



# Programare avansată

## Tratarea excepțiilor

# Ce este o excepție?

"Eveniment excepțional" survenit în timpul execuției

```
public class Exemplu {  
    public static void main(String args[]) {  
        int v[] = new int[10];  
        v[10] = 0; //Excepție !  
        System.out.println("Hello world ...?!");  
    }  
}
```

"Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException :10  
at Exemplu.main (Exemplu.java:4) "

*"throw an exception"*

*"catch the exception"*

*"exception handler"*

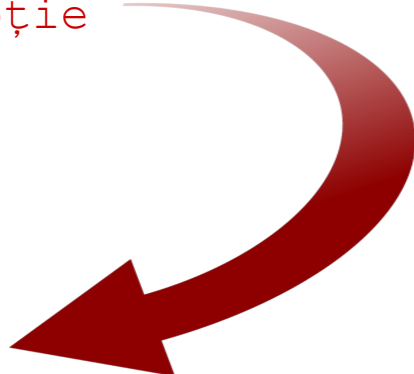
Tratarea excepțiilor nu mai este o opțiune ci o constrângere.

# Tratarea excepțiilor

## *try - catch - finally*

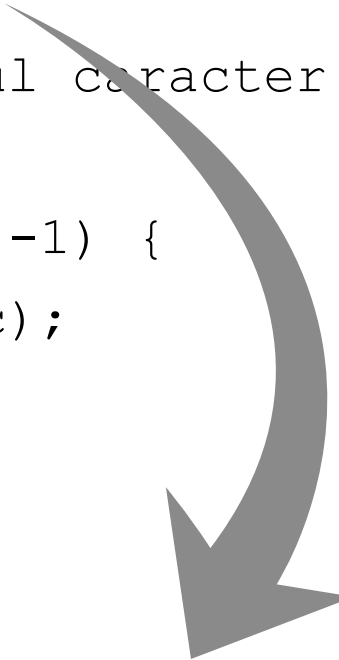
```
try {  
    // Bloc de instructiuni  
    metodaX()  
    metodaY()  
    metodaZ()  
}  
catch (TipExceptie1 variabila) {  
    // Tratarea exceptiilor de tipul 1  
}  
catch (TipExceptie2 variabila) {  
    // Tratarea exceptiilor de tipul 2  
}  
catch (TipExceptie3 | TipExceptie4 variabila) {  
    // Tratarea exceptiilor de tipul 3 sau 4  
}  
finally {  
    // Cod care se executa indiferent daca apar sau nu exceptii  
}  
...execuția continuă
```

→ S-a generat o excepție



# Exemplu – 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();  
}
```



**unreported exception FileNotFoundException;  
must be caught or declared to be thrown**

# “Prinderea” excepțiilor

```
public static void citeșteFisier(String fis) {
    FileReader f = null;
    try {
        f = new FileReader(fis);           // Deschidem fisierul
        int c;                             // Citim fisierul
        while ( (c=f.read()) != -1) {
            System.out.print((char)c);
        }
    } catch (FileNotFoundException e) {    //Tratam un tip de
excepție
        System.err.println("Fisierul " + fis + "nu a fost gasit!");
    } catch (IOException e) {              //Tratam alt tip de excepție
        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!");
            }
        }
    }
}
```

# “Aruncarea” excepțiilor

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

```
public class CitireFisier {  
    public static void citesteFisier(String fis)  
        throws FileNotFoundException, IOException {  
        FileReader f = new FileReader(fis);  
        int c;  
        while ( (c=f.read()) != -1)  
            System.out.print((char)c);  
        f.close();  
    }  
    public static void main(String args[]) {  
        try {  
            citesteFisier(args[0]);  
        } catch (FileNotFoundException e) {  
            System.err.println("Fisierul n-a fost gasit");  
        } catch (IOException e) {  
            System.out.println("Eroare la citire");  
        } catch (Exception e){  
            System.out.println("Ceva nu e in regula... Oare ce?");  
        }  
    }  
}
```

# *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();
    }
}
```

# *try-with-resources*

*Resurse* - obiecte care trebuie închise după utilizare; sunt de tipul *AutoCloseable*

```
try(FileReader f = new FileReader(numFisier)) {  
    int c;  
    while ( (c=f.read()) != -1)  
        System.out.print((char)c);  
}
```

```
try (Connection con = createConnection();  
    Statement stmt = con.createStatement();  
    ResultSet rs = stmt.executeQuery(query)) {  
  
    return (rs.next() ? rs.getObject(1) : null);  
} catch (SQLException e) {  
    System.err.println(e);  
}
```



# ✓ Separarea codului

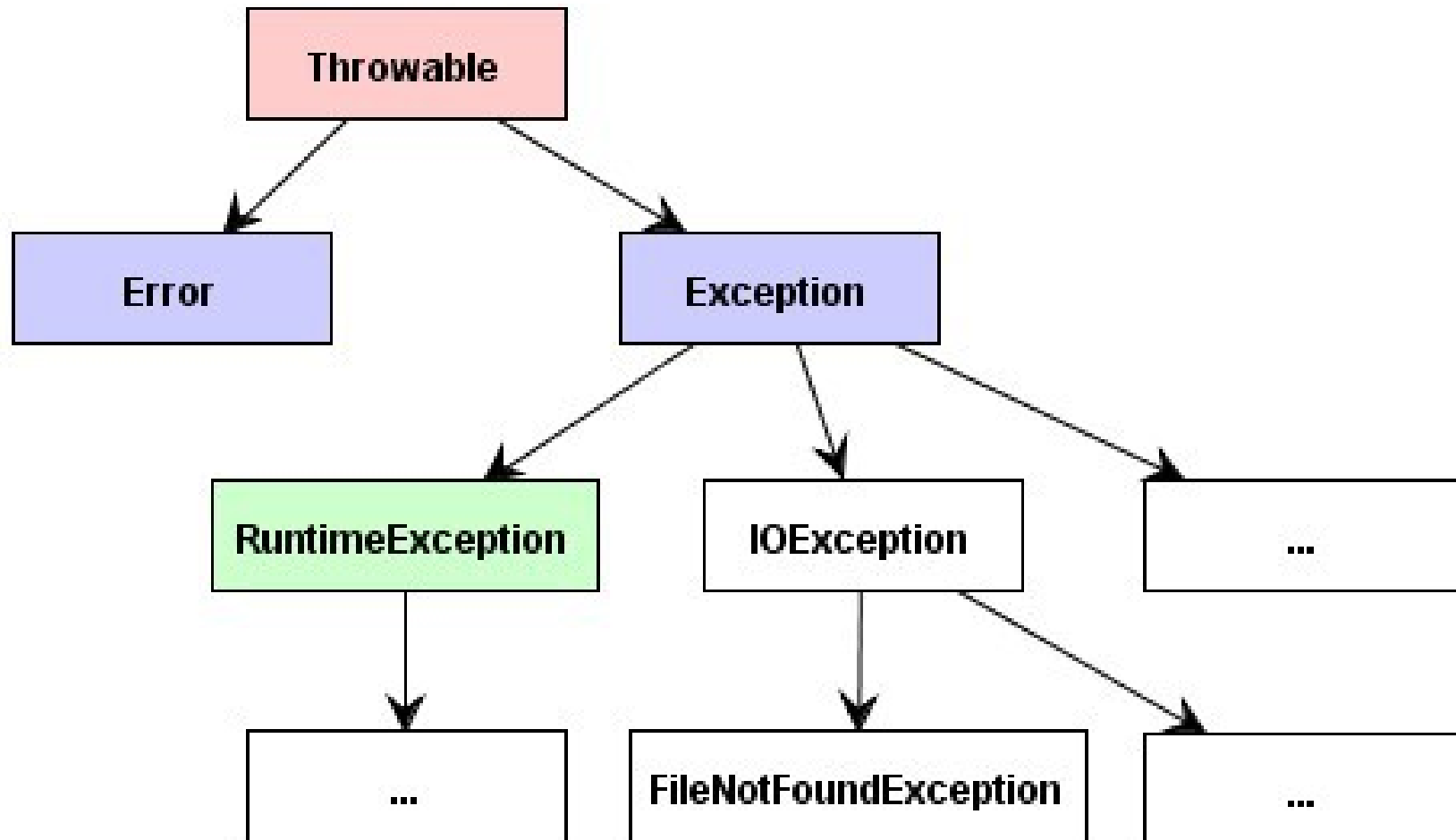
## Cod "traditional spaghetti":

```
int citesteFisier() {
    int codEroare = 0;
    deschide fisierul;
    if (fisierul s-a deschis) {
        determina dimensiunea fisierului;
        if (s-a determinat dimensiunea) {
            alocă memorie;
            if (s-a alocat memorie) {
                citeste fisierul in memorie;
                if (nu se poate citi din fisier){
                    codEroare = -1;
                }
            } else {
                codEroare = -2;
            }
        }
        ...
    }
    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 dim.){
        trateaza eroarea;
    } catch (nu s-a alocat memorie) {
        trateaza eroarea;
    } catch (nu se poate citi din fis.) {
        trateaza eroarea;
    } catch (nu se poate inchide fis.) {
        trateaza eroarea;
    }
}
```

# Ierarhia claselor care descriu excepții



# Checked versus Unchecked

- Checked Exceptions

- situații anormale, care nu pot fi controlate la momentul scrierii codului (*design-time*): fișier inexistent, erori de comunicare cu baze de date sau în rețea, etc.
- **trebuie explicit tratate** (“prinse” sau “aruncate”)
- extind **Exception**: `IOException`, `SQLException`, etc.

- Unchecked Exceptions

- erori din vina programului/programatorului, bug-uri
- **nu trebuie explicit tratate** (deși este posibil)
- extind **RuntimeException**: `NullPointerException`, `IllegalArgumentException`, `ArithmeticException`, `ArrayIndexOutOfBoundsException`, etc.

# *Error*

- Indică o problemă “serioasă”, care nu poate fi rezolvată într-un mod care să permită continuarea normală a execuției aplicației
- *Unchecked*
- Example:
  - `VirtualMachineError`
    - `InternalError`, `UnknownError`,
    - `OutOfMemoryError`, `StackOverflowError`

# ✓ Gruparea erorilor după tip

```
try {  
  
    String driverName = new String(  
        Files.readAllBytes(Paths.get("driver.txt")));  
  
    Class.forName(driverName).newInstance();  
  
} catch (IOException ex) {  
    // probleme cu fisierul din care vrem sa citim  
  
} catch (ClassNotFoundException ex) {  
    // nu exista clasa driver  
  
} catch (IllegalAccessException ex) {  
    // lipsa acces clasa  
  
} catch (InstantiationException ex) {  
    // clasa nu poate fi instantiata  
}
```

# Cine creează de fapt excepțiile?

Instrucțiunea **throw**

- Autorul unei metode va semnala situații excepționale creând și aruncând excepții

```
public class Stack {  
    ...                               //throws obligatoriu pentru exceptii checked  
    public Object peek() throws EmptyStackException {  
        int len = size();  
        if (len == 0) throw new EmptyStackException();  
        return elementAt(len - 1);  
    }  
}
```

- Mașina virtuală, în cazul excepțiilor “standard” din categoria **RuntimeException**


# Crearea propriilor excepții

- Extinderea unei clase existente din ierarhia cu rădăcina *Throwable*
- Decizie: *Checked* vs. *Unchecked*

```
public class ExceptieProprie extends RuntimeException {  
  
    //Proprietati si constructori  
  
    public ExceptieProprie(String mesaj) {  
        super(mesaj);  
        // Apeleaza constructorul superclasei  
    }  
}
```

# ✓ Propagarea excepțiilor

```
int metoda1() {  
    try {  
        metoda2();  
    } catch (TipExceptie e) {  
        //proceseaza exceptie  
    }  
}  
int metoda2() throws TipExceptie {  
    ...  
    metoda3();  
    ...  
}  
int metoda3() throws TipExceptie {  
    ...  
    throw new TipExceptie();  
    ...  
}
```





# *Exception Chaining (Wrapping)*

```
try {
    Person person =
        database.readPerson(personId);

} catch (SQLException sqlException) {
    // prindem exceptia initiala
    System.err.println(sqlException);

    // cream o exceptie proprie
    // care incapsuleaza exceptia initiala
    MyException ex =
        new MyException("Database Error", sqlException);
    ex.setDetail("Invalid person id " + personId);

    // aruncam exceptia proprie
    throw ex;
}
```

