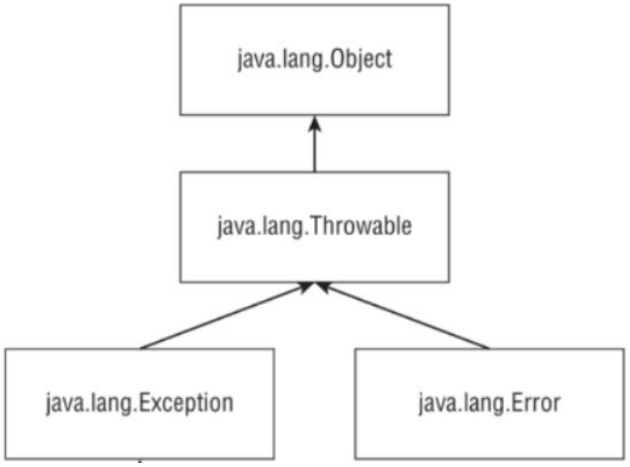
Laborator 6

Excepţii

- Excepţiile sunt evenimente care întrerup executia normala al unui program.
- O excepție este un obiect ce aparține clasei predefinite **Throwable** sau unei clase descendente din clasa Throwable. (Clasa Throwable o găsim în pachetul java.lang)
- Cele două subclase principale care mostenesc Throwable sunt **Error** si **Exception**. Ambele clase le regasiti in pachetul java.lang.

Ierarhia poate fi observată în imaginea de mai jos:



Cele doua clase au fost introduse pentru a delimita doua tipuri fundamentale de excepţii ce pot apărea într-o aplicaţie Java.

Error

- Clasa Error corespunde excepţiilor ce nu mai pot fi recuperate de către programator. Nu pot fi anticipate şi nu sunt în general tratate explicit într-o aplicaţie.
- Apariţia unei astfel de excepţii înseamnă ca a apărut o eroare deosebit de grava care a determinat încetarea executării unui program.
- Exemple de erori: **StackOverflowError**, **OutOfMemoryError**, **VirtualMemoryError** etc.

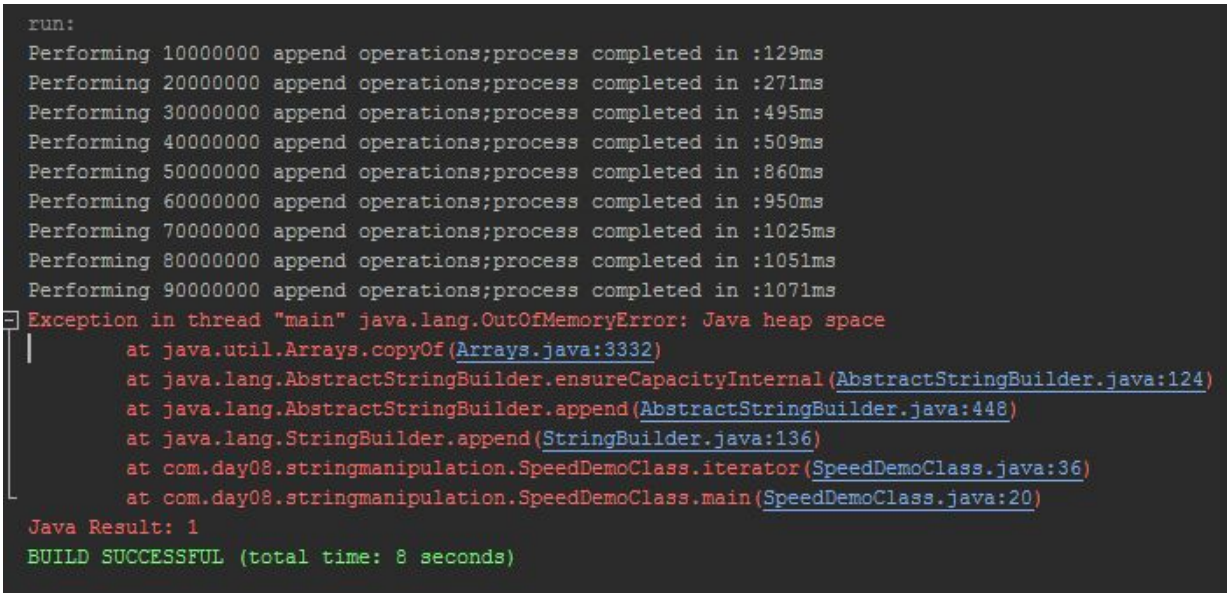


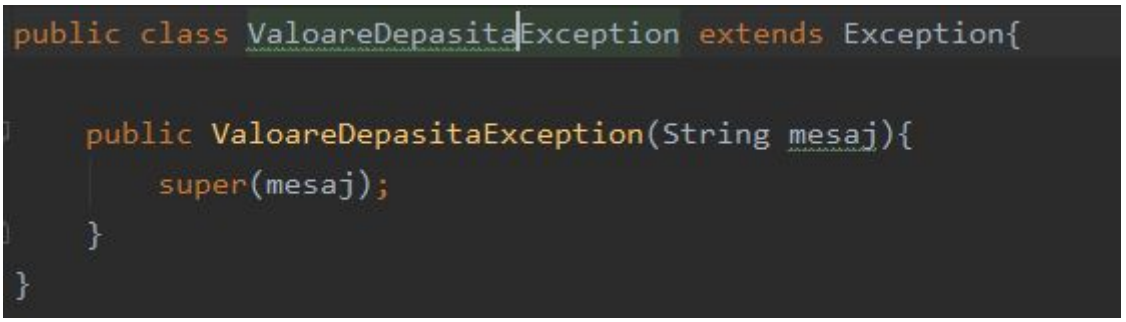
Fig 1 - Exemplu de **OutOfMemoryError** în timpul execuţiei unui program

Exception

- Clasa Exception este , de fapt, cea utilizată efectiv în procesul de tratare a excepţiilor.
- Atat clasa Exception cat şi descendentii ei se ocupă de excepţii ce pot fi rezolvate de către program, fără oprirea acestuia.
- Descendentii din clasa Exception pot fi de doua tipuri: **Checked Exceptions** şi **Unchecked Exceptions** (sau **RuntimeExceptions**).
- Excepţiile de tip **checked** sunt tratate la compilare. Toate excepţiile care mostenesc Clasa Throwable cu excepţia RuntimeException şi Error sunt cunoscute ca checked exceptions (IOException, SQLException, FileNotFoundException etc)

Fig 2 - Exemplu de excepţie checked semnalată la compilare

- În cazul excepţiilor **checked**, pot exista situaţii cand trebuie tratate anumite excepţii specifice de business, care sa tina doar de logica aplicaţiei, de exemplu un obiect poate sa contina campuri numerice care accepta valori într-un anumit interval. În cazul acesta trebuie sa ne definim noi un nou tip de excepţie care sa fie tratata în aplicaţie.

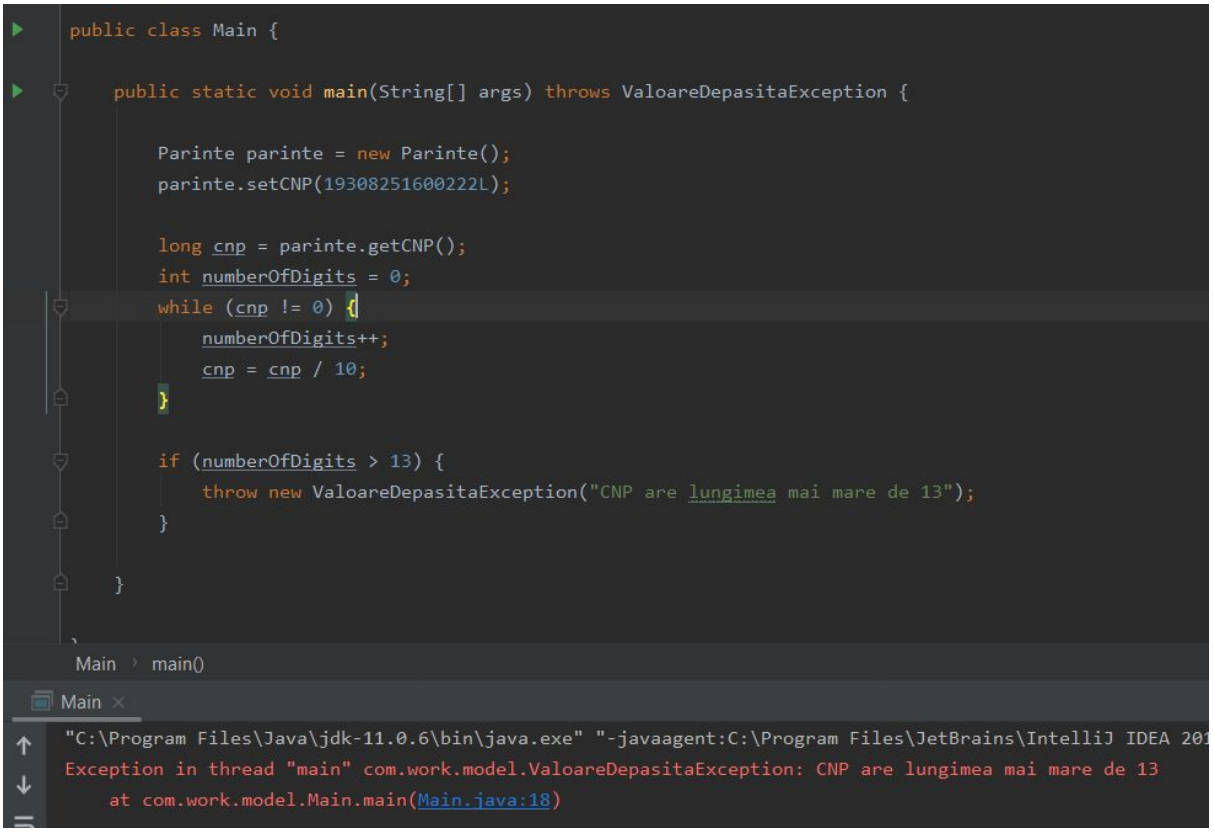


```
public class ValoareDepasitaException extends Exception{

    public ValoareDepasitaException(String mesaj){
        super(mesaj);
    }
}
```

Fig 3 - Exemplu de **excepţie definită**

  - Metoda **super** apelează constructorul din clasa excepţie părinte (în cazul nostru, Exception).



```
public class Main {

    public static void main(String[] args) throws ValoareDepasitaException {

        Parinte parinte = new Parinte();
        parinte.setCNP(19308251600222L);

        long cnp = parinte.getCNP();
        int numberOfDigits = 0;
        while (cnp != 0) {
            numberOfDigits++;
            cnp = cnp / 10;
        }

        if (numberOfDigits > 13) {
            throw new ValoareDepasitaException("CNP are lungimea mai mare de 13");
        }

    }
```

```
Main › main()
Main
"C:\Program Files\Java\jdk-11.0.6\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 201
Exception in thread "main" com.work.model.ValoareDepasitaException: CNP are lungimea mai mare de 13
    at com.work.model.Main.main(Main.java:18)
```

  - Pentru a arunca o excepţie se foloseşte keyword-ul **throw:**

throw new Exceptie1();

- Excepţiile de tip **unchecked** nu sunt tratate la compilare şi apar la execuţia programului. (ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc)

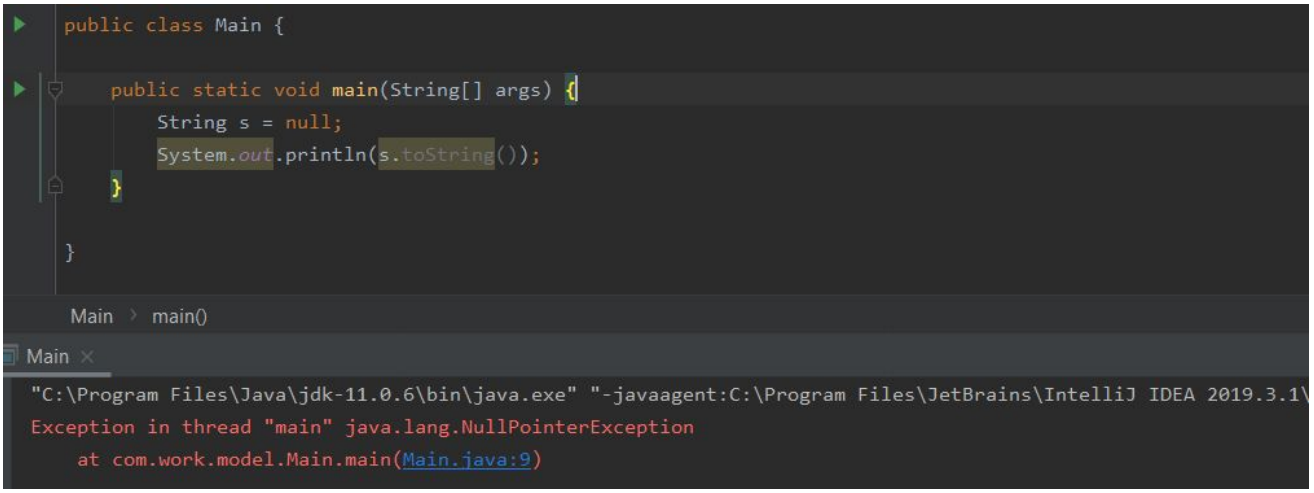


Fig 4 - Exemplu de **NullPointerException**

Tratarea Excepţiilor

- O data ce o excepţie a fost aruncată, trebuie tratată.
- Tratarea excepţiilor se realizeaza prin intermediul blocurilor de instrucţiuni **try**, **catch** şi **finally**.
- O secvenţa de cod care tratează anumite excepţii trebuie sa respecte următoarea structura:

```
try {
        // Instrucţiuni care pot genera excepţii
}
catch (TipExceptie1 variabila) {
        //  Tratarea excepţiilor de tipul 1
}
catch (TipExceptie2 variabila) {
        // Tratarea excepţiilor de tipul 2
}
...
finally {
        // Cod care se executa indiferent dacă apar sau nu excepţii
}
```

- **try** - keyword-ul “try” este folosit sa specifice un bloc de instrucţiuni care poate arunca o excepţie. El trebuie urmat fie de **catch** sau **finally**.
- **catch** - block-ul catch este folosit să tratam excepţiile aruncate. El trebuie precedat de block-ul try. Dacă nu există nici un bloc catch corespunzător excepției emise, aceasta va fi propulsată spre apelanții funcției curente, căutându-se un eventual bloc try înconjurător.
- **finally** - block-ul finally contine un set de instrucţiuni care o sa fie rulat indiferent dacă o excepţie este tratată sau nu. (de exemplu dacă vrem sa inchidem un flux de citire)

```
public static void citireFisier( String numeFisier ) {
    FileReader f = null ;

    try {
         // Deschidem fisierul
        System.out.println("Deschidem fisierul " + numeFisier);
        f = new FileReader(numeFisier);

        int c;

        while ( (c = f.read()) != -1){
            System.out.println((char) c);
        }
    } catch (FileNotFoundException e) {
        // Tratam un tip de exceptie
        System.out.println("Fisierul nu a fost gasit!");
        System.out.println("Exceptie: " + e.getMessage());
    } catch (IOException e ) {
        // Tratam alt tip de exceptie
        System.out.println("Eroare la citirea din fisier!");
        e.printStackTrace();
    }
    finally {
        if (f != null){
            // Inchidem fisierul
            System.out.println("Inchidem fisierul!");
        }

        try{
            f.close();
        } catch (IOException e) {
            System.out.println("Fisierul nu poate fi inchis!");
            e.printStackTrace();
        }
    }
}
```

Nota:
Block-urile try-catch tratează excepţiile de la cea mai specifica la cea mai generala.

Ce se intampla în cazul asta ?

```
try
    fall();
catch (Exception e)
    System.out.println("get up");
```

Dar acum?

```
try {
    fall();
}
```

Ce se afiseaza in consola?

```
public class Exception_Exercise1 {

    public static void main(String[] args) {
        String s = "";
        try {
            s += "t";
        } catch (Exception e) {
            s += "c";
        } finally {
            s += "f";
        }
        s += "a";

        System.out.print(s);
    }
}
```

- Incepand cu Java 7, putem sa tratam mai multe excepţii în acelaşi block catch.

Ce se intampla în exemplele următoare?

catch(Exception1 e | Exception2 e | Exception3 e) // ???

catch(Exception1 e1 | Exception2 e2 | Exception3 e3) //???

catch(Exception1 | Exception2 | Exception3 e) //???

- Sintaxa este similară cu cea a unui catch normal, doar ca doua sau mai multe excepţii sunt specificate

```
try {
        // instructiuni
} catch ( Exception1 | Exception 2 e){
        // tratam excepția
}
```

Sintaxa corecta multi-catch:

```
public class MultiCatchExceptions {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        try {
            System.out.println(y % x);
        } catch (NumberFormatException | ArithmeticException | ArrayIndexOutOfBoundsException e2) {

        }
    }
}
```

- Multi-catch-urile sunt folosite pentru excepţii care nu au o legatura intre ele.
- Nu pot fi specificate tipuri redundante într-un multi-catch.

Fluxuri de intrare/ieşire

- Un flux reprezinta un canal de comunicatie intre doua procese.
- Programele pot avea nevoie sa **preia** informaţii **de la surse** externe, sau sa **trimita** informaţii **către destinaţii** externe.
- Sursa și/sau destinaţia pot fi:
    - **fişier pe disc**
    - **reţea**
    - **memorie (alt program)**
    - **dispozitive IO standard (ecran, tastatura)**.

- Pachetul care oferă suport pentru operaţiile de intrare/iesire este java.io

- Flow-ul pentru prelucrarea fluxurilor este următorul:

  deschide canal comunicatie
  while (mai sunt informaţii) {
  citeşte/scrie informaţie;
  }
  închide canal comunicatie;

- Cand folosim operaţiile de intrare/ieşire ne putem lovi de excepţii:
  - **EOFException** la intalnirea sfarsitului unui fişier;
  - **FileNotFoundException** cand incercam sa deschidem un fisier inexistent;
  - **MalformedURLException** la accesarea unei adrese web invalide;
  - **InterruptedIOException** la întreruperea unei operaţii de intrare/ieşire, în timpul execuţiei.

- În funcţie de tipul de date transferat, fluxurile se împart în doua categorii:
  - fluxuri de octeţi (clasele de baza **InputStream**, **OutputStream**)
  - fluxuri de caractere (clase de baza **Reader**, **Writer**)

Metodele de baza:
• **int read();**
• **int read(byte buf[]);**
• **int write();**
• **int write(byte buf[]).**

Clasificare în funcţie de acţiunea pe fluxuri:

- fluxuri primare: **FileReader, FileWriter, FileInputStream, FileOutputStream**

```java
public class MainIO_Ex1 {

    public static void main(String [] args) throws IOException {
        //crearea unui flux de intrare pe caractere
        FileReader fileReader = new FileReader( fileName: "input.txt");

        //crearea unui flux de iesire pe caractere
        FileWriter fileWriter = new FileWriter( fileName: "output.txt");

    }
}
```

- fluxuri de procesare: **BufferedReader, BufferedWriter, BufferedInputStream, BufferedOutputStream.**

```java
public class Ex2 {

    public static void main(String[] args) throws IOException {
        // trebuie sa dam ca parametru la un flux de procesare, un flux primar
        BufferedReader in = new BufferedReader(new FileReader( fileName: "fisier.txt"));

        FileReader filein = new FileReader( fileName: "fisier.txt");
        BufferedReader bufferedReader = new BufferedReader(filein);

        System.out.println(bufferedReader.readLine());
    }
}
```

Ierarhia claselor pentru fluxuri

- fluxuri de octeţi pentru citire:
  - **FileInputStream**
  - **BufferedInputStream**
- fluxuri de octeţi pentru scriere:
  - **FileOutputStream**
  - **BufferedOutputStream**
- fluxuri de caractere pentru citire:
  - **FileReader**
  - **BufferedReader**
- fluxuri de caractere pentru scriere:
  - **FileWriter**
  - **BufferedWriter**

Tema

1. Să se scrie un program care citeşte de la tastatură perechi de numere în care primul număr trebuie să fie mai mic decât al doilea. Dacă această condiție nu este îndeplinită, folosind mecanismul excepțiilor, se va semnala eroare şi se va trata această eroare prin cererea unei alte perechi de numere. Toate perechile de numere care îndeplinesc condiția vor fi scrise într-un fişier.
2. Sa se citeasca de la tastatură un user ,şi o parolă. Acestea se vor compara cu înregistrările existente în fişierul parole.txt.
   Dacă user-ul şi parola se regăsesc printre acestea (pe aceeaşi linie), se va afişa mesajul ”acces permis”, dacă se regaseste doar userul, iar parola este greşită se va afişa ”parola greşită” şi se va mai cere introducerea parolei încă o data, dar nu mai mult de 3 ori, dacă se atinge acest prag se va afişa mesajul ”cont blocat”.
   În caz contrar se reia procesul de introduce a datelor, dar nu mai mult de 5 ori. Dacă se atinge limita de 5 intrări se va afişa mesajul ”Nu ai cont. Inregistreaza-te.”
   (Se vor folosi clasele FileInputStream şi FileOutputStream.)

   Exemplu de date în fişierul parole.txt:
   user user
   gigi parolaMea
   user1 parola1

3. Într-un fişier numit clienţi.txt sunt memorate date despre clienţii unui magazin virtual. Pe fiecare linie se reţine numele, prenumele ,şi varsta clienţilor. Se cere sa se afişeze numărul, şi lista clienţilor majori, şi numărul clienţilor minori.