad klientinė dalis galetų 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".

24

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
     }
   }
}
```


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.

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();
}
```


Pabaiga