

Excepții

- Ce sunt excepțiile
- ”Prinderea” și tratarea excepțiilor
- ”Aruncarea” excepțiilor
- Avantajele tratării excepțiilor
- Ierarhia claselor ce descriu excepții
- Excepții la execuție
- Crearea propriilor excepții

Ce sunt excepțiile ?

Excepție = ”eveniment excepțional”

```
public class Exemplu {  
    public static void main(String args[]) {  
        int v[] = new int[10];  
        v[10] = 0; //Excepție !  
        System.out.println("Aici nu se mai ajunge...");  
    }  
}
```

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

- ”throw an exception”
- ”exception handler”
- ”catch the exception”

Tratarea erorilor nu mai este o opțiune
ci o **constrângere**

”Prinderea” și tratarea excepțiilor

try - catch - finally

```
try {  
    // Instructiuni care pot genera exceptii  
}  
catch (TipExceptie1 variabila) {  
    // Tratarea exceptiilor de tipul 1  
}  
catch (TipExceptie2 variabila) {  
    // Tratarea exceptiilor de tipul 2  
}  
.  
.  
.  
finally {  
    // Cod care se executa indiferent  
    // daca apar sau nu exceptii  
}
```

Citirea unui fișier

```
public static void citesteFisier(String fis) {
    FileReader f = null;
    // Deschidem fisierul
    f = new FileReader(fis);

    // Citim si afisam fisierul caracter cu caracter
    int c;
    while ( (c=f.read()) != -1)
        System.out.print((char)c);

    // Inchidem fisierul
    f.close();
}
```

Pot provoca excepții:

- Constructorul lui `FileReader`
- `read`
- `close`

```

public static void citesteFisier(String fis) {
    FileReader f = null;
    try {
        // Deschidem fisierul
        f = new FileReader(fis);
        // Citim si afisam fisierul caracter cu caracter
        int c;
        while ( (c=f.read()) != -1)
            System.out.print((char)c);

    } catch (FileNotFoundException e) {
        //Tratam un tip de exceptie
        System.err.println("Fisierul nu a fost gasit");

    } catch (IOException e) {
        //Tratam alt tip de exceptie
        System.out.println("Eroare la citire");
        e.printStackTrace();

    } finally {
        if (f != null) {
            // Inchidem fisierul
            try {
                f.close();
            } catch (IOException e) {
                System.err.println("Fisierul nu poate fi inchis!");
                e.printStackTrace();
            }
        }
    }
}

```

”Aruncarea” excepțiilor

Clauza throws

```
[modificatori] TipReturnat metoda([argumente])
    throws TipExceptie1, TipExceptie2, ...
{
    ...
}
```

Atenție

O metodă care nu tratează o anumită excepție trebuie obligatoriu să o ”arunce”.

```

public class CitireFisier {
    public static void citesteFisier(String fis)
        throws FileNotFoundException, IOException {

        FileReader f = null;
        f = new FileReader(fis);
        int c;
        while ( (c=f.read()) != -1)
            System.out.print((char)c);
        f.close();
    }
    public static void main(String args[]) {
        if (args.length > 0) {
            try {
                citesteFisier(args[0]);
            } catch (FileNotFoundException e) {
                System.err.println("Fisierul n-a fost gasit");
            } catch (IOException e) {
                System.out.println("Eroare la citire");
            }
        } else
            System.out.println("Lipseste numele fisierului");
    }
}

```

try - finally

```
public static void citesteFisier(String fis)
    throws FileNotFoundException, IOException {
    FileReader f = null;
    try {
        f = new FileReader(umeFisier);
        int c;
        while ( (c=f.read()) != -1)
            System.out.print((char)c);
    }
    finally {
        if (f!=null)
            f.close();
    }
}
```

Aruncarea erorilor din metoda main

```
public static void main(String args[])
    throws FileNotFoundException, IOException {
    citeste(args[0]);
}
```

Instrucțiunea throw

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

Avantajele tratării excepțiilor

- Separarea codului
- Propagarea erorilor
- Gruparea erorilor după tip.

Separarea codului

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

Cod "tradițional" ("spaghetti"):

```
int citesteFisier() {
    int codEroare = 0;
    deschide fisierul;
    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 { ...
        }
    }
    return codEroare; }
```

```
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;}
}
```

Propagarea erorilor

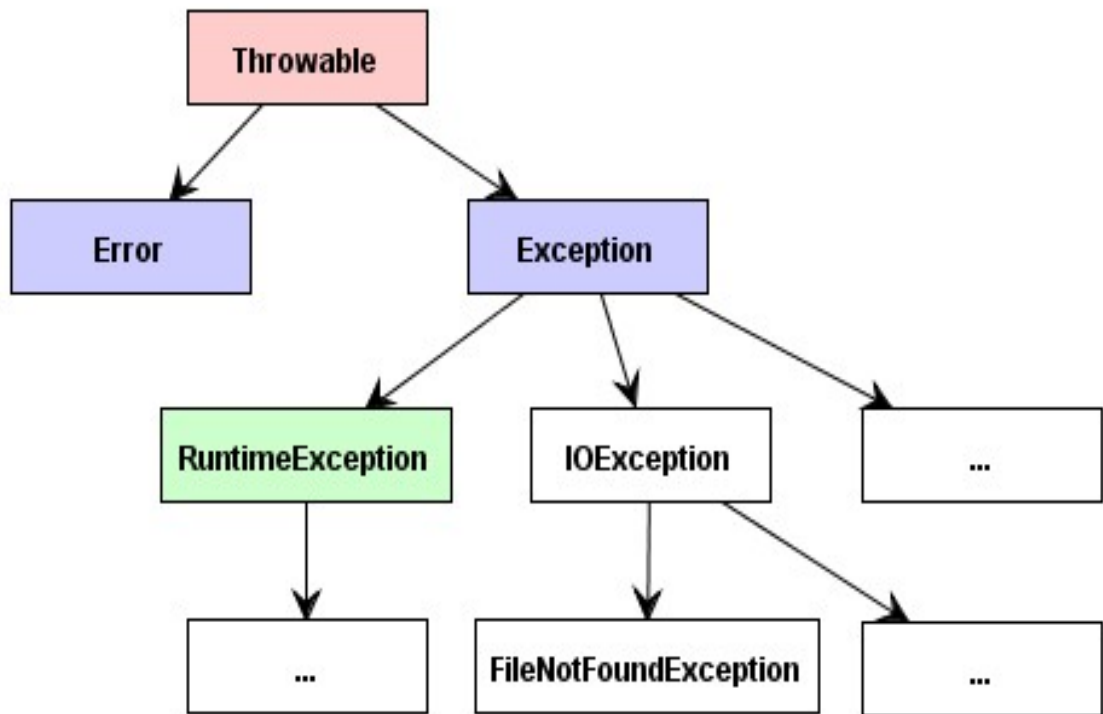
```
int metoda1() {
    try {
        metoda2();
    }
    catch (TipExceptie e) {
        //proceseazaEroare;
    }
    ...
}
int metoda2() throws TipExceptie {
    metoda3();
    ...
}
int metoda3() throws TipExceptie {
    citesteFisier();
    ...
}
```

Gruparea erorilor după tipul lor

- Fiecare tip de excepție este descris de o clasă.
- Clasele sunt organizate ierarhic.

```
try {
    FileReader f = new FileReader("input.dat");
    // Exceptie posibila: FileNotFoundException
}
catch (FileNotFoundException e) {
    // Exceptie specifica provocata de absenta
    // fisierului 'input.dat'
} // sau
catch (IOException e) {
    // Exceptie generica provocata de o operatie IO
} // sau
catch (Exception e) {
    // Cea mai generica exceptie soft
} //sau
catch (Throwable e) {
    // Superclasa exceptiilor
}
```

Ierarhia claselor ce descriu excepții



Metode:

- `getMessage`
- `printStackTrace`
- `toString`

Exceptii la executie

RuntimeException

- ArithmeticException
- NullPointerException
- ArrayIndexOutOfBoundsException

```
int v[] = new int[10];
try {
    v[10] = 0;
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Atentie la indecsi!");
    e.printStackTrace();
} // Corect, programul continua

v[11] = 0;
/* Nu apare eroare la compilare
   dar apare exceptie la executie si
   programul va fi terminat.
*/
System.out.println("Aici nu se mai ajunge...");
```

Crearea propriilor excepții

```
public class ExceptieProprie extends Exception {
    public ExceptieProprie(String mesaj) {
        super(mesaj);
        // Apeleaza constructorul superclasei Exception
    }
}

class ExceptieStiva extends Exception {
    public ExceptieStiva(String mesaj) {
        super(mesaj);
    }
}

class Stiva {
    int elemente[] = new int[100];
    int n=0; //numarul de elemente din stiva

    public void adauga(int x) throws ExceptieStiva {
        if (n==100)
            throw new ExceptieStiva("Stiva este plina!");
        elemente[n++] = x;
    }

    public int scoate() throws ExceptieStiva {
        if (n==0)
            throw new ExceptieStiva("Stiva este goala!");
        return elemente[n--];
    }
}
```

Alte tehnici de tratare a excepțiilor

”Exception chaining”

”Exception wrapping”

”Exception interception”