Seminar 8. Tratarea exceptiilor in Java

1 Partea Teoretica Timp de lucru estimat: 60 minute

1.1 Introducere in exceptii

In Java, exceptiile reprezinta evenimente neasteptate care apar in timpul rularii programului

(runtime) si care pot intrerupe fluxul normal al instructiunilor. Mecanismul de tratare a exceptiilor

permite: - detectarea erorilor - tratarea lor controlata (in locul opririi bruste a programului) -

separarea codului normal de codul de tratare a erorilor

1.2 Ierarhia exceptiilor in Java

Toate erorile si exceptiile sunt derivate din clasa de baza Throwable:

- Throwable - Error – erori grave ale JVM (de ex. OutOfMemoryError), nu se trateaza de obicei -
  `Exception - RuntimeException – exceptii necontrolate (unchecked) - NullPointerException -`

ArithmeticException - ArrayIndexOutOfBoundsException - NumberFormatException etc. - alte

exceptii verificate (checked) - IOException - SQLException - FileNotFoundException etc.

1.3 Tipuri de exceptii: checked vs unchecked

Exceptii verificate (checked): - sunt verificate la compilare - compilatorul obliga programatorul sa: -

le trateze cu try-catch SAU - sa le declare cu throws in semnatura metodei - exemple: IOException,

SQLException

Exceptii neverificate (unchecked): - sunt subclase ale RuntimeException - nu sunt verificate la

compilare - apar de obicei din erori de programare (bug-uri) - exemple: NullPointerException,

ArithmeticException, ArrayIndexOutOfBoundsException

1.4 Blocurile try-catch

Sintaxa generala:

```
try {
    // cod care poate arunca exceptii
} catch ( TipExceptie1 e ) {
```

```
        // tratament pentru TipExceptie1
    } catch ( TipExceptie2 e ) {
        // tratament pentru TipExceptie2
    }
```

- codul din try este monitorizat pentru aparitia exceptiilor
- daca apare o exceptie, se cauta primul bloc catch compatibil
- daca nu se gaseste niciun catch compatibil, exceptia se propaga mai departe (stack unwinding)

1.4.1 Exemplu simplu de try-catch

```
    public class Impartire {
        public static void main( String[] args ) {
int a = 10;
int b = 0;
    try {
int rezultat = a / b;
                System.out.println( "Rezultat: " + rezultat );
            } catch ( ArithmeticException e ) {
                System.out.println( "Eroare: impartire la zero!" );

            }
    System.out.println( "Programul continua dupa tratarea exceptiei." ); }
     }
```

1.4.2 Multiple blocuri catch

Putem trata diferite tipuri de exceptii separat:

```
    public class ExempluMultipleCatch {
        public static void main( String[] args ) {
            try {
String text = "123a";
int valoare = Integer.parseInt( text ); // poate arunca NumberForm
atException
    int[] v = {1, 2, 3}; int x = v[5]; // poate arunca ArrayIndexOutOfBoun
    dsException
    } catch ( NumberFormatException e ) { System.out.println( "Textul nu e
    ste un numar intreg valid." );
    } catch ( ArrayIndexOutOfBoundsException e ) { System.out.println( "In
    dex in afara limitelor
    vectorului." ); } catch ( Exception e ) { System.out.println( "A aparu
    t o alta exceptie: " +
    e.getMessage() ); } } }
```

Observatie: blocul catch (Exception e) trebuie pus intotdeauna la final (este cel mai general).

1.4.3 Multi-catch (un singur catch pentru mai multe tipuri)

```
public class ExempluMultiCatch {
    public static void main( String[] args ) {
        try {
String text = "10a";
int valoare = Integer.parseInt( text );
int[] v = {1, 2, 3};
int x = v[4];
        } catch ( NumberFormatException | ArrayIndexOutOfBoundsExcepti
on e ) {
            System.out.println( "Exceptie de date sau index: " + e );
        }
    }
}
```

1.5 Blocul finally

Blocul finally este executat intotdeauna, indiferent daca o exceptie a fost aruncata sau nu.

```
public class ExempluFinally {
    public static void main( String[] args ) {
        try {
int rezultat = 10 / 0;
            System.out.println( rezultat );
        } catch ( ArithmeticException e ) {
            System.out.println( "Impartire la zero." );
        } finally {
            System.out.println( "Acest mesaj se afiseaza intotdeauna."
 );
        }
    }
}
```

Utilizare tipica: inchiderea resurselor (fisiere, conexiuni la baza de date etc.)

1.6 Instructiunea throws si propagarea exceptiilor

O metoda poate declara ca arunca o anumita exceptie:

```
public void citesteFisier( String numeFisier ) throws IOException {
    // cod care poate arunca IOException
}
```

- throws se foloseste pentru a propaga exceptiile catre apelant - apelantul trebuie fie: - sa trateze

exceptia cu try-catch, fie - sa o redeclare cu throws

Exemplu:

```
   import java.io.BufferedReader;
   import java.io.FileReader;
   import java.io.IOException;
   public class CititorFisier {
   public static String citestePrimaLinie( String numeFisier ) throws IOE
   xception
    {
           BufferedReader br = new BufferedReader( new FileReader( numeFi
   sier ) )
;
String linie = br.readLine();
br.close();
return linie;
       }
   public static void main( String[] args ) {
           try {
String primaLinie = citestePrimaLinie( "test.txt" );
               System.out.println( primaLinie );
           } catch ( IOException e ) {
System.out.println( "Eroare la citirea fisierului: " + e.getMessag
e() );
           }
       }
   }
```

1.7 Try-with-resources

In Java 7+ putem folosi try-with-resources pentru a inchide automat resursele:

```
   import java.io.BufferedReader;
   import java.io.FileReader;
   import java.io.IOException;
   public class ExempluTryWithResources {
       public static void main( String[] args ) {
try ( BufferedReader br = new BufferedReader( new FileReader( "fisier.
   txt" ) ) ) {
String linie;
               while ( ( linie = br.readLine() ) != null ) {
                   System.out.println( linie );
               }
           } catch ( IOException e ) {
               System.out.println( "Eroare IO: " + e.getMessage() );
           }
```

```
        }
    }
```

- orice resursa declarata intre parantezele try (...) trebuie sa implementez
e interfata AutoCloseable
- resursele sunt inchise automat la sfarsitul blocului try, chiar daca apare o
exceptie

1.8 Exceptii personalizate (custom exceptions)

Putem defini propriile clase de exceptii, extinzand Exception sau RuntimeException:

```
    public class VarstaInvalidaException extends Exception {
        public VarstaInvalidaException( String mesaj ) {
```

super(mesaj);

```
        }
    }
```

Utilizare:

```
    public class Persoana {
    public static void verificaVarsta( int varsta ) throws VarstaInvalidaE
    xception
     {
            if ( varsta < 18 ) {
```

throw new VarstaInvalidaException("Varsta trebuie sa fie cel puti

n 18 ani.");

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

verificaVarsta(16);

```
                System.out.println( "Acces permis." );
            } catch ( VarstaInvalidaException e ) {
                System.out.println( "Exceptie personalizata: " + e.getMess
    age() );
            }
        }
    }
```

1.9 Bune practici in tratarea exceptiilor

- Nu folositi exceptii pentru controlul normal al fluxului programului. - Nu lasati blocuri catch goale:

catch (Exception e) { } - Afisati sau logati informatii utile pentru depanare. - Tratati specific

exceptiile, nu folositi mereu Exception. - Nu ascundeti cauza reala a exceptiei: - folositi

e.getMessage() sau e.printStackTrace() in timpul dezvoltarii. - Curatati corect resursele (de preferat

try-with-resources).

2 Partea Practica Timp de lucru estimat: 90 minute

2.1 Probleme Rezolvate

2.1.1 Problema 1: Tratarea ArithmeticException (impartire la zero)

Enunt: Scrieti un program care citeste doua numere intregi de la tastatura si afiseaza rezultatul

impartirii lor. Tratati cazul in care al doilea numar este zero folosind un bloc try-catch.

```
import java.util.Scanner;
public class ImpartireSigura {
    public static void main( String[] args ) {
        Scanner scanner = new Scanner( System.in );
System.out.print( "Introduceti numaratorul: " ); int a = scanner.nextI
nt();
System.out.print( "Introduceti numitorul: " ); int b = scanner.nextInt
();
try {
```

int rezultat = a / b;

```
            System.out.println( "Rezultatul impartirii este: " + rezul
tat );

        } catch ( ArithmeticException e ) {
            System.out.println( "Eroare: nu se poate imparti la zero."
 );
        }
System.out.println( "Program terminat." ); scanner.close(); } }
```

2.1.2 Problema 2: Tratarea NumberFormatException

Enunt: Scrieti un program care citeste de la tastatura un sir de caractere si incearca sa il

converteasca la un numar intreg folosind Integer.parseInt(). Tratati cazul in care sirul nu reprezinta

un numar valid.

```
import java.util.Scanner;
public class ConversieSigura {
    public static void main( String[] args ) {
        Scanner scanner = new Scanner( System.in );
System.out.print( "Introduceti un numar intreg: " ); String text = sca
nner.nextLine();
try {
```

int valoare = Integer.parseInt(text);

```
            System.out.println( "Ati introdus numarul: " + valoare );
        } catch ( NumberFormatException e ) {
System.out.println( "Eroare: sirul introdus nu este un numar intre
g valid." );
        }
scanner.close(); } }
```

2.1.3 Problema 3: Tratarea ArrayIndexOutOfBoundsException

Enunt: Scrieti un program care creeaza un vector de 5 elemente si cere utilizatorului un index.

Afisati elementul de la indexul respectiv, tratand cazul in care indexul este in afara limitelor vectorului.

```
import java.util.Scanner;
public class AccesVector {
    public static void main( String[] args ) {
int[] v = { 10, 20, 30, 40, 50 };
        Scanner scanner = new Scanner( System.in );
System.out.print( "Introduceti un index (0-4): " ); int index = scanne
r.nextInt();
try {
int valoare = v[index];
System.out.println( "Elementul de la indexul " + index + " este: "
+ valoare );
        } catch ( ArrayIndexOutOfBoundsException e ) {
System.out.println( "Eroare: index in afara limitelor vectorului."
);
        }
scanner.close(); } }
```

2.1.4 Problema 4: Propagarea exceptiilor cu throws

Enunt: Scrieti o metoda care calculeaza radacina patrata a unui numar. Daca numarul este negativ,

aruncati o exceptie IllegalArgumentException. Apelati metoda din main si tratati exceptia acolo.

```
public class RadacinaPatrata {
public static double calculeazaRadacina( double x ) {
        if ( x < 0 ) {
throw new IllegalArgumentException( "Numarul trebuie sa fie nenega
tiv." );
        }
```

```
return Math.sqrt( x );
        }
    public static void main( String[] args ) {
            try {
double rezultat = calculeazaRadacina( -4 );
                System.out.println( "Radacina este: " + rezultat );
            } catch ( IllegalArgumentException e ) {
                System.out.println( "Exceptie: " + e.getMessage() );
            }
        }
    }
```

2.1.5 Problema 5: Definirea si utilizarea unei exceptii personalizate

Enunt: Definiti o exceptie personalizata SalariuInvalidException care sa fie aruncata atunci cand se

incearca setarea unui salariu negativ pentru un angajat.

```
    // Exceptie personalizata
    class SalariuInvalidException extends Exception {
        public SalariuInvalidException( String mesaj ) {
super( mesaj );
        }
    }
    // Clasa Angajat
    class Angajat {
private String nume;
private double salariu;
    public Angajat( String nume ) {
this.nume = nume;
        }
    public void setSalariu( double salariu ) throws SalariuInvalidExceptio
    n {
            if ( salariu < 0 ) {
throw new SalariuInvalidException( "Salariul nu poate fi negativ."
);
            }
this.salariu = salariu;
        }
    public double getSalariu() {
return salariu;
        }
    }
```

```
public class TestAngajat {
    public static void main( String[] args ) {
        Angajat a = new Angajat( "Ion Popescu" );
        try {
a.setSalariu( -1000 );
            System.out.println( "Salariu: " + a.getSalariu() );
        } catch ( SalariuInvalidException e ) {
System.out.println( "Exceptie la setarea salariului: " + e.getMess

age() );
        }
    }
}
```

2.2 Probleme Propuse

1. Scrieti un program care citeste doua numere intregi de la tastatura si efectueaza operatiile de

adunare, scadere, inmultire si impartire. Tratati separat exceptia de impartire la zero si orice alta

exceptie neasteptata. - Folositi try-catch si un bloc catch (Exception e) general. - Afisati mesaje

clare pentru utilizator.

2. Realizati un program care citeste dintr-un fisier numere intregi (cate unul pe linie) si calculeaza

suma lor. Tratati atat IOException, cat si NumberFormatException (in cazul in care o linie nu

contine un numar valid). - Folositi BufferedReader. - Pentru fiecare linie invalida, afisati un

avertisment si continuati citirea.

3. Scrieti un program care cere utilizatorului sa introduca un numar intreg intre 1 si 10. Daca

numarul nu este in acest interval, aruncati o exceptie personalizata

NumarInAfaraIntervaluluiException si tratati-o in main. - Definiti exceptia extinzand Exception. - In

mesajul exceptiei includeti valoarea introdusa.

4. Creati un program care demonstreaza utilizarea try-with-resources pentru citirea dintr-un fisier

text. Tratati exceptiile corespunzator. - Declarati BufferedReader in parantezele try. - Tratati

`IOException intr-un bloc catch.`

5. Scrieti un program care simuleaza o operatie bancara de retragere din cont. Daca suma ceruta

depaseste soldul, aruncati o exceptie personalizata FonduriInsuficienteException. - Clasa

ContBancar trebuie sa contina metodele depozit si retrage. - Tratati exceptia in main si afisati soldul

curent.

6. Realizati un program care foloseste mai multe blocuri catch pentru a trata: - NullPointerException

- ArrayIndexOutOfBoundsException - orice alta exceptie generala Creati deliberat situatii care sa

produca aceste exceptii si observati comportamentul programului.

7. Scrieti un program care implementeaza o metoda citesteIntSigur() ce citeste un numar intreg de

la tastatura. Metoda trebuie: - sa ceara utilizatorului sa reintroduca valoarea pana cand primeste un

numar valid - sa trateze NumberFormatException folosind try-catch - sa returneze valoarea corecta

catre apelant.

8. Creati un program care arunca si trateaza o exceptie imbricata (exception chaining): - in interiorul

unei metode, prindeti o exceptie (de ex. NumberFormatException) - aruncati o noua exceptie (de

ex. Exception) care contine cauza initiala (initCause sau constructor cu cauza) - tratati exceptia din

main si afisati atat mesajul principal, cat si cauza.

9. Scrieti un program care demonstreaza diferenta dintre tratarea exceptiilor checked si unchecked:

- creati o metoda care arunca IOException (checked) si o alta care arunca RuntimeException

(unchecked) - observati ce impune compilatorul pentru fiecare metoda (folosirea throws sau nu).

10. Realizati un program care defineste o ierarhie de exceptii personalizate, de exemplu: -

AplicatieException (baza) - DateInvalideException - OperatieNepermisaException Scrieti o metoda

care poate arunca oricare dintre aceste exceptii si tratati-le intr-un mod diferentiat in main, folosind

atat tipurile specifice, cat si tipul de baza.

2.3 Rezolvari pentru problemele propuse

2.3.1 Problema 1

Enunt (rezumat):

Se citesc doua numere intregi de la tastatura si se efectueaza operatiile de adunare, scadere, inmultire si impartire.

Se trateaza separat impartirea la zero si orice alta exceptie neasteptata.

Rezolvare (varianta profesionala):

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class CalculatorAritmetic {
    public static void main( String[] args ) {
        Scanner scanner = new Scanner( System.in );
        try {
            System.out.print( "Introduceti primul numar intreg: " );
int a = scanner.nextInt();

            System.out.print( "Introduceti al doilea numar intreg: " )
;
int b = scanner.nextInt();

            System.out.println( "Adunare: " + ( a + b ) );
            System.out.println( "Scadere: " + ( a - b ) );
            System.out.println( "Inmultire: " + ( a * b ) );

            try {
int impartire = a / b;
                System.out.println( "Impartire: " + impartire );
            } catch ( ArithmeticException e ) {
                System.out.println( "Eroare: impartire la zero nu este
 permisa." );
            }

        } catch ( InputMismatchException e ) {
            System.out.println( "Eroare: trebuia sa introduceti numere
 intregi." );
        } catch ( Exception e ) {
            System.out.println( "A aparut o eroare neasteptata: " + e.
getMessage() );
        } finally {
scanner.close();
            System.out.println( "Program terminat." );
        }
    }
}
```

Acest program trateaza separat:

- erorile de intrare (InputMismatchException),
- impartirea la zero (ArithmeticException),

- orice alta eroare general (Exception).

2.3.2 Problema 2

Enunt (rezumat):

Se citesc dintr-un fisier numere intregi (cate unul pe linie) si se calculeaza suma lor.

Se trateaza IOException si NumberFormatException, iar liniile invalide nu opresc programul.

Rezolvare:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class SumaDinFisier {
    public static void main( String[] args ) {
String numeFisier = "numere.txt";
long suma = 0;
int liniaCurenta = 0;

        try ( BufferedReader br = new BufferedReader( new FileReader(
numeFisier ) ) ) {
String linie;
            while ( ( linie = br.readLine() ) != null ) {
liniaCurenta++;
linie = linie.trim();
                if ( linie.isEmpty() ) {
continue;
                }
                try {
int valoare = Integer.parseInt( linie );
suma += valoare;
                } catch ( NumberFormatException e ) {
System.out.println(
"Avertisment: linia " + liniaCurenta +
" nu contine un numar intreg valid: \"" + linie + "\""
);
                }
            }
            System.out.println( "Suma numerelor valide din fisier este
: " + suma );
        } catch ( IOException e ) {
```

```
            System.out.println( "Eroare la citirea fisierului: " + e.g
etMessage() );
        }
    }
}
```

Programul foloseste try-with-resources pentru a inchide automat fisierul si trateaza separat erorile de parsare la nivel de linie.

2.3.3 Problema 3

Enunt (rezumat):

Se cere utilizatorului un numar intreg intre 1 si 10.

Daca numarul nu este in interval, se arunca o exceptie personalizata NumarInAfaraIntervaluluiException, care este tratata in main.

Rezolvare:

```
import java.util.Scanner;

// Exceptie personalizata
class NumarInAfaraIntervaluluiException extends Exception {
    public NumarInAfaraIntervaluluiException( String mesaj ) {
super( mesaj );
    }
}

public class CitireNumarInterval {

    public static int citesteNumarIntre1si10() throws NumarInAfaraInte
rvaluluiException {
        Scanner scanner = new Scanner( System.in );
        System.out.print( "Introduceti un numar intre 1 si 10: " );
int n = scanner.nextInt();
        if ( n < 1 || n > 10 ) {
throw new NumarInAfaraIntervaluluiException(
"Valoare in afara intervalului [1, 10]: " + n
);
        }
return n;
    }

    public static void main( String[] args ) {
        try {
int n = citesteNumarIntre1si10();
            System.out.println( "Ati introdus: " + n );
```

```
            } catch ( NumarInAfaraIntervaluluiException e ) {
                System.out.println( "Exceptie: " + e.getMessage() );
            }
        }
    }
```

Aceasta solutie ilustreaza utilizarea unei exceptii checked personalizate pentru validare de intrare.

2.3.4 Problema 4

Enunt (rezumat):

Se demonstreaza utilizarea try-with-resources pentru citirea dintr-un fisier text,

tratand corespunzator IOException.

Rezolvare:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class CitireFisierText {
    public static void main( String[] args ) {
String numeFisier = "date.txt";

        try ( BufferedReader br = new BufferedReader( new FileReader(
numeFisier ) ) ) {
String linie;
            while ( ( linie = br.readLine() ) != null ) {
                System.out.println( linie );
            }
        } catch ( IOException e ) {
            System.out.println( "Eroare la lucrul cu fisierul: " + e.g
etMessage() );
        }
    }
}
```

Resursa BufferedReader este inchisa automat, chiar daca apare o exceptie in interiorul blocului try.

2.3.5 Problema 5

Enunt (rezumat):

Se implementeaza o clasa ContBancar cu metodele depozit si retrage.

Daca suma retrasa depaseste soldul, se arunca o exceptie personalizata FonduriInsuficienteException, tratata in main.

Rezolvare:

```
   // Exceptie personalizata
   class FonduriInsuficienteException extends Exception {
       public FonduriInsuficienteException( String mesaj ) {
super( mesaj );
       }
   }

   class ContBancar {
private double sold;

       public ContBancar( double soldInitial ) {
this.sold = soldInitial;
       }

       public void depozit( double suma ) {
           if ( suma > 0 ) {
sold += suma;
               System.out.println( "Depunere: " + suma + " | Sold: " + so
   ld );
           }
       }

       public void retrage( double suma ) throws FonduriInsuficienteExcep
   tion {
           if ( suma <= 0 ) {
               throw new IllegalArgumentException( "Suma de retras trebui
   e sa fie pozitiva." );
           }
           if ( suma > sold ) {
throw new FonduriInsuficienteException(
"Fonduri insuficiente. S-a cerut " + suma + ", dar soldul este " + sold
);
           }
sold -= suma;
           System.out.println( "Retragere: " + suma + " | Sold: " + sold
   );
       }

       public double getSold() {
return sold;
       }
   }
```

```
public class TestContBancar {
    public static void main( String[] args ) {
        ContBancar cont = new ContBancar( 500 );
        try {
cont.depozit( 200 );
cont.retrage( 100 );
cont.retrage( 800 ); // va genera exceptie
        } catch ( FonduriInsuficienteException e ) {
            System.out.println( "Operatie esuata: " + e.getMessage() )
;
        }
        System.out.println( "Sold final: " + cont.getSold() );
    }
}
```

Programul combina exceptii standard (IllegalArgumentException) cu o exceptie custom pentru logica de business.

2.3.6 Problema 6

Enunt (rezumat):

Se cere un program care foloseste mai multe blocuri catch pentru a trata:

NullPointerException, ArrayIndexOutOfBoundsException si orice alta exceptie generala.

Rezolvare:

```
public class MultipleExceptiiDemo {
    public static void main( String[] args ) {
        try {
String s = null;
            System.out.println( s.length() ); // NullPointerException
int[] v = { 1, 2, 3 };
            System.out.println( v[5] ); // ArrayIndexOutOfBoundsExcept
ion
        } catch ( NullPointerException e ) {
            System.out.println( "Ati incercat sa accesati o referinta
null." );
        } catch ( ArrayIndexOutOfBoundsException e ) {
            System.out.println( "Index in afara limitelor tabloului."
);
        } catch ( Exception e ) {
            System.out.println( "Alta exceptie: " + e.getClass().getNa
me() + " - " + e.getMessage() );
```

```
            }
        }
    }
```

Blocurile catch sunt ordonate de la cel mai specific la cel mai general (Exception).

2.3.7 Problema 7

Enunt (rezumat):
Se implementeaza o metoda citesteIntSigur() care citeste un numar intreg de la tastatura, cerand utilizatorului sa reintroduca valoarea pana cand primeste un numar valid.
Se trateaza NumberFormatException.

Rezolvare:

```
import java.util.Scanner;

public class CitireIntSigur {

    public static int citesteIntSigur( Scanner scanner ) {
        while ( true ) {
            System.out.print( "Introduceti un numar intreg: " );
String linie = scanner.nextLine();
            try {
int valoare = Integer.parseInt( linie.trim() );
return valoare;
            } catch ( NumberFormatException e ) {
                System.out.println( "Valoare invalida. Va rugam introd
uceti un numar intreg." );
            }
        }
    }

    public static void main( String[] args ) {
        Scanner scanner = new Scanner( System.in );
int x = citesteIntSigur( scanner );
        System.out.println( "Ati introdus: " + x );
scanner.close();
    }
}
```

Aceasta metoda incapsuleaza logica de validare si nu permite intoarcerea pana la obtinerea unui intreg valid.

2.3.8 Problema 8

Enunt (rezumat):

Se cere un program care arunca si trateaza o exceptie imbricata (exception chaining): se prinde o exceptie (de ex. NumberFormatException) si se arunca o noua exceptie care contine cauza initiala.

Rezolvare:

```
public class ExceptionChainingDemo {

    public static void proceseazaText( String text ) throws Exception
{
        try {
int valoare = Integer.parseInt( text );
            System.out.println( "Valoarea este: " + valoare );
        } catch ( NumberFormatException e ) {
            Exception ex = new Exception( "Eroare la conversia textulu
i in intreg." );
ex.initCause( e );
throw ex;
        }
    }

    public static void main( String[] args ) {
        try {
proceseazaText( "12a3" );
        } catch ( Exception e ) {
            System.out.println( "Exceptie prinsa in main: " + e.getMes
sage() );
Throwable cauza = e.getCause();
            if ( cauza != null ) {
                System.out.println( "Cauza initiala: " + cauza.getClas
s().getName() + " - " + cauza.getMessage() );
            }
        }
    }
}
```

Se pune in evidenta lantul de exceptii prin metoda initCause si prin afisarea cause-ului in main.

2.3.9 Problema 9

Enunt (rezumat):

Se cere un program care demonstreaza diferenta dintre exceptiile checked si unchecked

folosind o metoda care arunca IOException (checked) si o alta care arunca RuntimeException (unchecked).

Rezolvare:

```
import java.io.IOException;

public class CheckedUncheckedDemo {

    // Exceptie checked
    public static void metodaChecked() throws IOException {
        throw new IOException( "Eroare IO simulat..." );
    }

    // Exceptie unchecked
    public static void metodaUnchecked() {
        throw new RuntimeException( "Eroare unchecked simulata..." );
    }

    public static void main( String[] args ) {
        try {
metodaChecked();
        } catch ( IOException e ) {
            System.out.println( "Am tratat IOException: " + e.getMessa
ge() );
        }

        // Pentru metodaUnchecked compilatorul nu ne obliga sa folosim
 try-catch
        try {
metodaUnchecked();
        } catch ( RuntimeException e ) {
            System.out.println( "Am tratat RuntimeException: " + e.get
Message() );
        }
    }
}
```

Se observa ca metodaChecked() trebuie declarata cu throws si tratata,
in timp ce metodaUnchecked() nu obliga tratarea la nivelul compilarii.

2.3.10 Problema 10

Enunt (rezumat):
Se defineste o ierarhie de exceptii personalizate:
AplicatieException (baza), DateInvalideException si OperatieNepermisaException.
O metoda poate arunca oricare dintre acestea, iar in main se trateaza diferentiat.

Rezolvare:

```
   // Exceptia de baza
   class AplicatieException extends Exception {
       public AplicatieException( String mesaj ) {
super( mesaj );
       }
   }

   // Exceptie pentru date invalide
   class DateInvalideException extends AplicatieException {
       public DateInvalideException( String mesaj ) {
super( mesaj );
       }
   }

   // Exceptie pentru operatii nepermise
   class OperatieNepermisaException extends AplicatieException {
       public OperatieNepermisaException( String mesaj ) {
super( mesaj );
       }
   }

   public class IerarhieExceptiiDemo {

       public static void proceseaza( String operatie, int valoare )
               throws AplicatieException {
           if ( valoare < 0 ) {
               throw new DateInvalideException( "Valoare negativa: " + va
   loare );
           }
           if ( "sterge".equals( operatie ) && valoare == 0 ) {
throw new OperatieNepermisaException(
"Nu se poate sterge elementul cu id 0." );
           }
           System.out.println( "Operatie " + operatie + " realizata cu va
   loarea " + valoare );
       }

       public static void main( String[] args ) {
           try {
proceseaza( "sterge", 0 );
           } catch ( DateInvalideException e ) {
               System.out.println( "Date invalide: " + e.getMessage() );
           } catch ( OperatieNepermisaException e ) {
```

```
            System.out.println( "Operatie nepermisa: " + e.getMessage(
) );
        } catch ( AplicatieException e ) {
            System.out.println( "Alta eroare de aplicatie: " + e.getMe
ssage() );
        }
    }
}
```

Aceasta solutie evidentiaza tratarea diferentiata a exceptiilor derivate din acelasi tip de baza.