Aplicatii JAVA

7

JAVA
Excepții în Java

Adrian Runceanu
www.runceanu.ro/adrian

Curs 7

Excepții în Java

Excepții în Java

1. Ce sunt excepțiile?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

Ce sunt excepțiile?

Termenul **exceptie** este o prescurtare pentru "eveniment exceptional" si poate fi definit astfel:

*O **exceptie** este un eveniment ce se produce în timpul executiei unui program si care provoaca întreruperea cursului normal al executiei.*

Excepții

Exceptiile pot aparea din diverse cauze si pot avea nivele diferite de gravitate: de la erori fatale cauzate
de echipamentul hardware până 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 excepții

Excepții

```
public class Exceptii {
    public static void main(String argsst) {
    int v[] = new int[10];
    v[10] = 111;            // exceptie, vectorul are elementele v[0]...v[9]
    System.out.println("Aici nu se mai ajunge...");
    }
}
```

La rularea programului va fi generata o exceptie si se va afisa mesajul:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException :10
at Exceptii.main (Exceptii.java:4)
```

Excepții

- 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 excepție) 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.

Excepții

- 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 (în cazul exemplului de mai sus mesajul "Aici nu se mai ajunge..." nu va fi tiparit).

Excepții în Java

1. Ce sunt excepțiile ?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

Avantajele excepțiilor

Prin modalitatea sa de tratare a excepțiilor, Java are urmatoarele avantaje față de mecanismul tradițional de tratare a erorilor:

1. Separarea codului pentru tratarea unei erori de codul în care ea poate să apară
2. Propagarea unei erori până la un analizor de exceptii corespunzător
3. Gruparea erorilor după tipul lor

Avantajele excepțiilor

1. Separarea codului pentru tratarea unei erori de codul în care ea poate să apară

In programarea traditionala tratarea erorilor se combina cu codul ce poate produce aparitia lor conducând la așa-numitul "cod spaghetti".

Sa consideram urmatorul exemplu: o functie care încarca un fisier în memorie:

```
citesteFisier {
        deschide fisierul;
        determina dimensiunea fisierului;
        aloca memorie;
        citeste fisierul in memorie;
        inchide fisierul;
}
```

Avantajele excepțiilor

Problemele care pot aparea la aceasta functie, aparent simpla sunt de genul: "Ce se întâmpla daca: ... ?"

- fisierul nu poate fi deschis
- nu se poate determina dimensiunea fisierului
- nu poate fi alocata suficienta memorie
- nu se poate face citirea din fisier
- fisierul nu poate fi închis

Avantajele excepțiilor

Un cod traditional care sa trateze aceste erori ar arata astfel:

```
int citesteFisier {
int codEroare = 0;
deschide fisier;
if (fisierul s-a deschis) {
        determina dimensiunea fisierului;
        if (s-a determinat dimensiunea) {
                aloca memorie;
                if (s-a alocat memorie) {
                        citeste fisierul in memorie;
                        if (nu se poate citi din fisier) codEroare = -1;
                        else codEroare = -2;
                }
                } else codEroare = -3;
        }
```

Avantajele excepțiilor

```
            inchide fisierul;
            if (fisierul nu s-a inchis && codEroare == 0) {
                        codEroare = -4;
            } else codEroare = -4;
} else codEroare = -5;
return codEroare;
} // cod "spaghetti"
```

Acest stil de progamare este extrem de susceptibil la erori si îngreunează extrem de mult înțelegerea sa.

Avantajele excepțiilor

In Java, folosind mecansimul exceptiilor, codul ar arata astfel:

```
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 din fisier) { trateaza eroarea; }
        catch (nu se poate inchide fisierul) { trateaza eroarea; }
}
```

Avantajele excepțiilor

2. Propagarea unei erori până la un analizor de excepții corespunzător

Sa presupunem ca apelul la metoda citesteFisier este consecinta unor apeluri imbricate de metode:

```
int metoda1 {
        apel metoda2;
        . . .
}
```

```
int metoda2 {
        apel metoda3;
        . . .
}
int metoda3 {
        apel citesteFisier;
        . . .
}
```

Avantajele excepțiilor

Sa presupunem de asemenea ca dorim sa facem tratarea erorilor doar în metoda1.
Traditional, acest lucru ar trebui facut prin propagarea erorii întoarse de metoda citesteFisier pâna la metoda1.

```
int metoda1 {
int codEroare = apel metoda2;
if (codEroare != 0)
        proceseazaEroare;
. . .
}
```

```
int metoda2 {
int codEroare = apel metoda3;
if (codEroare != 0)
        return codEroare;
. . .
}
int metoda3 {
int codEroare = apel citesteFisier;
if (codEroare != 0)
        return codEroare;
. . .
}
```

Avantajele excepțiilor

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 își 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;
. . .
}
```

Avantajele excepțiilor

3. Gruparea erorilor după tipul lor

- In Java exista clase corespunzatoare tuturor exceptiilor care pot aparea la executia unui program.
- Acestea sunt grupate în functie de similaritatile lor într-o ierarhie de clase.
- De exemplu, clasa IOException se ocupa cu exceptiile ce pot aparea la operatii de intrare/iesire si diferentiaza la rândul ei alte tipuri de exceptii, cum ar fi FileNotFoundException, EOFException, etc.

Avantajele excepțiilor

- La rândul ei clasa IOException se încadreaza într-o categorie mai larga de exceptii si anume 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:

Avantajele excepțiilor

```
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!
}
```

Excepții în Java

1. Ce sunt excepțiile ?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

3. "Prinderea" și tratarea excepțiilor

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
}
```

3. "Prinderea" și tratarea excepțiilor

Sa consideram urmatorul exemplu : citirea unui fisier si afisarea lui pe ecran.
Fara a folosi tratarea exceptiilor codul programului ar arata astfel:

```
// ERONAT!
import java.io.*;
public class CitireFisier {
public static void citesteFisier() {
FileInputStream sursa = null;  // sursa 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.

3. "Prinderea" și tratarea excepțiilor

Folosind mecanismul exceptiilor metoda citesteFisier își poate trata singura erorile pe care le poate provoca:

```
// CORECT
import java.io.*;
public class CitireFisier {
public static void citesteFisier() {
        FileInputStream sursa = null; // sursa 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);
                }
```

3. "Prinderea" și tratarea excepțiilor

```
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();
        }
```

3. "Prinderea" și tratarea excepțiilor

```
        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();
        }
}
```

3. "Prinderea" și tratarea excepțiilor

- 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.

- Inchiderea fisierului se face în blocul "finally", deoarece acesta este sigur ca se va executa.

3. "Prinderea" și tratarea excepțiilor

Fara a folosi blocul "finally" închiderea fisierului ar fi trebuit facuta în fiecare situatie în care fisierul ar fi fost deschis, ceea ce ar fi dus la scrierea de cod redundant:

```
try {
. . .
sursa.close();
}
. . .
catch (IOException e) {
. . .
sursa.close(); //cod redundant
}
```

Nota:
Obligatoriu un bloc de instructiuni "try" trebuie sa fie urmat de unul sau mai multe blocuri "catch", în functie de exceptiile provocate de acele instructiuni sau (optional) de un bloc "finally"

Excepții în Java

1. Ce sunt excepțiile ?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

4. "Aruncarea" excepțiilor

- In cazul în care o metoda nu își asuma responsabilitatea tratarii uneia sau mai multor exceptii pe care le pot provoca anumite instructiuni din codul sau atunci ea poate sa "arunce" aceste exceptii catre metodele care o apeleaza, urmând ca acestea sa implementeze tratarea lor sau, la rândul lor, sa "arunce" mai departe exceptiile respective.
- Acest lucru se realizeaza prin specificarea în declaratia metodei a clauzei **throws**:

```
metoda throws TipExceptie1, TipExceptie2, ... {
. . .
}
```

4. "Aruncarea" excepțiilor

O metoda care nu trateaza o anumita exceptie trebuie obligatoriu sa o "arunce".
In exemplul de mai sus daca nu facem tratarea exceptiilor în cadrul metodei citesteFisier atunci metoda apelanta (main) va trebui sa faca acest lucru:

```
import java.io.*;
public class CitireFisier {
public static void citesteFisier() throws FileNotFoundException, IOException
{
        FileInputStream sursa = null; // sursa 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();
}
```

4. "Aruncarea" excepțiilor

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

4. "Aruncarea" excepțiilor

Se observa ca, în acest caz, nu mai putem diferentia exceptiile provocate de citirea din fisier si de închiderea fisierului ambele fiind de tipul IOException.

Aruncarea unei exceptii se poate face si implicit prin instructiunea throw ce are formatul: **throw obiect_de_tip_Exceptie**.

Exemple:

```
throw new IOException();
if (index >= vector.length)
        throw new ArrayIndexOutOfBoundsException();
        catch(Exception e) {
                System.out.println("A aparut o exceptie);
                throw e;
        }
```

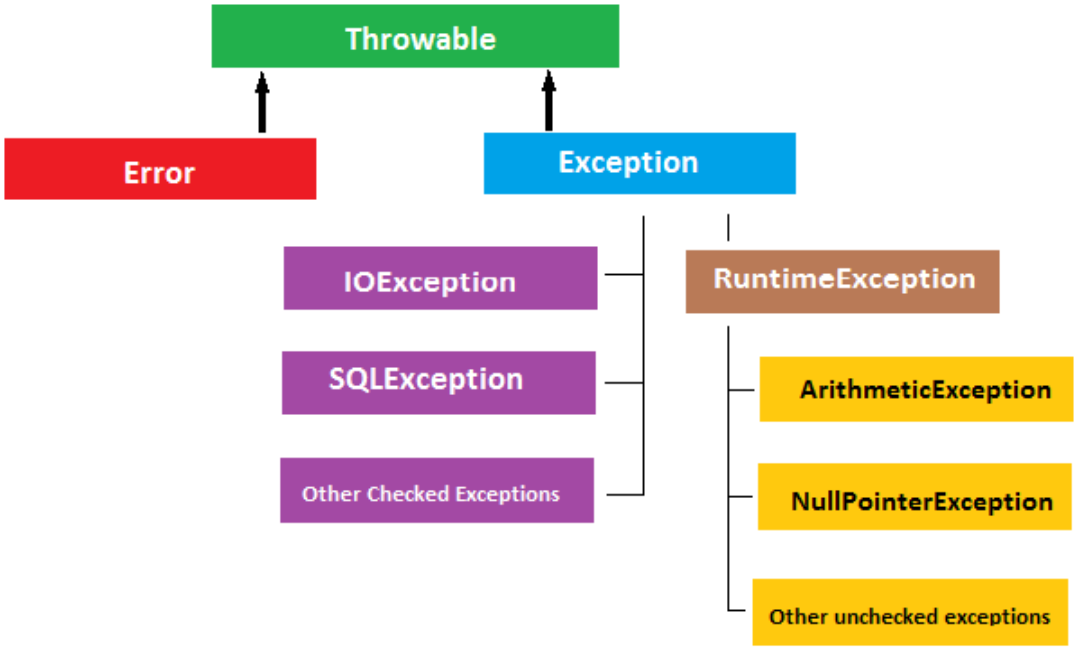
Aceasta instructune este folosita mai ales la aruncarea exceptiilor proprii care, evident, nu sunt detectate de catre mediul de executie.

Excepții în Java

1. Ce sunt excepțiile ?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

5. Ierarhia claselor ce descriu excepții

Radacina claselor ce descriu exceptii este clasa **Thowable** iar cele mai importante subclase ale sale sunt Error, Exception si RuntimeException, care sunt la rândul lor superclase pentru o serie întreaga de tipuri de exceptii.



5. Ierarhia claselor ce descriu excepții

1. Clasa Error

- Erorile (obiecte de tip Error) sunt cazuri speciale de exceptii generate de functionarea anormala a echipamentului hard pe care ruleaza un program Java si sunt invizibile programatorilor.

- Un program Java nu trebuie sa trateze aparitia acestor erori si este improbabil ca o metoda Java sa provoace asemenea erori.

5. Ierarhia claselor ce descriu excepții

2. Clasa Exception

- Obiectele de acest tip sunt exceptiile standard care trebuie tratate de catre programele Java.
- In Java, tratarea exceptiilor nu este o optiune ci o constrângere.
- Exceptiile care pot "scapa" netratate sunt încadrate în subclasa RuntimeException si se numesc *exceptii la executie*.

5. Ierarhia claselor ce descriu excepții

In general metodele care pot fi apelate pentru un obiect exceptie sunt definite în clasa **Throwable** si sunt publice, astfel încât pot fi apelate pentru orice tip de exceptie.

Cele mai uzuale sunt:
String getMessage( ) - tipareste detaliul unei exceptii
void printStackTrace( ) - tipareste informatii despre localizarea exceptiei
String toString( ) - metoda din clasa Object, care returneaza reprezentarea ca sir de caractere a exceptiei

Excepții în Java

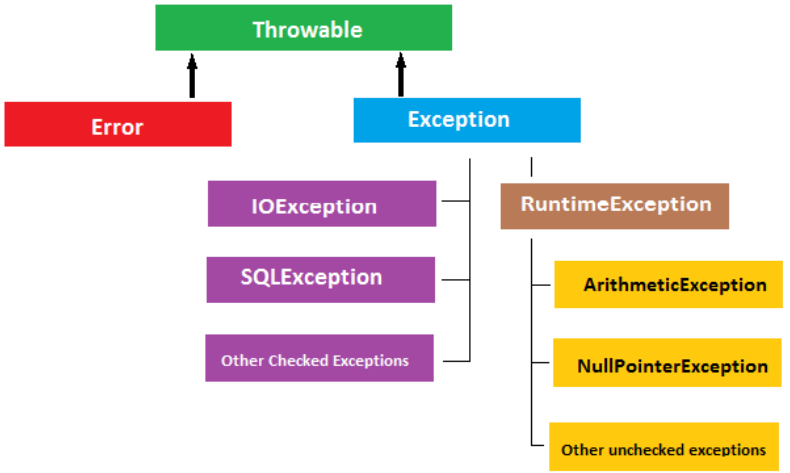
1. Ce sunt excepțiile ?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

6. Excepții la execuție

In general tratarea exceptiilor este obligatorie în Java.

De la acest principu se sustrag însa asa numitele *exceptii la executie* sau, cu alte cuvinte, *exceptiile care pot proveni strict din vina programatorului si nu generate de o cauza externa*.

6. Excepții la execuție



Aceste exceptii au o superclasa comuna si anume RuntimeException si în aceasta categorie sunt incluse:

1. operatii aritmetice (împartire la zero)
2. accesarea membrilor unui obiect ce are valoarea null
3. operatii cu elementele unui vector (accesare unui index din afara domeniului, etc)

6. Excepții la execuție

Aceste exceptii pot aparea oriunde în program si pot fi extrem de numeroare iar încercarea de "prindere" a lor ar fi extrem de anevoioasa.
Din acest motiv compilatorul permite ca aceste exceptii sa ramâna netratate, tratarea lor nefiind însa ilegala.

```
int v[] = new int[10];
try {
        v[10] = 111;
} catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Atentie la indecsi!");
        e.printStackTrace();
} // legal
```

Excepții în Java

1. Ce sunt excepțiile ?
2. Avantajele excepțiilor
3. "Prinderea" și tratarea excepțiilor
4. "Aruncarea" excepțiilor
5. Ierarhia claselor ce descriu excepții
6. Excepții la execuție
7. Crearea propriilor excepții

7. Crearea propriilor excepții

Adeseori poate aparea necesitatea crearii unor exceptii proprii pentru a pune în evidenta cazuri speciale de erori provocate de clasele unei librarii, cazuri care nu au fost prevazute în ierarhia exceptiilor standard Java.

O exceptie proprie trebuie sa se încadreze în ierarhia exceptiilor Java, cu alte cuvinte clasa care o implementeaza trebuie sa fie subclasa a unei clase deja existente în aceasta ierarhie, preferabil una apropiata ca semnificatie sau superclasa **Exception**.

7. Crearea propriilor excepții

```
class MyException extends Exception {
        public MyException() {}
        public MyException(String msg) {
        super(msg);
        // apeleaza constructorul superclasei Exception
        }
}
```

7. Crearea propriilor excepții

Un exemplu de folosire a exceptiei nou create:

```
public class TestMyException {
public static void f() throws MyException {
        System.out.println("Exceptie in f()");
        throw new MyException();
}
public static void g() throws MyException {
        System.out.println("Exceptie in g()");
        throw new MyException("aruncata din g()");
}
```

7. Crearea propriilor excepții

```
public static void main(String[] args) {
        try {
                f();
        } catch(MyException e) {e.printStackTrace();}
        try {
                g();
        } catch(MyException e) {e.printStackTrace();}
        }
}
```

7. Crearea propriilor excepții

- Fraza cheie este extends Exception care specifica faptul ca noua clasa MyException este subclasa a clasei Exception si deci implementeaza obiecte ce reprezinta exceptii.

- In general codul adaugat claselor pentru exceptii proprii este nesemnificativ: unul sau doi constructori care afiseaza un mesaj de eroare la iesirea standard.

7. Crearea propriilor excepții

Rularea programului de mai sus va produce urmatorul rezultat:

```
Exceptie in f()
MyException()
at TestMyException.f(TestMyException.java:12)
at TestMyException.main(TestMyException.java:20)
Exceptie in g()
MyException(): aruncata din g
at TestMyException.g(TestMyException.java:16)
at TestMyException.main(TestMyException.java:23)
```

7. Crearea propriilor excepții

- *Procesul de creare a unei noi exceptii poate fi dus mai departe prin adaugarea unor noi metode clasei ce descrie acea exceptie, însa aceasta dezvoltare nu îsi are rostul în majoritatea cazurilor.*
- In general, exceptiile proprii sunt descrise de clase foarte simple chiar fara nici un cod în ele, cum ar fi:

```
class SimpleException extends Exception { }
```

- Aceasta clasa se bazeaza pe constructorul implicit creat de compilator însa nu are constructorul SimpleException(String), care în practica nici nu este prea des folosit.

Referinte

- Curs practic de Java, Cristian Frasinaru – capitolul Exceptii
- https://docs.oracle.com/javase/tutorial/essential/exceptions/

Întrebări?