Zpracování výjimečných situací

Jazyky bez ovladačů výjimek

výskyt výjimky způsobí přenesení výpočtu do operačního systému.

Ten vypíše zprávu a ukončí program.

Jazyky s ovladačem výjimek

programy mohou zachytávat některé situace, provést náhradní akci a pokračovat ve výpočtu

Def.

Výjimečná situace je neobvyklou událostí (chybovou či nechybovou), která je detekovatelná hardwarem či softwarem a vyžaduje speciální zpracování.

Druhy výjimek: **-vestavěné**

-uživatelské

Výjimka vzniká, nastane-li s ní asociovaná událost.

Speciální zpracování vyžadované po detekci výjimky se nazývá zpracováním výjimky. Programová jednotka, která je provádí se nazývá ovladačem výjimky (exception handler).

Zpracování výjimečných situací

Co je třeba ujasnit při použití mechanismu zpracování výjimek:

-jak a kde specifikovat ovladače a v jakém rozsahu působí?

-jak je výskyt výjimky (volání výjimky) propojeno s ovladačem?
(dynamicky ? staticky)
-kde pokračuje výpočet poté, co ovladač výjimky skončí činnost?
(za ovladačem ? Za místem vzniku)
-jsou zavedeny vestavěné výjimky?

-mohou být vestavěné výjimky explicitně vyvolány?

-jsou hardwarem detekovatelné chyby chápány jako výjimky které lze ošetřit?

-jak specifikovat uživatelské výjimky?

-lze výjimky potlačit a jak?

Zpracování výjimečných situací

✓Jak se zpracovává výjimečná situace? = Šíření (propagace) výjimky

✓Kde pokračovat ve výpočtu po výjimce?

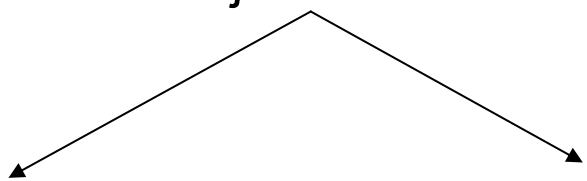
? Za příkazem co ji způsobil

? Za programovou jednotkou kde výjimka vznikla

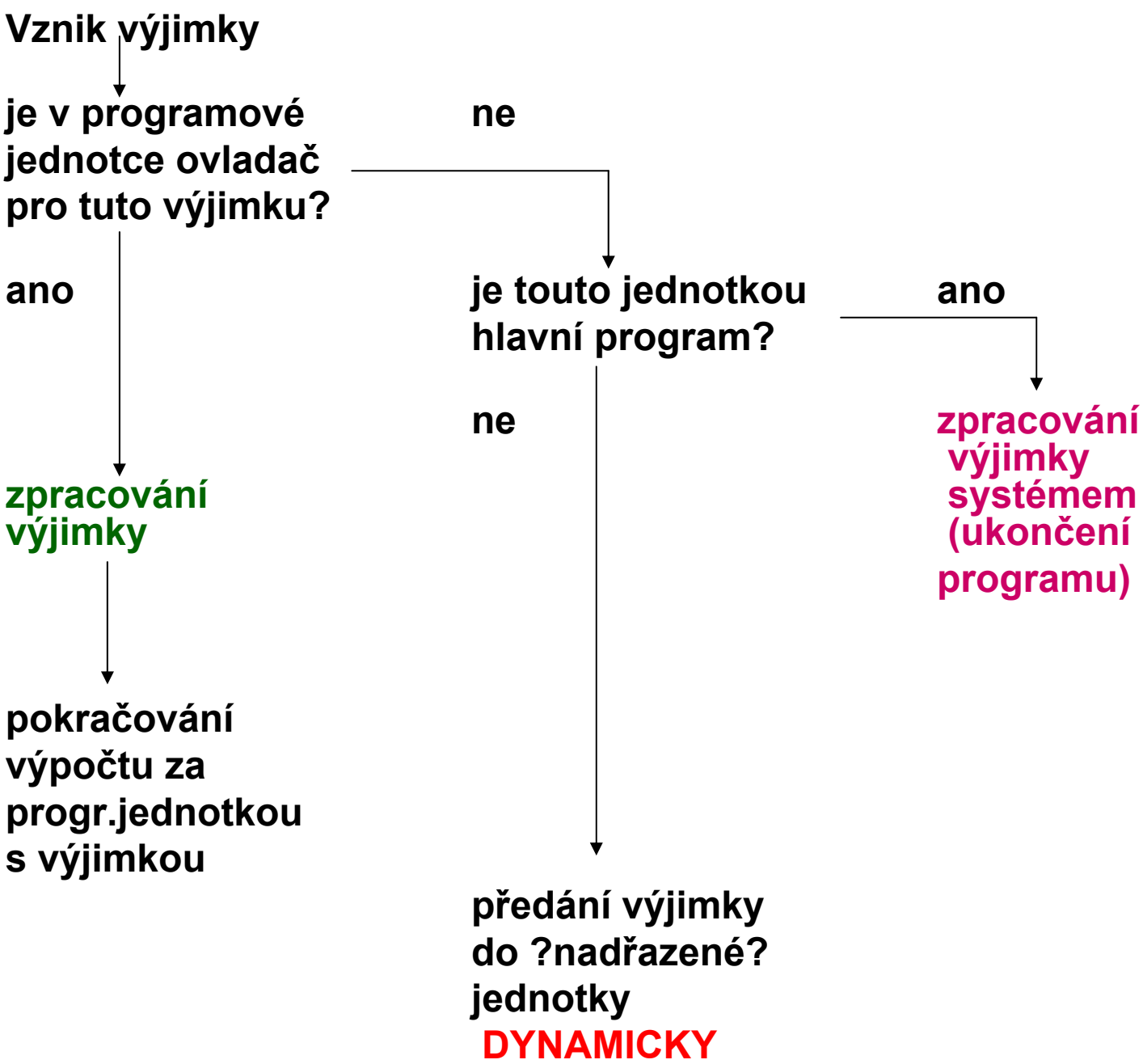
? Ukončit výpočet

V sekvenci příkazů zpracování výjimky může být použit příkaz pro předání výjimky do nadřazené jednotky.

Není-li v programové části kde vznikla výjimka → hledat ovladač v nadřazené jednotce



Staticky nadřazené ? Dynamicky nadřazené


Vznik výjimky
je v programové jednotce ovladač pro tuto výjimku?
ne
ano
je touto jednotkou hlavní program?
ano
ne
zpracování výjimky
zpracování výjimky systémem (ukončení programu)
pokračování výpočtu za progr.jednotkou s výjimkou
předání výjimky do ?nadřazené? jednotky
DYNAMICKY

Zpracování výjimečných situací

Poprvé zavedeny v PL1 s tvarem ovladače:

ON jméno výjimky BEGIN …END;

-výjimky jsou dynamicky vázány k naposledy provedenému ON příkazu (zhoršuje čitelnost).

-mechanismus výjimek je nejednotný. Některé výjimky vrací řízení do příkazu, který způsobil výjimku, jiné ukončují program. Některé jsou implicitně povoleny (je možné je zakázat), jiné implicitně zakázány a musí se povolit. Uživatelské výjimky mohou přenášet řízení na libovolné místo programu označené návěštím.

-uživatelské výjimky se definují konstrukcí

DCL CONDITION jméno

-vestavěné **ZERODIVIDE, SUBSCRIPTRANGE, ENDFILE**, … (celkem 22)

-výjimky lze explicitně vyvolat příkazem **SIGNAL CONDITION (jméno)**

př. **ON SUBSCRIPTRANGE /* ovladac */ BEGIN … END; …**

(NOOVERFLOW): BEGIN … END; /*vypnuti kontroly preteceni */

(SUBSCRIPTRANGE): BEGIN … END; /*zapnuti kontroly indexu*/

Zpracování výjimečných situací v C++

Klíčová slova **try, catch, throw**

Tvar ovladače:

```
try {
   //programový text, ve kterém  vzniká výjimka
}
catch (formalní parametr) {
  //příkazy ovladače
}
…
catch (formalní parametr) {
--příkazy ovladače
}
```

-catch je jméno všech ovladačů, rozlišují se formálním parametrem. Nemusí jím být proměnná, může jím být jméno typu. Form. parametr lze použít k přenosu informace do ovladače. catch(…) chytá všechny výjimky.

Zpracování výjimečných situací v C++

-výjimky jsou vyvolávány pouze explicitně příkazem

throw [výraz]; //[ ] jsou metasymbolem

-typ výrazu určuje, který příslušný ovladač použít, což není příliš informativní

-throw bez operandu se může vyskytnout jen v ovladači, způsobí pak znovuvyvolání výjimky a její zpracovávání ve vyšší úrovni

-neošetřené výjimky se propagují do místa volání funkce, ve které výjimka vznikla. Propagace může pokračovat až do funkce main. Pokud ani tam není ovladač nalezen, program je ukončen.

-po provedení příkazů ovladače je řízení přeneseno na příkaz za posledním z ovladačů (z nichž jeden zpracoval výjimku)

-všechny výjimky jsou uživatelské, hardwarem detekovatelné výjimky nemá, což je škoda.

Zpracování výjimečných situací v C++

př. 1VyjimkaC.CPP

```
#include <iostream.h>
void Ftest(int fi)
{
	cout << "jsem v Ftest:"<< fi << "\n";
	if (fi) throw fi;	// testovaci funkce vyhazuje vyjimku
}
int main()
{
	cout << "start\n";
	try {
	   cout << "jsem v try \n";
	   Ftest(0);
	   Ftest(1);
	   Ftest(2);
	   }
	   catch (int i)
	   {
	      cout << "zachycena hodnota:" << i << "\n";
	   }
	cout << "konec";
	return 0;
}
```

Zpracování výjimečných situací - Java

-založeno na C++, s ohledem na OOP paradigma

try, catch throw

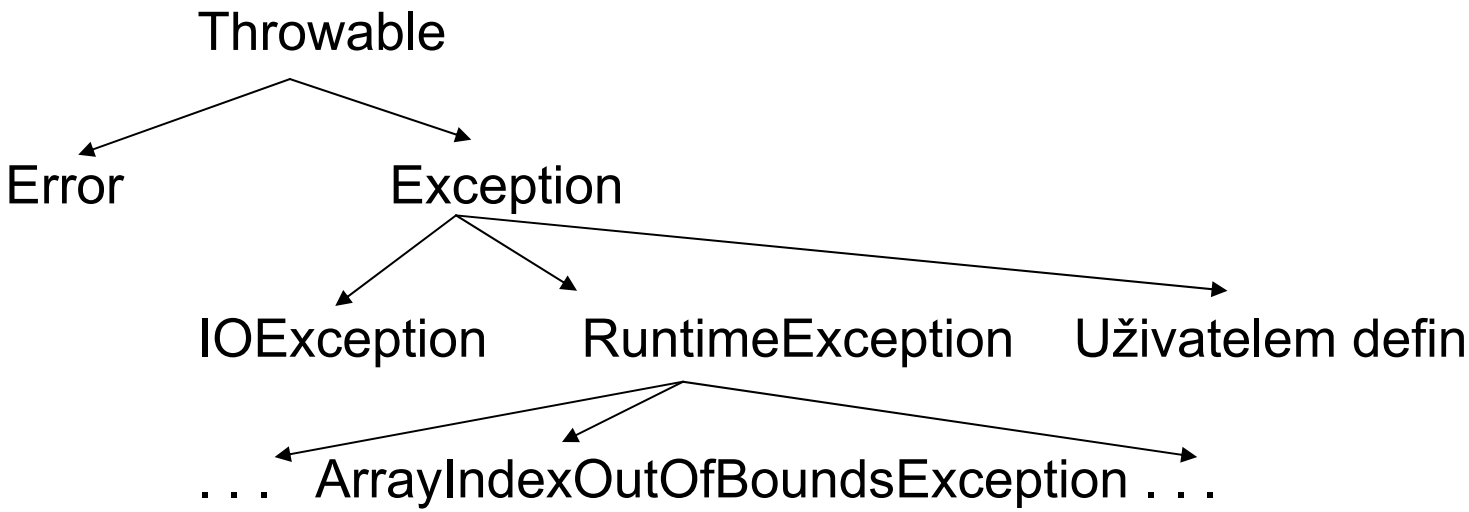
navíc má finally, throws

-všechny výjimky jsou objekty tříd, které jsou potomky třídy Throwable

-knihovna Javy obsahuje dvě podtřídy Throwable

1. Error výjimky této třídy jsou vyvolávány Java interpretem (např. při přetečení haldy), jejich zpracování nepřísluší uživateli.
2. Exception uživatelské výjimky jsou obvykle jejími potomky. Má dvě předdefinované podtřídy:
   - IOException
   - RuntimeException

Zpracování výjimečných situací - Java



Třída Error: Chyby vzniklé v JVM, na které uživatel nemůže reagovat
Např. OutOfMemoryError

Třída Exception Program by měl zpracovávat tyto výjimky

IOException Chyby při vstup/výstupních operacích

-RuntimeException reprezentují různé časté chyby (např. jejím potomkem definovaným v java.util je ArrayIndexOutOfBoundsException).

Zpracování výjimečných situací - Java

-catch musí mít parametr. Tyto parametry musí být potomky třídy Throwable
Parametr určuje jaké výjimky bude catch zachytávat

-try klauzule je stejná jako v C++

-throw slouží jako v C++ k vyvolávání výjimek. Může být použit spolu s operátorem new k vytvoření objektu

např.

```
class MyException extends Exception {
        public MyException() { }
        public MyException(String message) {
            super(message); //posila zpravu predkovi = Exception
        }
}
```

lze ji manuálně vyvolat

```
throw new MyException;
```

Zpracování výjimečných situací - Java

nebo vytvořit instanci příkazem

MyException mE = new MyException( );

a použít ji příkazem

throw mE;

Konstruktor s parametrem message lze použít příkazem

throw new MyException("zprava o chybe");

-výjimka je zachycena prvým ovladačem, který má parametr stejné třídy jako parametr **throw** příkazu, nebo parametr throw je potomkem stejné třídy.

-není-li nalezen ovladač pro příslušnou konstrukci **try**, hledá se v nejbližším obepínajícím **try**, nenajde-li se v celé metodě, hledá se v jednotce, která metodu vyvolala … až do main. Pokud ani tam není, program je ukončen.

Zpracování výjimečných situací - Java

-výjimky třídy Error a RuntimeException se nazývají unchecked (nekontrolované) exception. Programátor je nemusí ošetřit.

Ostatní jsou checked exception.

-checked exceptions (kontrolované, typicky jsou to IOException), které metoda vyvolává, musí:

buď mít v metodě uvedený ovladač,

nebo být uvedeny v tzv. seznamu **throws**.

-seznam throws je součástí hlavičky metody. Např.

void jmenoMetody( ) throws IOException { …}

-metoda, která volá metodu citující kontrolovanou výjimku ve své throws klauzuli, má tři možnosti jak s výjimkou naložit:

1. zachytit ji a zpracovat ve svém ovladači catch,
2. zachytit ji, a v ovladači vyvolat některou výjimku ze svého seznamu throws,
3. deklarovat ji ve svém throws seznamu a žádný ovladač pro ni nedělat.

Zpracování výjimečných situací - Java

Příklad 2java zachycení dělení nulou

```
public class Vyjimka
{
  public static void main (String [ ] args )
    {
     int [ ] cislo = {2, 4, 8, 16, 32};
     int [ ] d =     {2, 1, 0, 4,  8};
     for (int i = 0; i<cislo.length; i++)
      {
       try {
           System.out.println
            (
             cislo[i] / d[i]
            );
           }
       catch (ArithmeticException exc)
           {
           System.out.println("nelze delit nulou");
           }
      }
    }
}
```

Zpracování výjimečných situací - Java

Příklad 3java zachycení překročení rozsahu mezí pole

```
class VyjimkaDemo {
  public static void main(String args[]) {
    int cisla[] = new int[4];

    try {
      System.out.println("Před generovanim vyjimky");

      // Generuj index out-of-bounds vyjimku
      cisla[7] = 10;
      System.out.println("nedostupna cast");
    }
    catch (ArrayIndexOutOfBoundsException exc) {
      // zachyceni vyjimky
      System.out.println("Index mimo meze");
    }
    System.out.println("Za catch prikazem");
  }
}
```

```
// př. 4Java Výjimka může být generována jednou metodou a zachytávána jinou
class ExcTest {
  // Generuj výjimku.
  static void genException() {
    int cisla[] = new int[4];
    System.out.println("Před generováním výjimky");
   // generuj index mimo meze výjimku
    cisla[7] = 10;
    System.out.println("nedosazitelne");
  }
}

class Vyjimka2 {
  public static void main(String args[]) {
    try {
      ExcTest.genException();
    }
    catch (ArrayIndexOutOfBoundsException exc) {
      // zachycení výjimky
      System.out.println("Index mimo meze");
    }
    System.out.println("Za catch příkazem");
  }
}
```

-pro možnost provést "úklid" bez ohledu co se stane v příkazech konstrukce try, je zavedena klauzule finally tvaru:

finally { příkazy úklidu }

-uvádí se za klauzulemi catch a její příkazy se provedou bez ohledu zda výjimka v throw vznikla/nevznikla, byla/nebyla ošetřena některým catch, či bude propagována.

-Konstrukce try bez uvedení catch může být následována klauzulí finally.

Má význam, obsahuje-li složený příkaz buď break nebo continue nebo return příkaz. Např.

```
try {
    for (index = 0; index < 100; index++) {
      …
      if (…) {
        return;
      }  //** end of if
  }  //** end of try clause
  finally {
    …
  }  //** end of try construct
```

Klauzule finally bude provedena bez ohledu zda cyklus končí vyčerpáním, nebo příkazem return.

```
//Př. 5Java Použití finally.
class UzijFinally {
  public static void genException(int neco) {
    int t;
    int cisla[] = new int[2];

    System.out.println("Dostal " + neco);
    try {
      switch(neco) {
        case 0:
          t = 10 / neco; // generuje div-by-zero error
          break;
        case 1:
          cisla[4] = 4; // generuje array index error.
          break;
        case 2:
          return; // návrat z try bloku
      }
    }
```

```
 catch (ArithmeticException exc) {
     // catch the exception
     System.out.println("Nelze dělit nulou");
     return; // return from catch
   }
   catch (ArrayIndexOutOfBoundsException exc) {
     // catch the exception
     System.out.println("Nenalezen prvek");
   }
   finally {
     System.out.println("Konec try.");
   }
  }
}

class VyjimkasFinally {
  public static void main(String args[]) {

   for(int i=0; i < 3; i++) {
     UzijFinally.genException(i);
     System.out.println();
   }
  }
}
```

Zpracování výjimečných situací - Java

Supertřídy a podtřídy výjimek

Klauzule catch pro nadtřídu se vztahuje také na všechny její podtřídy.

Tzn. catch (TypVyjimky objektVyjimky)

zachytí jak výjimky typu TypVyjimky tak i výjimky všech jejích podtříd

Důsledek: Chcete-li zachytit vyjimku podtřídy, musíte její catch uvést dříve než catch její nadtřídy. Opačný zápis catch by způsobil nedosažitelný kód.

Př. 51Java

```
class SubVyjimka {
  public static void main(String args[]) {
    // citatelu je vic nez jmenovatelu
    int citat[] = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int jmenov[] = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<citat.length; i++) {
      try {
        System.out.println(citat[i] + " / " +
                           jmenov[i] + " je " +
                           citat[i]/jmenov[i]);
      }
      catch (ArrayIndexOutOfBoundsException exc) {
        // chytne vyjimku
        System.out.println("Neni dosti operandu");
      }
      catch (Throwable exc) {
        System.out.println("Nastala jakasi vyjimka");
      }
    }
  }
}
```

Zpracování výjimečných situací - Java

Vnořování bloků try

Do **try** bloku lze vnořit další **try** bloky

```
try  {    . . .
  try     {         . . .
  }
          catch úsek1 programu
. . .
}
catch úsek2 programu
```

Výjimky z vnitřního try nezachycené v úseku1 budou propagovány do staticky i dynamicky (platí oboje) nadřazené jednotky (úsek2)

Zpracování výjimečných situací - Java

Vyhození výjimky

Kromě automatického generování výjimky systémem Javy, lze výjimku generovat „manuálně“ programovým příkazem

throw objektVyjimka ;

kde objektVyjimka je instancí třídy odvozené z **Throwable.**

Př.52Java

```
// rucne vyhozena vyjimka
class Vyhozeni {
  public static void main(String args[]) {
    try {
      System.out.println("Pred vyhozenim");
      throw new ArithmeticException();
    }
    catch (ArithmeticException exc) {
      System.out.println("Vyjimka chycena");
    }
    System.out.println("Za blokem try/catch");
  }
}
```

Zpracování výjimečných situací - Java

Opakované vyhození výjimky

Při zachycení v catch můžeme výjimku znovu vyhodit použitím příkazu throw v bloku catch. Je pak zachytávána opakovaně v dynamicky nadřazených jednotkách v souladu s pravidly propagace výjimek.

Poslouží to pro zpracování výjimečné situace po částech, každé z catch může mít jiné schopnosti.

Př.53Java

```
class Opakovane {
  public static void genException() {
    int citatel[] = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int jmenovatel[] = { 2, 0, 4, 4, 0, 8 };
    for(int i=0; i<citatel.length; i++) {
      try {
        System.out.println(citatel[i] + " / " +
                           jmenovatel[i] + " je " +
                           citatel[i]/jmenovatel[i]);
      }
      catch (ArithmeticException e) {
        System.out.println("Deleni nulou");
      }
      catch (ArrayIndexOutOfBoundsException e) {
       System.out.println("Neni dosti operandu");
        throw e; //znovuvyhozeni vyjimky
 }}} }
```

Zpracování výjimečných situací - Java

```
class OpakDemo {
  public static void main(String args[]) {
    try {
      Opakovane.genException();
    }
    catch(ArrayIndexOutOfBoundsException e) {
      // zachyceni po znovuvyhozeni
     System.out.println("Nastala chyba  " +
                        "program ukoncen");
    }
  }
}
```

//konec příkladu 53Java

Zpracování výjimečných situací - Java

Použití throws

✓Některé výjimky, pokud je metoda nemůže zpracovat, musí vyjmenovat v throws seznamu, a tím jasně deklarovat, že je bude vyhazovat (propagovat) dál.

✓Do seznamu throws není třeba uvádět výjimky odvozené z RuntimeError a z Error. Java předpokládá, že tyto výjimky může vyhazovat každá metoda. (tzv nekontrolované)

✓Ostatní musí být v throws seznamu, jinak se program nepřeloží. Vynucuje tím např ošetření I/O operací (jsou kontrolované)

Př.54Java

Zpracování výjimečných situací - Java

```
class ThrowsDemo {
  public static char vypis(String s)
    throws java.io.IOException {
    System.out.print(s + ": ");
    return (char) System.in.read();
  }

  public static void main(String args[]) {
    char ch;
    try {
      ch = vypis("Napis pismeno");
    }
    catch(java.io.IOException e) {
      System.out.println("nastala I/O vyjimka");
      ch = ' ';
    }

    System.out.println("napsal jsi " + ch);
  }
}
```

Zpracování výjimečných situací - Java

Výjimky vytvořené programem

✓Programátor může definovat vlastní výjimky jako podtřídy třídy Exception

✓Exception nemá žádné vlastní metody, dědí ale metody svého rodiče Throwable. Takže podtřídy Exception je také zdědí.

Př.55Java

```
class NecelociselnyPodil extends Exception {
  int n;   int d;
  NecelociselnyPodil(int i, int j) {
    n = i;  d = j;
  }
  public String toString() {
    return "Podil " + n + " / " + d + " neni celociselny ";
}}
```

Zpracování výjimečných situací - Java

```
class UzivVyjimka {
  public static void main(String args[]) {
    int citatel[] = { 4, 8, 15, 32, 64, 127, 256, 512 };
    int jmenovatel[] = { 2, 0, 4, 4, 0, 8 };
    for(int i=0; i<citatel.length; i++) {
      try {
        if((citatel[i]%jmenovatel[i]) != 0)
          throw new
            NecelociselnyPodil(citatel[i], jmenovatel[i]);
        System.out.println(citatel[i] + " / " +
                           jmenovatel[i] + " je " +
                           citatel[i]/jmenovatel[i]);
      }
      catch (ArithmeticException e) {
        System.out.println("Deleni nulou ");
      }
      catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Neni dosti operandu ");
      }
      catch (NecelociselnyPodil e) {
        System.out.println(e);
}}} }
```

Zpracování výjimečných situací - ADA

Předdefinované: CONSTRAINT _ERROR
NUMERIC_ERROR
STORAGE_ERROR
TASKING_ERROR
PROGRAM_ERROR

Syntax:

```
…
begin
        příkazy
exception
        when  výjimka1  => příkazy
        when  výjimka2  => příkazy
                …
        when  výjimkaN  => příkazy
end;
…
```

Zpracování výjimečných situací - ADA

Výjimky lze potlačit pomocí pragma SUPRESS (i selektivně) např.

```
declare
        pragma SUPRESS(RANGE_CHECK, M);
        subtype MALE_INT is INTEGER range 1..2;
        M, N: MALE_INT;
        MIMO: INTEGER = 3;
begin
        M := MIMO;      // nevznikne vyjimka
        N := MIMO;      // vznikne vyjimka
        …
end;
```

Jak nelze výjimku použít:

```
<<opakuj>>
        …
exception  …  =>  … ;   goto  opakuj ;          // to se nepřeloží
```

Př. 6ADA Způsobí výjimku, není-li součin dvou čísel kladný

```
with TEXT_IO; use TEXT_IO;
procedure V is
 function CHYBNA(I,J:INTEGER) return POSITIVE is
 begin
  return POSITIVE(I*J);
 exception when CONSTRAINT_ERROR =>PUT_LINE("VYJIMKA");
                              return 1000;
 end CHYBNA;
 I,J:INTEGER;
 package I_IO is new INTEGER_IO(INTEGER);
 package P_IO is new INTEGER_IO(POSITIVE);
begin
  loop
   PUT_LINE("zadej dve integer cisla");
   I_IO.GET(I);
   I_IO.GET(J);
   P_IO.PUT(CHYBNA(I,J));
   NEW_LINE;
   exit when I=J;
  end loop;
end V;
```

Zpracování výjimečných situací - ADA

Uživatelem zavedené výjimky v Adě

-deklaruje se jako proměnná

-vyvolá se příkazem ***raise jméno*** (lze tak vyvolat i předdefinovanou**)**

Př. 8ADA

```
package STACK1 is --specifikace
  subtype STACK_ITEM is INTEGER range -10_000..10_000;
  procedure PUSH(ITEM:STACK_ITEM);
  function POP return STACK_ITEM;
  STACK_OVERFLOW : exception;
  STACK_UNDERFLOW : exception;
end STACK1;
```

```
package body STACK1 is
  --deklaracni cast
  STACK_SIZE: constant INTEGER := 1_000;
  STACK_ARRAY: array(1..STACK_SIZE) of STACK_ITEM;
  top: INTEGER range 0..STACK_SIZE;
  procedure PUSH(ITEM: STACK_ITEM) is
  begin
    if TOP = STACK_SIZE then raise STACK_OVERFLOW; end if;
    TOP := TOP + 1;
    STACK_ARRAY(TOP) := ITEM;
  end PUSH;
  function POP return STACK_ITEM is
  begin
    if TOP = 0 then raise STACK_UNDERFLOW; end if;
    TOP := TOP - 1;
    return STACK_ARRAY(TOP + 1);
  end POP;
begin      --inicializacni cast
  TOP := 0;
end STACK1;
```

Zpracování výjimečných situací - ADA

```
--hlavni program, vyuziva kompilacni jednotku STACK1
with STACK1,TEXT_IO; use STACK1,TEXT_IO;
procedure STACK1H is
  package STACK_IO is new INTEGER_IO(STACK_ITEM);
  use STACK_IO;
  X,Y: STACK_ITEM;
begin
  PUSH(5);
  X := 4;
  PUSH(X);
  PUT(POP);  --vystup 4
  Y := POP;  --vystup 5
  PUT(Y);
  NEW_LINE; // dalsi prikazy
exception
  when STACK_OVERFLOW => PUT_LINE("je plny") ;
  when STACK_UNDERFLOW => PUT_LINE("je prazdny") ;
  when others => PUT_LINE("jina zavada") ;
end STACK1H;
```

Zhodnocení výjimek

- C++ může vyhazovat výjimky libovolného (v programu či v systému definovaného) typu. Uživateli taková výjimka moc neřekne
- Java může vyhazovat pouze objekty odvozované z třídy Throwable, tím dovolí separovat výjimky od ostatních objektů
- Java klauzulí throws řekne uživateli, které kontrolované výjimky smí vyhazovat a sama je nezpracovává.
- Java finally umožňuje čisticí akce bez ohledu na výsledek try příkazu
- JVM implicitně vyhazuje řadu předdeklarovaných výjimek, C++ může pracovat jen s explicitně vyhazovanými
- Schopnosti výjimek Javy a Ady jsou zhruba rovnocenné. V Adě není ekvivalent throws seznamu (ten zlepšuje čitelnost).
- Výjimky Javy jsou s ohledem na systémem detekovatelné výjimky bližší Adě než C++
- C# zavádí konstrukce výjimek velmi podobné Javě. Nemá však throws klauzuli.