Universitatea Politehnica Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Tema: Programarea orientata pe obiect din perspectiva ingineriei software

Subtema: Tratarea exceptiilor

Pruiu Ana-Maria

Grupa 441A

Cuprins:

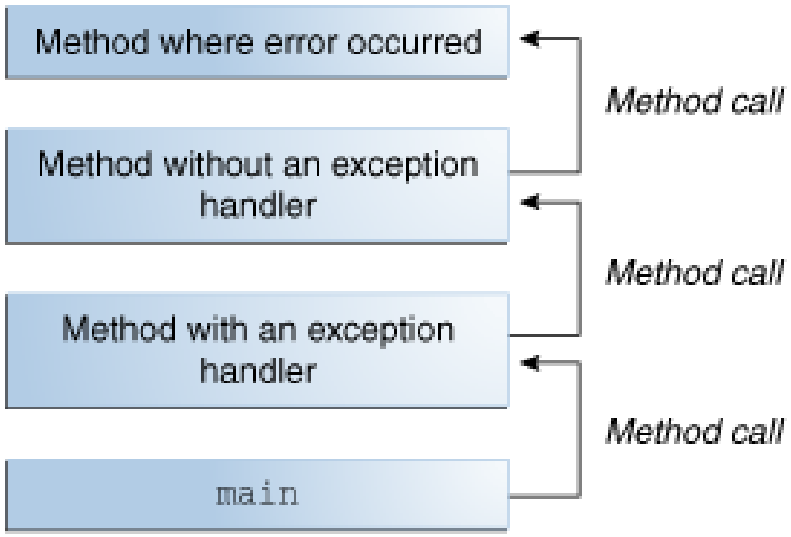
Tratarea exceptiilor

1. Introducere [8]

Notiunea de exceptie este o prescurtare pentru "eveniment exceptional" si se poate defini in felul urmator: o exceptie este un eveniment ce se produce în timpul executiei unui program si provoaca întreruperea cursului normal al executiei programului.

Atunci cand apare o eroare in interiorul unei metode, metoda creeaza un obiect si il trimite la sistemul de operare. Obiectul, numit “obiect exceptie” contine informatia despre eroare, incluzand tipul acesteia si starea programului la momentul cand a aparut eroarea. Actiunea de a crea un obiect exceptie si de a-l trimite la sistemul de operare se numeste “lansarea unei exceptii” (throwing an exception).

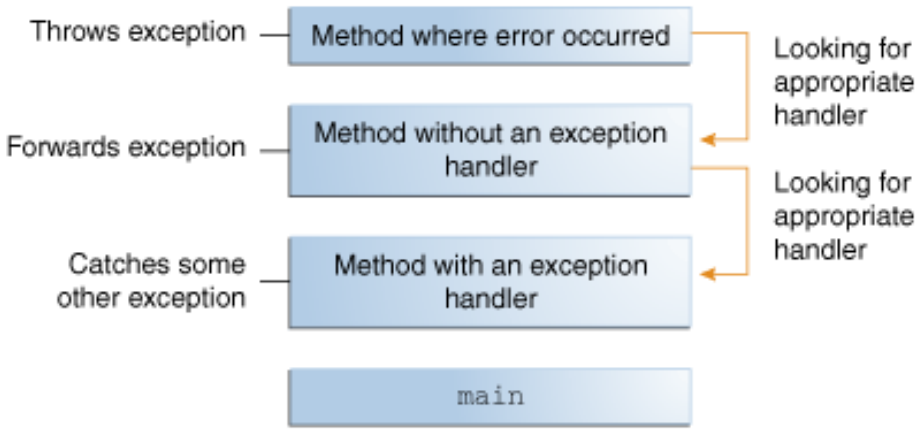
Dupa ce a fost lansata o exceptie, sistemul de operare incearca sa gaseasca ceva care sa rezolve exceptia. Variantele posibile care ar putea sa rezolve exceptia sunt ordonate intr-o lista de metode care a fost apelata pentru a ajunge la metoda in care a aparut eroarea. Aceasta lista de metode este cunoscuta sub numele de “stiva de apeluri” si arata astfel:



Sistemul de operare cauta in stiva de apeluri metoda care contine un bloc al codului care poate solutiona exceptia. Acest bloc de cod este numit “manipulator de exceptii” (exception handler). Cautarea incepe cu metoda in care a aparut eroarea si continua explorarea stivei de apeluri in ordine inversa fata de cum au fost apelate metodele. Cand este gasit un handler apropiat sistemul de operare paseaza exceptia handler-ului. Un handler de exceptii este considerat

apropiat daca tipul exceptiei aruncate se potriveste cu tipul care poate fi manipulate de handler.

Handler-ul de exceptii ales este cel care rezolva exceptia (“catch the exception”). Daca sistemul de operare cauta exhaustiv toate metodele in stiva de apeluri fara sa gaseasca un handler de exceptii apropiat, asa cum este reprezentat in figura urmatoare, sistemul de operare (si in consecinta si programul) se inchide.



2. Tratarea exceptiilor in C++:

In C++ este implementat un mecanism foarte simplu de tratare a exceptiilor. Putem spune ca o exceptie este un obiect a carui adresa este trimisa din zona de cod unde a aparut eroarea intr-o alta zona de cod care trebuie sa rezolve aceasta eroare. [3]

Pentru tratarea exceptiilor in C++ se urmeaza pasii [3]:

- Se gasesc acele zone din program in care se efectueaza o operatie despre care se stie ca ar putea genera o exceptie si apoi se aceste zone se marcheaza in cadrul unui bloc de tip try. In cadrul unui bloc de tip try se testeaza conditia de aparitie a exceptiei si in caz pozitiv se semnaleaza aparitia exceptiei prin intermediul cuvantului cheie “throw”.
- Se creeaza blocuri de tip catch pentru a prelua exceptiile atunci cand acestea sunt gasite.

Semnificatia cuvintelor cheie este [4]:

- **Try** delimiteaza o protiune de cod in care se instituie controlul sistemului asupra exceptiilor in timpul rularii.

- **Throw** lanseaza o exceptie de un anumit tip
- **Catch** preia o exceptie lansata

Sintaxa pentru try [6]:

```
try
{
// cod
throw TipExceptie;
}
```

Sintaxa pentru throw [6]:

```
throw TipExceptie;
```

Sintaxa pentru catch [6]:

```
catch(TipExceptie)
{
// cod tratare exceptie
}
```

Un bloc “try” poate fi urmat de unul sau mai multe blocuri “catch”. Daca exceptia corespunde cu una din declaratiile de tratare a exceptiilor,atunci aceasta este apelata. Daca nu exista definita nici o rutina de tratare a exceptiei, este apelata rutina predefinita, care incheie executia programului in curs. Dupa ce rutina este executata, programul continua cu instructiunea imediat urmatoare blocului *try*.[5]

Un exemplu de cod este urmatorul:[5]

```
#include <iostream>
using namespace std;

int main () {
  try
  {
    throw 20;
  }
  catch (int e)
  {
    cout << "An exception occurred. Exception Nr. " << e << endl;
  }
  return 0;
}
```

Codul de sub handler-ul de exceptie este inclus in blocul try. In exemplul de mai sus acest cod este pur si simplu aruncat intr-o exceptie:

```
throw 20;
```

Un alt exemplu marcheaza prezenta unei exceptii de tip intreg. La executie se poate observa ca dupa ce eroarea a fost semnalata prin “throw”, instructiunile care urmau in blocul “try” nu mai sunt executate si controlul este preluat de blocul “catch”. [6]

```
#include <iostream.h>

int main(){

 cout<<"Intrare in main"<<endl;

 try{
  cout<<"Intrare in blocul try"<<endl;
  throw 10; //lansare exceptie, se preda controlul blocului catch
  cout<<"Instructiune dupa throw"<<endl; //nu se va tipari
 } //terminare bloc try
 catch (int e){ //linie marcata
  cout<<"Exceptia tratata este:"<<e<<endl;
 }
 cout<<"Terminare main"<<endl;
 return 0;
}
```

Iesirea programului de mai sus este:

Intrare in main
Intrare in blocul try
Exceptia tratata este:10
Terminare main

Daca linia marcata se inlocuieste cu : “catch (float e) { } “ atunci programul se va termina anormal deoarece exceptia de tip intreg care este marcata in blocul “try”, nu gaseste un bloc “catch” care sa o trateze.

3. Tratarea exceptiilor in .net:

In .net sunt definite exceptii standard pentru tipurile de erori care pot aparea intr-un program. In .net exceptiile se pot trata intr-o maniera structurata si controlata, aceasta insemnand ca este usurata munca programatorului care mai nu trebuie sa verifice personal daca o operatie este executata cu succes sau nu. Pentru tratarea exceptiilor se folosesc cuvintele cheie “try”,”catch”, “finally”,”throw”. Acestea formeaza un sistem unitar, ceea ce inseamna ca utilizarea uneia dintre ele implica si utilizarea celorlalte.

In blocul “try” se scriu instructiunile care trebuie verificate pentu aparitia erorilor. Daca pe parcursul executiei acestor instructiuni apare vreo exceptie atunci aceasta exceptie este “aruncata” (lansata).[11]

Cu ajutorul lui “catch” programul poate gasi exceptia si o poate trata in concordanta cu restul programului. Instructiunile care sunt cuprinse in blocul “catch” sunt executate doar daca se lanseaza o exceptie.
Instructiunile din blocul “finally” se vor executa intotdeauna.
Un exemplu de tratare a exceptiei de impartire la 0 si de utilizare a blocurilor “try”-“catch”-“finally” este urmatorul:

```
try
{
   int x = 0;
   //raise the exception
   int y = 4 / x;
}

catch
{
   //catch the exception
   Console.WriteLine("X must be greater than zero");
}

finally
{
   //this code will be allways executed
   Console.WriteLine("Program completed");
   Console.Read();
}
```

In general, instructiunea *catch* nu are parametru. Totusi, ii putem adauga parametru in cazul in care vom avea nevoie de accesul la obiectul care reprezinta exceptia. Aceasta situatie este utila pentru furnizarea informatiilor suplimentare despre eroarea produsa.

```
catch (Exception exception)
{
   Console.WriteLine(exception.Message);
   Console.WriteLine(exception.GetType());
}
```

Asociat unui “try” putem avea mai multe instructiuni “catch”, iar instructiunea care se va executa va fi aleasa in functie de tipul exceptiei, toate celelate fiind ignorate.

```
static void Compute(int [] numbers)
{
   try
```

```
    {
        int secondNumber = numbers[0];
        Console.Write(secondNumber);
    }

    catch (IndexOutOfRangeException)
    {
        Console.WriteLine("Must provide more than an argument");
    }

    catch (NullReferenceException)
    {
        Console.WriteLine("Argument is null");
    }

    catch (FormatException)
    {
        Console.WriteLine("Argument is not a number");
    }

    catch (Exception)
    {
        Console.WriteLine("Another exception occured");
    }
}
```

In aceasta situatie se scriu mai intai instructiunile “catch” mai specifice si apoi cele mai generale. CLR-ul cauta instructiunea “catch” care va trata exceptia. Daca metoda curenta in care ne aflam nu are niciun bloc “catch”, atunci CLR-ul va cauta in metoda care a apelat metoda curenta si tot asa mai departe conform stivei de apeluri. Daca totusi nu se gaseste niciun “catch” atunci CLR-ul va afisa un mesaj in care spune ca exceptia nu este tratata (“unhandled exception message”) si apoi se opreste executia programului.

4. Tratarea exceptiilor in Java:

Exceptiile pot aparea din diverse cauze si pot avea nivele diferite de gravitate: de la erori fatale cauzate de echipamentul hardware pâna la erori ce tin strict de codul programului, cum ar fi accesarea unui element din afara spatiului alocat unui vector. In momentul când o asemenea eroare se produce în timpul executiei sistemul genereaza automat un obiect de tip exceptie ce contine:

- informatii despre exceptia respectiva

- starea programului în momentul producerii acelei exceptii

Crearea unui obiect de tip exceptie se numeste aruncarea unei exceptii ("throwing an exception"). In momentul în care o metoda genereaza o exceptie (arunca o exceptie) sistemul de executie este responsabil cu gasirea unei secvente de cod dintr-o metoda care sa trateze acea exceptie. Cautarea se face recursiv, începând cu metoda care a generat exceptia si mergând înapoi pe linia apelurilor catre acea metoda.

Secventa de cod dintr-o metoda care trateaza o anumita exceptie se numeste analizor de exceptie ("exception handler") iar interceptarea si tratarea exceptiei se numeste prinderea exceptiei ("catch the exception").

Cu alte cuvinte la aparitia unei erori este "aruncata" o exceptie iar cineva trebuie sa o "prinda" pentru a o trata. Daca sistemul nu gaseste nici un analizor pentru o anumita exceptie atunci programul Java se opreste cu un mesaj de eroare .

In Java tratarea erorilor nu mai este o optiune ci o constrângere. Orice cod care poate provoca exceptii trebui sa specifice modalitatea de tratare a acestora.

Prin felul in care trateaza Java exceptiile prezinta urmatoarele avantaje fata de mecanismul clasic de tratare a exceptiilor:

- Realizeaza separarea codului pentru a trata o eroare aparuta in codul respective
- Propagarea unei erori pana la un analizor de exceptii corespunzator
- Gruparea erorilor dupa tipul lor

Exemple de exceptii in Java [10]:

- - IOException
- - EOFException
- - ArrayIndexOutOfBoundsException
- - FileNotFoundException
- - InterruptedException.

Aceste exceptii pot fi tratate în Java în mai multe moduri :

- - pot fi ignorate ;
- - pot fi tratate direct în codul în care apar;
- - pot fi transmise codului ce a apelat metoda care a generat exceptia, în ideea ca vor fi tratate de acesta.

Codul de tratare a exceptiilor se afla în pachetul java.lang. Exceptiile sunt tratate cu ajutorul a trei tipuri de instructiuni : try, catch si finally.

In Java, folosind mecansimul exceptiilor, codul de tratare a exceptiilor este:

```
int citesteFisier {
```

```
        try {
                deschide fisierul;
                determina dimensiunea fisierului;
                aloca memorie;
                citeste fisierul in memorie;
                inchide fisierul;
        }
        catch (fisierul nu s-a deschis) {trateaza eroarea;}
        catch (nu s-a determinat dimensiunea) {trateaza eroarea;}
        catch (nu s-a alocat memorie) {trateaza eroarea          }
        catch (nu se poate citi dun fisier) {trateaza eroarea;}
        catch (nu se poate inchide fisierul) {trateaza eroarea;}
}
```

Java permite unei metode sa arunce exceptiile aparute în cadrul ei la un nivel superior, adica functiilor care o apeleaza sau sistemului. Cu alte cuvinte o metoda poate sa nu isi asume responsabilitatea tratarii exceptiilor aparute în cadrul ei:

```
metoda1 {
        try {
                apel metoda2;
        }
        catch (exceptie) {
                proceseazaEroare;
        }
        . . .
}
metoda2 throws exceptie{
        apel metoda3;
        . . .
}
metoda3 throws exceptie{
        apel citesteFisier;
        . . .
}
```

In Java exista clase corespunzatoare tuturor exceptiilor care pot aparea in carul unui program. Aceste exceptii sunt grupate in functie de asemanarile dintre ele intr-o ierarhie de clase.

De exemplu pentru exceptiile legate de operatii de intrare/iesire avem clasa IOException, care recunoaste la randul ei si alte tipuri de exceptii cum ar fi FileNotFoundException, EOFException. La randul ei aceasta clasa se afla intr-o categorie mai larga de exceptii care este clasa Exception. Radacina acestei ierarhii este clasa Throwable. Interceptarea unei exceptii se poate face fie la

nivelul clasei specifice pentru acea exceptie fie la nivelul uneia din superclasele sale, în functie de necesitatile programului:

```
try {
        FileReader f = new FileReader("input.dat");
    //acest apel poate genera exceptie de tipul FileNotFoundException
    //tratarea ei poate fi facuta in unul din modurile de mai jos
}
        catch (FileNotFoundException e) {
    //exceptie specifica provocata de absenta fisierului 'input.dat'
        } //sau
        catch (IOException e) {
    //exceptie generica provocata de o operatie de intrare/iesire
        } //sau
        catch (Exception e) {
            //cea mai generica exceptie - NERECOMANDATA!
        }
```

Tratarea exceptiilor se realizeaza prin intermediul blocurilor de instructiuni **try, catch si finally**. O secventa de cod care trateaza anumite exceptii trebuie sa arate astfel:

```
try {
        Instructiuni care pot genera o exceptie
}
catch (TipExceptie1 ) {
        Prelucrarea exceptiei de tipul 1
}
catch (TipExceptie2 ) {
        Prelucrarea exceptiei de tipul 2
}
. . .
finally {
        Cod care se executa indiferent daca apar sau nu exceptii
}
```

Fie exemplul urmator: citirea unui fisier si afisarea lui pe ecran. Fara a folosi tratarea exceptiilor codul programului ar arata astfel:

```
import java.io.*;
public class CitireFisier {

    public static void citesteFisier() {
        FileInputStream sursa = null; //s este flux de intrare
```

```
                int octet;
                sursa = new FileInputStream("fisier.txt");
                octet = 0;
                //citesc fisierul caracter cu caracter
                while (octet != -1) {
                        octet = sursa.read();
                        System.out.print((char)octet);
                }
                sursa.close();
        }

        public static void main(String args[]) {
                citesteFisier();
        }
}
```

Acest cod va furniza erori la compilare deoarece în Java tratarea erorilor este obligatorie. Folosind mecanismul exceptiilor metoda citesteFisier isi poate trata singura erorile pe care le poate provoca:

```
import java.io.*;
public class CitireFisier {

        public static void citesteFisier() {
                FileInputStream sursa = null; //s este flux de intrare
                int octet;
                try {
                        sursa = new FileInputStream("fisier.txt");
                        octet = 0;
                        //citesc fisierul caracter cu caracter
                        while (octet != -1) {
                                octet = sursa.read();
                                System.out.print((char)octet);
                        }
                catch (FileNotFoundException e) {
                        System.out.println("Fisierul nu a fost gasit !");
                        System.out.println("Exceptie: " + e.getMessage());
                        System.exit(1);
                }
                catch (IOException e) {
                        System.out.println("Eroare de intrare/iesire");
                        System.out.println("Exceptie: " + e.getMessage());
```

```
                        System.exit(2);
                }
                finally {
                        if (sursa != null) {
                                System.out.println("Inchidem fisierul...");
                                try {
                                        sursa.close();
                                }
                                catch (IOException e) {
                                        System.out.println("Fisierul poate fi inchis!");
                                        System.out.println("Exceptie: " +
e.getMessage());
                                        System.exit(3);
                                }
                        }
                }
        }

        public static void main(String args[]) {
                citesteFisier();
        }
}
```

Blocul "try" contine instructiunile de deschidere a unui fisier si de citire dintr-un fisier ambele putând produce exceptii. Exceptiile provocate de aceste instructiuni sunt tratate în cele doua blocuri "catch", câte unul pentru fiecare tip de exceptie.

Bibliografie:

1. “Software Engineering”- Sommerville
2. “Introducere in programarea obiect orientata. Concepte fundamentale din perspectiva ingineriei software”-Mircea Cezar Preda, Editura Polirom
3. http://www.biosfarm.ro/~dragos/papers/C++-Book/cap6.html
4. http://aut.unitbv.ro/aut/cpp/old/Tratarea%20exceptiilor.pdf
5. http://www.cplusplus.com/doc/tutorial/exceptions/
6. http://www.timsoft.ro/aux/module/modul5-plus.html
7. http://www.tutorialspoint.com/java/java_exceptions.htm
8. http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html
9. http://www.codeproject.com/Articles/8049/User-Friendly-ASP-NET-Exception-Handling
10. http://software.ucv.ro/~cstoica/MPV/VPE_Lab2.pdf
11. http://blog.zeltera.eu/?p=2295