Проектиране и Тестиране на Софтуер
ТУ, кат. КС, летен семестър 2012

Лекция 5

Тема:

Обработка на Изключения

Обработка на изключения Exception Handling Techniques

Съдържание:

- Въведение в Exception Handling
- ЕН концепции и дефиниции
- Предимства на Exception Handling
- Проектиране на ПП за обработка на изключения
- Exception Handling техники при ПЕ: Fortran, PL/1, Basic, Pascal, C, Ada, C++, Java, C#

Exception Handling-Въведение

ПЕ предоставят средства и механизми, когато

hardware-откриваеми и/или

software-откриваеми

събития (условия) настъпят, възникнат.

Програмистът дефинира събития (условия) и собствена потребителска реакция на тези събития (условия), която се активира, когато те възникнат по време на изпълнение.

Описаните механизми се наричат обработка на изключения - **EXCEPTION HANDLING**.

EH – Basic Concepts

Изключения са:

- Грешки, хардуерно открити (as disk read errors)
- Необичайни хардуерно открити ситуации (as end-of-file).
- Грешки и необичайни ситуации, които се регистрират софтуерно.

Exceptions

Interrupts

EH – термини

- **Exception** – необичайно събитие (грешка или не), регистрирано хардуерно или софтуерно, което налага специално третиране-processing.
- **Exception Handling** – специална обработка, която се налага при поява на изключение - exception.
- **Exception Handler** – програмен код, който извършва специална обработка, т.е. exception handling (манипулатор на изключение).
- Изключение се вдига (Exception is **raised**), когато настъпи съответно събитие.

Предимства от ЕН

- Без ЕН, кодът нужен за откриване на грешка(и), потенциално объркава и задръства програмата.
- ПЕ, поддържащ ЕН, стимулира програмисти да осмислят всички възможни събития, които биха настъпили при изпълнение и да предвидят още по време на етап проектиране програмна реакция на тези събития без да се излиза от програмата, която се изпълнява.
- Така се създава надежден (**solid , reliable**) код.

Design Issues

ЕН система е присъща на всеки модерен ПЕ. Следва да се поддържат:

- Вградени Built-in exceptions и EHandlers
- Потребителски User-defined exceptions и EHandlers

Вж сл. Слайд за основната идея.

// typical C-like program unit without
// Exception Handling support
// there is a dangerous division operator specified

```
void example()
  {
     . . .
   average = sum / total;
   . . .
   return;
  } // end of program unit example
```

// typical C-like program unit with Exception Handling support

```
void example()
 {
    . . .
  average = sum / total;
  . . .
  return;
  // exception handlers
  when zero_divide
  {
   average=0;
   cout<<"\nErr:divisor(total) is zero";
  }
  // other exception handlers
 }  // end of program unit example
```

ЕН

- Как се конструира (форма) на ПП за обработка на изключения
- Как се обвързва настъпило събитие към ПП за обработка на изключения
- Как продължава изпълнение след като завърши работа ПП за обработка на изключения

EH поток на управление

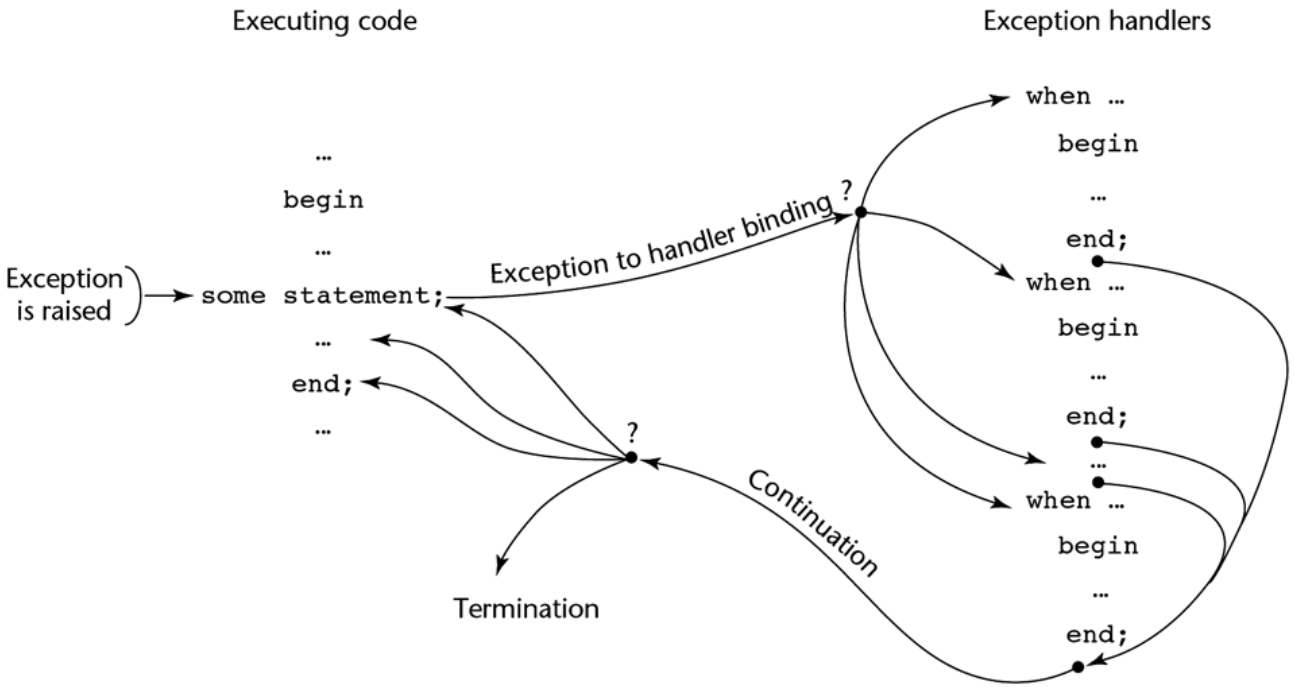


Figure 14.1

Exception-handling control flow

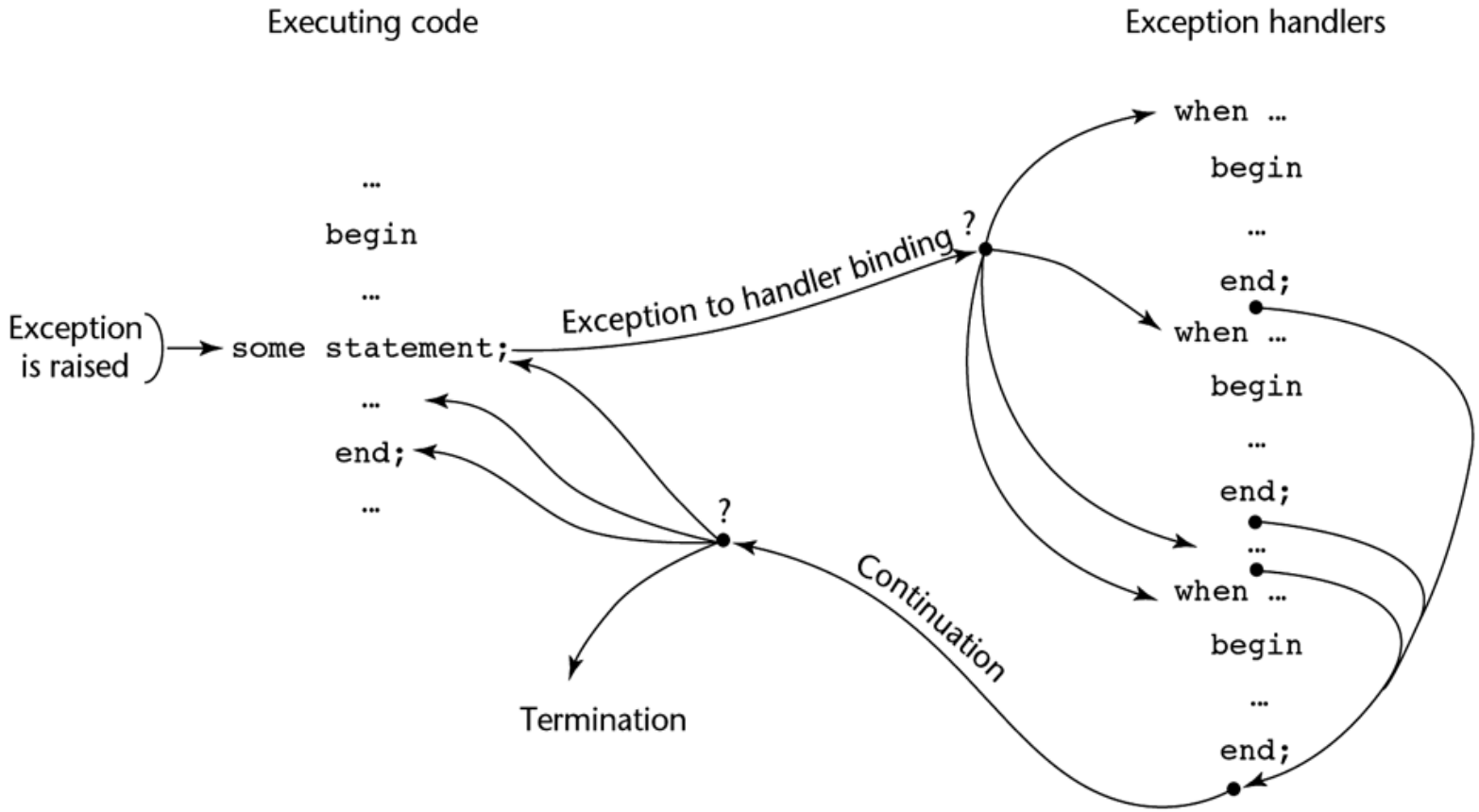


Figure 14.1

Exception-handling control flow

ЕН при Fortran

- READ(5,10) A,B
- READ(5,10,END=100,ERR=20)A,B

EH при PL/1

ON <condition> <statement>;

CONDITION <exception name>;

SIGNAL(<exception name>);

EH при BASIC

On Error GoTo <label>

On Error GoTo 0

On Error Resume Next

Resume Next

Resume

Resume<label>

EH при Pascal

```
{$I-}
  readln(x);
{$I+}
IF IOResult<>0 then        begin
                    writeln('IO Error …');
                    halt;
                         end;
```

EH при C

```
in = fopen(“IN.TXT”, “rt”);

if ((in = fopen(“IN.TXT”, “rt”))==NULL)
                {
                printf(“Can’t open file”);
                exit(1);
                }
```

EH при Ada

```
begin
        { the block of unit body }
exception
  when    exception1    =>
                                first handler
  when    exception2    =>
                                second handler
  other handlers
end
```

ЕН при C++, Java, C#

- ЕН се базира на синтактична конструкция **try**, която определя обсега (зоната) на ПП за ЕН
- Една конструкция **try** включва:

  съставен оператор, наречен try клауза;

  следван от ПП-а(и) за обработка на ЕН.

Съставният **try** оператор определя обсега (зоната) на действие (активност) на ПП за ЕН.

```
try {
 … // код, който се очаква да вдигне или
 … //        предизвика събитие/изключение
}
 catch(<formal parameter>) {
 … // код – EHandler тяло
}
 …
catch(<formal parameter>) {
 … // код – EHandler тяло
}
```

Всяка една catch ф-ия е ПП за ЕН (ehandler).

Catch функците имат един формален параметър със синтаксис на стандартна C/C++ функция.

Формалният параметър може да е само тип или тип, следван от идентификатор.

Когато се предава инфо за изключението на ПП ehandler, параметърът съдържа име на var.

Когато не се предава инфо за изключението на ПП ehandler, параметърът съдържа само тип и той служи за еднозначно разпознаване на ПП ehandler (uniquely identifiable).

Привързване на изключения към ПП EHandlers

C++ изключения се активират чрез явен оператор **throw**, в една от следните форми:

```
throw <израз>;
throw ;
```

Типът на throw израза обвързва/избира/ ПП ehandler със съвпадащ тип на формалния параметър.

Throw без операнд се ползва само в тялото на ПП ehandler за ре-активиране (повторно) на изключението и неговата обработка на друго място.

Продължение след обработка на изключение

След като ПП ehandler завърши, управлението се предава на първия оператор след конструкцията try.

Ако не се намери подходяща ПП ehandler на локално ниво, изключението се предава за обработка на модула (caller) на функцията, в която изключението е активирано.

Ако не се намери ПП ehandler в цялата програма, изпълнението й се прекратява (terminated).

Примери с EH при C++

Пример 1:

Source text: SBExcept1.cpp

Executable code: SBExcept1.exe

```
void main()
{
int val, sum = 0;      short int eofcondition=-55;

try          {
       cout << "\n Enter integer value:";
      while( 1 )
             {
              if ( !(cin >> val) )  throw eofcondition;
              sum = sum + val;
             cout << "\n\n Enter new integer value or CTRL/Z to end:";
             }
    }
catch (short int) // handler for end of input data   // only type specifier
    {
      cout << "\n END OF INPUT DATA ENCOUNTERED";
      cout << "\n\n Total sum accumulated is " << sum;
    }

cout << "\n\n END OF PROGRAM\n";
}
```

Примери с EH при C++

Пример 2:

Source text: SBExcept2.cpp

Executable code: SBExcept2.exe

```
void main()
{     int val, sum1 = 0,  sum2 = 0,  sum3 = 0;        short int eofcondition = 0;
      try  {
            cout << "\n Enter integer value:";
            while( 1 )
            {
                  if ( !(cin >> val) )  throw eofcondition;
                  { // inner try construct
                        try {
                              if ( val>=20 && val<= 100 ) throw(val);
                        }
                        catch (int valparam)
                        {
                              sum2 = sum2 + valparam;
                              if (valparam==22) sum3 = sum3 + valparam;
                        }

                  } // end of inner try construct
                  sum1 = sum1 + val;
                  cout << "\n\n Enter new integer value o CTRL/Z to end:";
            }
        }
      catch (short int) // handler for end of input data,  only type specifier
        {     cout << "\n\n Total sum accumulated is " << sum1;
              cout << "\n\n Totall sum of values in range 20-100 is "<< sum2;
              cout << "\n\n Total sum of values 22 is " << sum3;
        }
}
```

Примери с ЕН при C++

Пример 3:

Source text: SBExcept3.cpp

Executable code: SBExcept3.exe

```
void main()
{       int val, pom[20]={0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        short int eofcondition=-55;
        try   {
                    cout << "\n Enter integer value:";
                    while( 1 )
                    {
                      if ( !(cin >> val) )  throw eofcondition;
                        { // inner try construct
                            try {
                                    if ( val<0 || val>19) throw(val);
                                    pom[val] ++;
                                }
                             catch (int valparam)
                                {
                                    cout << "\n\n Array subscript range exception raised";
                          cout << "\n  You entered value " << valparam << "  which is not in range 0 - 19";
                                    cout << "\n  The value you entered is ignored";
                                 }
                            } // end of inner try construct
                           cout << "\n\n Enter new integer value or CTRL/Z to end:";
                    }
              }
        catch (short int) // handler for end of input data, only type specifier
            {     cout << "\n Total result of input data\n";
                  int i;  for (i=0; i<=19; i++) cout << pom[i]<< "  ";
            }
```

Примери с ЕН при C++

Пример 4:

Source text: SBExcept4.cpp

Executable code: SBExcept4.exe

```
void main()
{       char val1, val2;        short int eofcondition=-55;
        try   {
                cout << "\n Enter character value:";
                while( 1 )
                {
                  if ( !(cin >> val1) )  throw eofcondition;
                   { // inner try construct
                      try {
                            if (val1<'a' || val1>'z') throw(val1);
                            val2 = val1 - ('a' - 'A'); // lower case to upper case
                          }
                      catch (char val1param)
                          {
                            cout << "\n\nYou entered the '"<<val1param<<
                                                "' that can't be converted to upper case letter";
                            cout << "\nCharacter replaced with '@'";
                            val2 = '@';
                          }
                   } // end of inner try construct
                            cout << "\nEntered char:"<<val1<<"     Converted char:"<< val2;
                }
            }
        catch (short int) // handler for end of input data,  only type specifier
            {
                cout << "\n END OF INPUT DATA ENCOUNTERED";
            }
}
```

Примери с ЕН при C++

Пример 5:

Source text: SebestaExcept.cpp

Executable code: SebestaExcept.exe

```
void  main(){
   int new_grade, index, limit_1, limit_2,      freq[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
   short int eof_condition=-55;
   try    {
    while (1)     {
      new_grade = 400;
      if (!(cin >> new_grade)) throw eof_condition;     // cin detects eof
      index = new_grade/10;
      {
        try  {
                if (index<0 || index>9) throw (new_grade);
                freq[index]++;
              }
        catch(int grade)  // handler of index errors
              {
                if (grade==100) freq[9]++;
                else cout <<"\nError-new grade "<<grade<< " is out of range";
              }
       } // end of inner try compound
     } // end of while(1)
  }  // end of outer try compound
  catch(short int) // handler for eof condition
  {   cout << "\n\n\nFrom\tTo\tFrequence";   cout << "\n=========================";
      for (index=0; index<10; index++)
     {limit_1=10*index; limit_2 = limit_1 +9;       if(index==9) limit_2 = 100;
       cout << "\n"<<limit_1<<"\t"<<limit_2<<"\t"<<freq[index];
      }
  } // end of this catch
}  // end of main
```

Част 2

Изключения и грешки, генерирани от C++ класове

ЕН преди ерата на ООР

- Си програми сигнализират грешка или особен случай чрез връщане на специална стойност от функцията, в която се вдига изключението.
- Всяко обръщение към функция следва да проверява връщаната стойност.

- Този програмен фрагмент

```
func1();
 // proceed normally
func2();
 // proceed normally
func3();
 // proceed normally
func4():
 // proceed normally
```

- Следва да се замести с (вж сл. слайд)

```
if ( func1() == ErrRetVal )
      // handle the error
 else
      // proceed normally

 if ( func2() == NULL )
      // handle the error
 else
      // proceed normally

 if ( func3() == ErrRetVal )
      // handle the error
 else
      // proceed normally
```

Проблеми с ЕН стар стил

Проблем 1: всяко обръщение към функция се загражда с if … else statement и оператори за обработка на грешки, а това прави кода тромав и труден за проследяване.

Проблеми с ЕН стар стил

Проблем 2: При ООР конструктори се викат неявно и нямат return values. How will the program find out if error occurred in class constructor? Няма начин да се вградят **if** statements за проверка на value returned by the constructor

Проблеми с ЕН стар стил

Проблеми като изброените предполагат друг, нов подход за третиране на ЕН.

Exception Синтаксис

- Илюстрация

Изключение – идеологически пример

```
class AClass
{
  public:    class AnError // exception class
                          {
                          };
             void Func()  // member f unction
                          {
                            if (error cond) throw AnError();
                          }
};
void main()               // application
{
  try
     {
       AClass obj1;  // interact with AClass objects
        obj1.Func();  // may cause error
     }
 catch (AClass::AnError)   // exception handler
     {
     // tell user about error
     }
}
```

Първи пример
Simple Working Example

.

EH с класове в C++

Демо програма:

Source text: SBExcept5.cpp

Executable code: SBExcept5.exe

Първи пример 1/2

```
const int MAX = 3;
class Stack
{
private: int st[MAX];          int top;
public:
     class Range  // exception class for Stack
     {                                  // note: empty class body
     };
     Stack() { top = -1; }
     void push(int var)
     {
              if (top >= MAX-1) throw Range();
              st[++top] = var;
     }
     int pop() {
                 if (top < 0) throw Range();
                  return st[top--];
     }
};
```

Първи пример 2/2

```
void main()
{
    Stack c;
    try
    {
        c.push(11); c.push(22);  c.push(33); c.push(44);
        cout << endl << "1: " << c.pop();
        cout << endl << “2: " << c.pop();
        cout << endl << “3: " << c.pop();
        cout << endl << “4: " << c.pop();
    }
    catch(Stack::Range)
    {
        cout << endl << "Exception: Stack full or empty";
    }
    cout << endl << "Arrive here after catch or normal exit ";
}
```

Спецификация на Exception Class

```
// exception class for Stack
class Range
  {
  // note: empty class body
  };
```

Throwing the Exception

```
if (top >= MAX-1) throw Range();
```

```
if (top < 0) throw Range();
```

try Блок

```
try
   {
   // code that operates on objects that
   // might cause an exception
   }
```

```
try
   {
   c.push(11); c.push(22);  c.push(33); c.push(44);
   cout << endl << "1: " << c.pop();
   cout << endl << "2: " << c.pop();
   cout << endl << "3: " << c.pop();
   cout << endl << "4: " << c.pop();
   }
```

EHandler (catch блок)

```
catch(Stack::Range)
{
    // code that handles the exception
}
```

```
catch(Stack::Range)
{
cout << "\nException: Stack full or empty";
}
```

Последователност/верига/ от събития

- Code is executing normally outside a try block.
- Control enters the try block.
- A statement in the try block causes an error in a member function.
- The member function throws an exception.
- Control transfers to the exception handler.

Multiple Exceptions

Един клас може да поддържа произволен брой />1/ изключения

EH с класове в C++

Демо програма:

Source text: SBExcept6.cpp

Executable code: SBExcept6.exe

Multiple Exceptions (1/2)

```
#include <iostream>
using namespace std;
const int MAX = 3;
class Stack
{
private: int st[MAX]; int top;
public:
      class Full {};  // exception class for Stack
      class Empty{};  // note: empty class body

      Stack() { top = -1; }
      void push(int var)
      {
            if (top >= MAX-1) throw Full();
            st[++top] = var;
      }
      int pop()
      {
            if (top < 0) throw Empty();
            return st[top--];
      }
};
```

Multiple Exceptions (2/2)

```
void main()
{
	Stack c;
	try
	{
		c.push(11); c.push(22);  c.push(33); //c.push(44);
		cout << endl << "1: " << c.pop();
		cout << endl << "2: " << c.pop();
		cout << endl << "3: " << c.pop();
		cout << endl << "4: " << c.pop();
	}
	catch(Stack::Full)
	{
		cout << endl << "Exception: Stack full";
	}
	catch(Stack::Empty)
	{
		cout << endl << "Exception: Stack empty";
	}
	cout << endl << "Arrive here after catch or normal exit ";
}
```

Изключения и клас Distance

Distance клас

- Проблем: inches следва да са <12.0
- Въвежда се нормализация на обекти от клас Distance
  - `Distance d1(10, 14.5);`
  - `Distance d2;  d2.GetDist();`

EH с класове в C++

Демо програма:

Source text: SBExcept7.cpp

Executable code: SBExcept7.exe

Изключения и клас Distance 1/2

```
class Distance
{
private: int feet;  float inches;
public:
     class InchesEx {};  // exception class for Distance
     Distance() { feet=0; inches = 0.0; }
     Distance(int ft, float in)  {
                                  if (in >= 12.0) throw InchesEx();
                                  feet = ft;
                                  inches = in;
                                 }
     void GetDist()  {
                           cout <<"\nEnter feet:"; cin >> feet;
                           cout <<"\nEnter inches:"; cin >> inches;
                           if (inches >= 12.0) throw InchesEx();
                        }
     void ShowDist()                {
              cout << feet << "\'-" << inches << '\"';
                           }
};
```

Изключения и клас Distance 2/2

```
void main()
{
    try
    {
    Distance d1(17, 4.5);
    Distance d2;
    d2.GetDist();

    cout << "\n d1 = "; d1.ShowDist();
    cout << "\n d2 = "; d2.ShowDist();
    cout << endl;
    }
    catch(Distance::InchesEx)
    {
            cout << endl << "Initialization error: inches value >= 12.0";
    }
    cout << endl;
}
```

Изключения с Аргументи

Изключения с Аргументи

- Как да се процедира, ако се налага да предадем инфо към EH handler
- За примера с класа Distance:
  - Колко е неправилната стойност за inches
  - Откъде идва изключението: 2-arg constructor или GetDist()
- How to pass such an info from the member function where the exception was thrown to the application that catches it
  - Throwing an exception involves not only control transfer to the EHandler but also creating an object of the exception class by calling its constructor

Изключения с Аргументи

Решението е да е дефинира exception class който:

- Не е празен
- Е регулярен с даннови компоненти и конструктор(и)

EH с класове в C++

Демо програма:

Source text: SBExcept8.cpp

Executable code: SBExcept8.exe

Изключения с Аргументи 1/2

```
class Distance
{
private: int feet;  float inches;
public:
      class InchesEx    // exception class for Distance
      {
      public: string origin; // for name of routine
                  float ivalue;  // for faulty inches value
                  InchesEx(string or, float in)  {          origin = or;  ivalue = in;  }
      };
      Distance() { feet=0; inches = 0.0; }
      Distance(int ft, float in) {
                  if (in >= 12.0) throw InchesEx("2-arg constructor", in);
                  feet = ft;
                  inches = in;
      }
      void GetDist()        {
                  cout <<"\nEnter feet:"; cin >> feet;
                  cout <<"\nEnter inches:"; cin >> inches;
                  if (inches >= 12.0) throw InchesEx("GetDist() method", inches);
      }
      void ShowDist()      {  out << feet << "\'-" << inches << '\"';
};
```

Изключения с Аргументи 2/2

```
void main()
{
	try
	{
	Distance d1(17,44.5);
	Distance d2;
	d2.GetDist();

	cout << "\n d1 = "; d1.ShowDist();
	cout << "\n d2 = "; d2.ShowDist();
	cout << endl;
	}
	catch(Distance::InchesEx ix)
	{
		cout << endl << "Initialization error in: "<< ix.origin;
		cout << endl << ix.ivalue << " inches value >= 12.0";
	}
	cout << endl;
}
```

Задаване на член данни в Exception клас

```
class InchesEx    // exception class for Distance
   {
   public: string origin; // for name of routine
           float ivalue;  // for faulty inches value
           InchesEx(string or, float in)
               {
               origin = or;
               ivalue = in;
              }
   };
```

Инициализация на Exception обект

- в 2-arg конструктор

```
if (in >= 12.0)
throw InchesEx("2-arg constructor", in);
```

- В метод GetDist()

```
if (inches >= 12.0)
throw InchesEx("GetDist() method", inches);
```

Извличане на данни от Exception Обект

```
catch(Distance::InchesEx ix)
    {
    // access ix.origin and ix.ivalue directly
    cout << endl << "Initialization error in: "<< ix.origin;
    cout << endl << ix.ivalue << " inches value >= 12.0";
    }
```

Standard C++ съдържа вградени Exception класове

Класът bad_alloc

C++ вграден exception клас

- Типична употреба на този клас изключение bad_alloc class за обработка на грешки при заделяне на памет от heap memory с операция **new**.

bad_alloc Exception клас

```
#include <iostream>
using namespace std;
 int  main()
{
      const unsigned long SIZE = 10000;
      char* ptr;
      try
      {
               ptr = new char[SIZE];
      }
      catch(bad_alloc)
      {
               cout << "\nBad_alloc exception. Can't allocate memory.";
                  return  1;
      }
delete[] ptr;
cout << "\n memory use is successful";
 return 0;
}
```

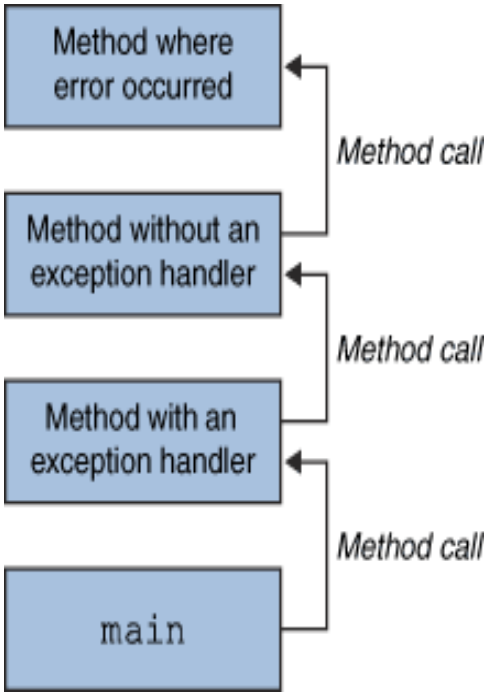
Част 3

Изключения и грешки, генерирани от Java класове

After a method throws an exception, the runtime system attempts to find some code to handle it.

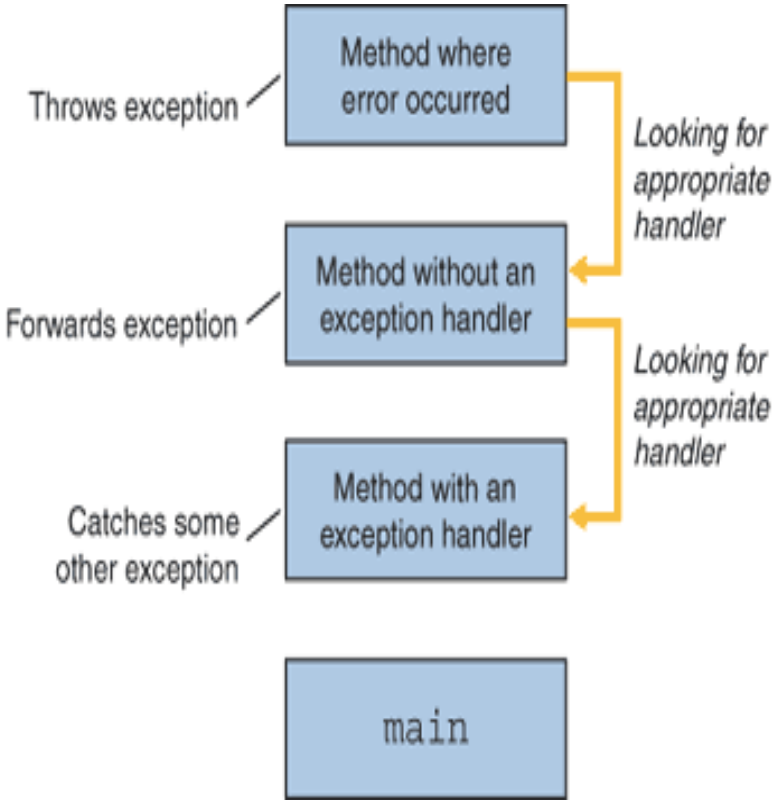
The set of possible code to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred.

The list of methods is known as the *call stack*



The exception handler chosen is said to *catch the exception*.

If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the runtime system (and, consequently, the program) terminates.



Exception-Handling Overview

- To demonstrate EH, including how an exception object is created and thrown:
- Open Quotient.java that reads two integers and displays their quotient.
- Run program with regular data
- Run program with illegal data (integer zero div)
- FYI: FP real value divided by zero does not raise an exception.

Exception-Handling Overview

- Pure, naked, risky assignment statement

- ```
  quotient = dividend / divisor;
  ```


  ```

Exception-Handling Overview

- Simple way to fix the problem is to add if stmt to test the divisor:
- Open QuotientWithIf.java that reads two integers and displays their quotient.
- Run program with regular data
- Run program with illegal data (integer zero div)

.

Exception-Handling Overview

- User specified if statement

```
if (divisor != 0) {
      quotient = dividend / divisor;
      System.out.println(dividend + " / " + divisor + " is = " + quotient);
      }
      else
            System.out.println("Divisor cannot be zero ");
```

Exception-Handling Overview

- To demonstrate EH, including how to create, throw, catch and handle an exception object:
- Open QuotientWithExcepionVerLiang.java that reads two integers and displays their quotient.
- Run program with regular data
- Run program with illegal data (integer zero div)

Exception-Handling Overview

- Explicitly wiritten code to throw an exception

```
try {
        …

        if (divisor == 0)
          throw new ArithmeticException("Divisor cannot be zero ");

        quotient = dividend / divisor;
        System.out.println(dividend + " / " + divisor + " is = " + quotient);
      }  // end of try
catch (ArithmeticException aeRef) {
        System.out.println(" Exception: annnn integer cannot be divided by zero ");
        System.out.println(aeRef.getMessage());
        aeRef.printStackTrace(); // print call stack
        }
catch (InputMismatchException imeRef) {
        System.out.println(" Exception " + imeRef.toString() );
        }
finally {
        System.out.println("\n\n Finally block executed");
      }
```

Exception-Handling Overview

- To demonstrate EH, including how to create, throw, catch and handle an exception object:
- Open QuotientWithExcepionVerMalik.java that reads two integers and displays their quotient.
- Run program with regular data
- Run program with illegal data (integer zero div)

Exception-Handling Overview

- No explicit code to throw an exception
- Run-time system creates exception and throws it
- try {

```
        …
        quotient = dividend / divisor;
        System.out.println(dividend + " / " + divisor + " is = " + quotient);
    }  // end of try
```

- catch (ArithmeticException aeRef) {

```
        System.out.println(" Exception: ann integer cannot be divided by zero ");
        System.out.println(aeRef.getMessage());
        aeRef.printStackTrace(); // print call stack
        }
catch (InputMismatchException imeRef) {
        System.out.println(" Exception " + imeRef.toString() );
        }
finally {
        System.out.println("\n\n Finally block executed");
    }
```

Exception-Handling Advantages

QuotientWithMethod

- To demonstrate EH advantages:
- Open QuotientWithMethod.java that reads two integers and displays their quotient.
- Run program with regular data
- Run program with illegal data (integer zero div)
- Now you see the *advantages* of using exception handling. It enables a method to throw an exception to its caller. Without this capability, a method must handle the exception or terminate the program.

Exception-Handling Advantages

- Exception explicitly raised in one method
- Exception caught in another method
- public static int quotient(int num1, int num2) throws ArithmeticException {

```
    if (num2 == 0)
          throw new ArithmeticException("Divisor cannot be zero ");
        return num1/num2;
   }
```

- public static void main(String[] args)

```
   {
    try {
      result = quotient(dividend, divisor);
      System.out.println(dividend + " / " + divisor + " is = " + result);
    }  // end of try
    catch (ArithmeticException aeRef) {
       System.out.println(" Exception: an integer cannot be divided by zero ");
       System.out.println(aeRef.getMessage());
       aeRef.printStackTrace(); // print call stack
       }
    catch (InputMismatchException imeRef) {
       System.out.println(" Exception " + imeRef.toString() );
       }
```

Exception-Handling Advantages

- Exception implicitly raised in one method
- Exception caught in another method
- public static int quotient(int num1, int num2) throws ArithmeticException {

```
    //if (num2 == 0)
     //     throw new ArithmeticException("Divisor cannot be zero ");
        return num1/num2;
   }
```

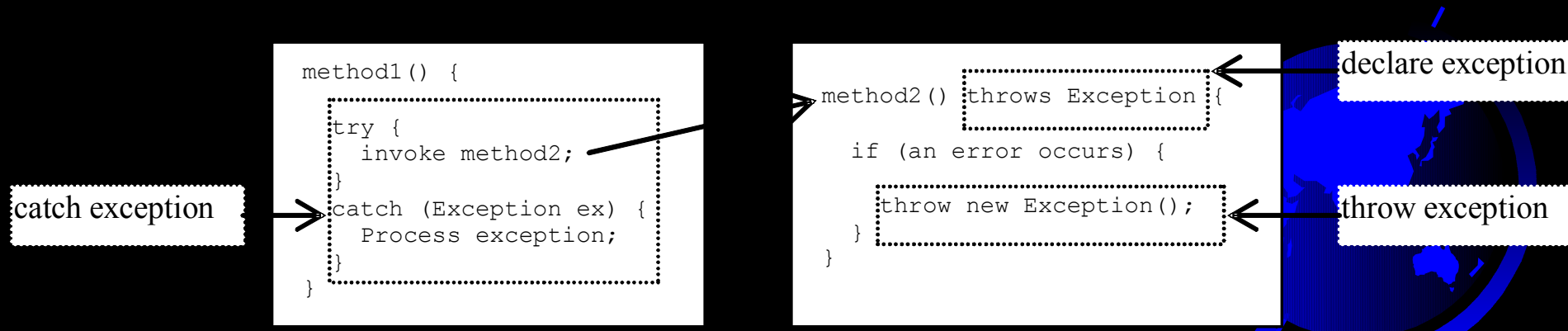
- public static void main(String[] args)

```
   {
    try {
      result = quotient(dividend, divisor);
      System.out.println(dividend + " / " + divisor + " is = " + result);
    }  // end of try
    catch (ArithmeticException aeRef) {
       System.out.println(" Exception: an integer cannot be divided by zero ");
       System.out.println(aeRef.getMessage());
       aeRef.printStackTrace(); // print call stack
       }
    catch (InputMismatchException imeRef) {
       System.out.println(" Exception " + imeRef.toString() );
       }
```

Declaring, Throwing, and Catching Exceptions

Java's EH model is based on three operations:

- declaring an exception
- throwing an exception
- catching an exception



Catching Exceptions pp465

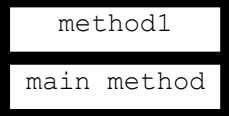
```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

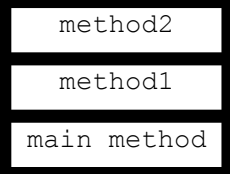
```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

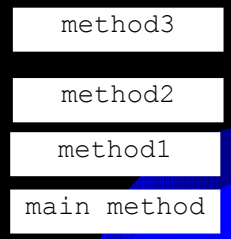
```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

An exception is thrown in method3

Call Stack







Example: Declaring, Throwing, and Catching Exceptions

- Objective: This example demonstrates declaring, throwing, and catching exceptions by modifying the setRadius method in the Circle class. The new setRadius method throws an exception if radius is negative.
- Open CircleWithException.java file

TestCircleWithException

CircleWithException

Example: Declaring, Throwing, and Catching Exceptions

```
class Circle {
  …
  // methods mutators
  public void setRadius(double par) throws IllegalArgumentException {
    if(par >= 0.0) radius = par;
   else throw new IllegalArgumentException("Radius cannot be negative");
   }
  …
 } // end of class Circle

public class CircleWithException {
   public static void main(String[] args) {
     try {
      Circle a = new Circle();
      Circle b = new Circle(5.);
      Circle c = new Circle(-5.);
     }
     catch (IllegalArgumentException ex){
       System.out.println("\n-->>" + ex + "\n-->>");
       System.out.println("\n-->>" + ex.getMessage() + "\n-->>");
       ex.printStackTrace(); // print call stack
       }
     finally {
       System.out.println("\n\n Finally block executed");
     }


  } // end of method main()

} // end of class
```

Defining Custom Exception Classes

- Use the exception classes in the API whenever possible.
- Define custom exception classes if the predefined classes are not sufficient.
- Define custom exception classes by extending Exception or a subclass of Exception.
- the setRadius method throws an exception if the radius is negative. Suppose you wish to pass the radius to the handler, you have to create a custom exception class.
- Open CircleWithUserException.java file

Defining Custom Exception Classes

```
class IllegalRadiusException  extends  Exception  {
   private double rad;
      public IllegalRadiusException(double radius) { // constructor
      super("Invalid radius " + radius);
      rad = radius;
      }
    public double getRad() { return rad; }
}
class Circle {
   // methods mutators
   public void setRadius(double par) throws IllegalRadiusException {
      if(par >= 0.0) radius = par;
      else throw new IllegalRadiusException(par);
    }
 } // end of class Circle
public class CircleWithUserException {
    public static void main(String[] args) {
       try {
        Circle a = new Circle(88.);   //a.setRadius(-22.);
        Circle b = new Circle(5.);    //  Circle c = new Circle(-5.);
       }
       catch (IllegalRadiusException ex){
         System.out.println("\n-->>\nInvalid radius is " + ex.getRad() + "\n-->>");
         }
    } // end of method main()
} // end of class
```

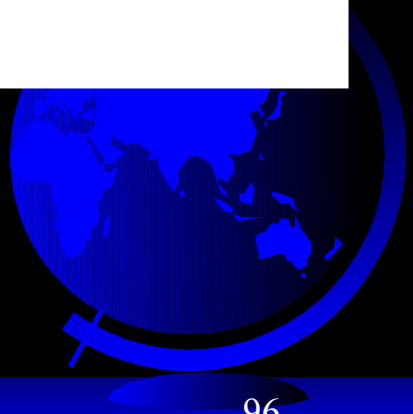
```
try {
    // normal code here!
}

catch(Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace(); // print call stack
}
```

- getMessage() is a method of the Throwable class that returns the error message describing the exception.
- printStackTrace() is also a method of the Throwable class

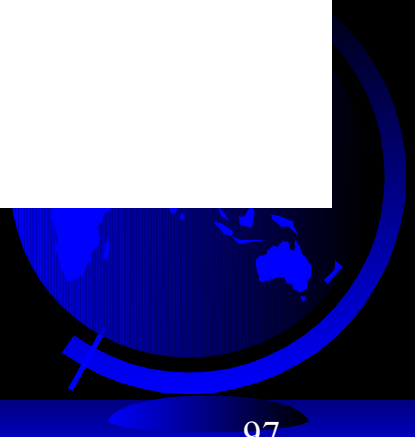
Rethrowing Exceptions

```
try {
   statements;
}
catch(TheException ex) {
   perform operations before exits;
   throw ex;
}
```



The `finally` Clause

```
try {
   statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}
```



Cautions When Using Exceptions

- Exception handling separates error-handling code from normal programming tasks, thus making programs easier to read and to modify. Be aware, however, that exception handling usually requires more time and resources because it requires instantiating a new exception object, rolling back the call stack, and propagating the errors to the calling methods.

When to Throw Exceptions

- An exception occurs in a method. If you want the exception to be processed by its caller, you should create an exception object and throw it. If you can handle the exception in the method where it occurs, there is no need to throw it.

When to Use Exceptions

When should you use the try-catch block in the code? You should use it to deal with unexpected error conditions. Do not use it to deal with simple, expected situations. For example, the following code

```
try {
   System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
   System.out.println("refVar is null");
}
```

When to Use Exceptions

is better to be replaced by

```
if (refVar != null)
  System.out.println(refVar.toString());
else
  System.out.println("refVar is null");
```

Благодаря
За
Вниманието