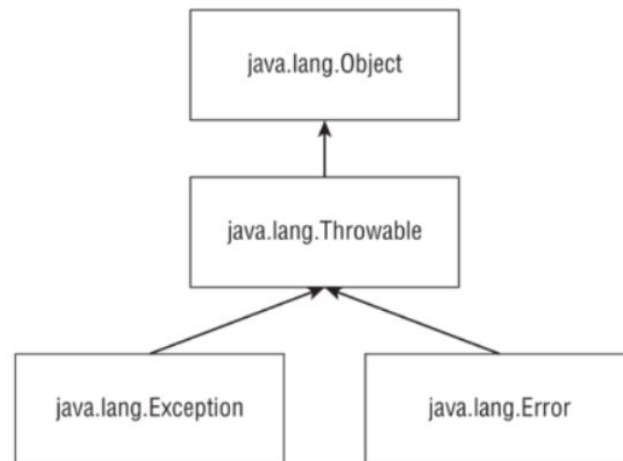


Laborator 6

Excepții

- Excepțiile 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 două clase au fost introduse pentru a delimita două tipuri fundamentale de excepții ce pot apărea într-o aplicație Java.

Error

- Clasa Error corespunde excepțiilor ce nu mai pot fi recuperate de către programator. Nu pot fi anticipate și nu sunt în general tratate explicit într-o aplicație.
- Apariția unei astfel de excepții înseamnă că a apărut o eroare deosebit de gravă care a determinat încetarea executării unui program.
- Exemple de erori: **StackOverflowError**, **OutOfMemoryError**, **VirtualMemoryError** etc.

```
run:
Performing 10000000 append operations;process completed in :129ms
Performing 20000000 append operations;process completed in :271ms
Performing 30000000 append operations;process completed in :495ms
Performing 40000000 append operations;process completed in :509ms
Performing 50000000 append operations;process completed in :860ms
Performing 60000000 append operations;process completed in :950ms
Performing 70000000 append operations;process completed in :1025ms
Performing 80000000 append operations;process completed in :1051ms
Performing 90000000 append operations;process completed in :1071ms
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.util.Arrays.copyOf(Arrays.java:3332)
    at java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.java:124)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:448)
    at java.lang.StringBuilder.append(StringBuilder.java:136)
    at com.day08.stringmanipulation.SpeedDemoClass.iterator(SpeedDemoClass.java:36)
    at com.day08.stringmanipulation.SpeedDemoClass.main(SpeedDemoClass.java:20)
Java Result: 1
BUILD SUCCESSFUL (total time: 8 seconds)
```

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

Exception

- Clasa Exception este , de fapt, cea utilizată efectiv în procesul de tratare a excepțiilor.
- Atat clasa Exception cat și descendentele ei se ocupă de excepții ce pot fi rezolvate de către program, fără oprirea acestuia.
- Descendentele din clasa Exception pot fi de doua tipuri: **Checked Exceptions** și **Unchecked Exceptions** (sau **RuntimeExceptions**).
- Excepțiile de tip **checked** sunt tratate la compilare. Toate excepțiile care mostenesc Clasa Throwable cu excepția RuntimeException și Error sunt cunoscute ca checked exceptions (IOException, SQLException, FileNotFoundException etc)

```
public class Main {

    public static void main(String[] args) {
        FileInputStream fis = null;

        fis = new FileInputStream("File.txt");
    }
}
```

Unhandled exception: java.io.FileNotFoundException

Add exception to method signature Alt+Shift+Enter More actions... Alt+Enter

Fig 2 - Exemplu de excepție checked semnalată la compilare

- În cazul excepțiilor **checked**, pot exista situații cand trebuie tratate anumite excepții specifice de business, care sa tina doar de logica aplicației, 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 excepție care sa fie tratata în aplicație.

```
public class ValoareDepasitaException extends Exception{

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

Fig 3 - Exemplu de excepție definită

- Metoda **super** apelează constructorul din clasa excepție 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 x

"C:\Program Files\Java\jdk-11.0.6\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.3.1\lib\idea_rt.jar=1273.0:C:\Program Files\JetBrains\IntelliJ IDEA 2021.3.1\bin" -Dfile.encoding=UTF-8

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 excepție se folosește keyword-ul **throw**:

throw new Exceptie1();

- Excepțiile de tip **unchecked** nu sunt tratate la compilare și apar la execuția programului. (ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc)

```

public class Main {
    public static void main(String[] args) {
        String s = null;
        System.out.println(s.toString());
    }
}

```

Main > main()

Exception in thread "main" java.lang.NullPointerException
 at com.work.model.Main.main(Main.java:9)

Fig 4 - Exemplu de **NullPointerException**

Tratarea Excepțiilor

- O data ce o excepție a fost aruncată, trebuie tratată.
- Tratarea excepțiilor se realizează prin intermediul blocurilor de instrucțiuni **try**, **catch** și **finally**.
- O secvență de cod care tratează anumite excepții trebuie să respecte următoarea structură:

```

try {
    // Instrucțiuni care pot genera excepții
}
catch (TipExcepție1 variabila) {
    // Tratarea excepțiilor de tipul 1
}
catch (TipExcepție2 variabila) {
    // Tratarea excepțiilor de tipul 2
}
...
finally {
    // Cod care se execută indiferent dacă apar sau nu excepții
}

```

- **try** - keyword-ul "try" este folosit sa specifice un bloc de instrucțiuni care poate arunca o excepție. El trebuie urmat fie de **catch** sau **finally**.
- **catch** - block-ul catch este folosit să tratam excepțiile aruncate. El trebuie precedat de block-ul try. Dacă nu există nici un bloc catch corespunzător excepției emise, aceasta va fi propulsată spre apelantii funcției curente, căutându-se un eventual bloc try înconjurător.
- **finally** - block-ul finally conține un set de instrucțiuni care o sa fie rulat indiferent dacă o excepție 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) {
        // Tratatam un tip de exceptie
        System.out.println("Fisierul nu a fost gasit!");
        System.out.println("Exceptie: " + e.getMessage());
    } catch (IOException e) {
        // Tratatam 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 excepțiile 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);
    }
}
```

- Inceput cu Java 7, putem sa tratam mai multe excepții în același 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 excepții 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 excepții care nu au o legatura intre ele.
- Nu pot fi specificate tipuri redundante într-un multi-catch.

Fluxuri de intrare/ieșire

- Un flux reprezinta un canal de comunicatie intre doua procese.
- Programele pot avea nevoie sa **preia** informații **de la surse** externe, sau sa **trimita** informații **către destinații** externe.
- Sursa și/sau destinația pot fi:
 - **fișier pe disc**
 - **rețea**
 - **memorie (alt program)**
 - **dispozitive IO standard (ecran, tastatura).**
- Pachetul care oferă suport pentru operațiile de intrare/iesire este java.io

- Flow-ul pentru prelucrarea fluxurilor este următorul:

deschide canal comunicatie

```
while (mai sunt informatii) {  
    citește/scrie informație;  
}
```

închide canal comunicatie;

- Cand folosim operațiile de intrare/ieșire ne putem lovi de excepții:
 - **EOFException** la intalnirea sfarsitului unui fișier;
 - **FileNotFoundException** cand incercam sa deschidem un fisier inexistent;
 - **MalformedURLException** la accesarea unei adrese web invalide;
 - **InterruptedException** la întreruperea unei operații de intrare/ieșire, în timpul execuției.
- În funcție de tipul de date transferat, fluxurile se împart în doua categorii:
 - fluxuri de octeți (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 funcție de acțiunea pe fluxuri:

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


```

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**.

```

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 octeți pentru citire:
 - **FileInputStream**
 - **BufferedInputStream**
- fluxuri de octeți 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 citește 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 semnaliza eroare și se va trata această eroare prin cererea unei alte perechi de numere. Toate perechile de numere care îndeplinesc condiția vor fi scrise într-un fișier.
2. Sa se citeasca de la tastatura un **user** și o **parola**. Acestea se vor compara cu înregistrările existente în fișierul parole.txt. Dacă se regasesc printre acestea, se va afișa mesajul "acces permis". În caz contrar se va relua procesul.
Exemplu de date în fișierul parole.txt:
user user
gigi parolaMea
user1 parola1
3. Într-un fișier numit clienți.txt sunt memorate date despre clienții unui magazin virtual. Pe fiecare linie se reține numele, prenumele, și vârsta clienților. Se cere să se afișeze numărul, și lista clienților majori, și numărul clienților minori.