Programarea orientată pe obiecte

SERIA CB

GENERALITĂȚI DESPRE EXCEPȚII

```java
import java.util.Scanner;

public class Calcul {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Introduceti doua valori: ");
        int a = s.nextInt();
        int b = s.nextInt();
        System.out.println("Rezultat = " + (a/b));
    }
}
```

```java
import java.util.Scanner;

public class Calcul {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Introduceti doua valori: ");
        int a = s.nextInt();
        int b = s.nextInt();
        if(b!= 0)
            System.out.println("Rezultat=" + (a/b));
        else
            System.out.println("Eroare la impartire");
    }
}
```

O **excepție** este un obiect care prezintă o condiție ce previne execuția normală a programului. Dacă excepția nu este "prinsă" (en., **catch**), programul nu va funcționa normal.

Dacă pentru a doua valoare se introduce `0`, atunci va apărea o eroare la rulare (`Runtime Exception`) deoarece nu se poate împărți un număr la `0`. O modalitate de rezolvare "clasică" poate fi prin adăugarea unei condiții de testare a valorii celui de-al doilea număr.

Programul conține un bloc `try` și un bloc `catch`.

Blocul `try` conține cod care este executat în condiții de funcționare normală.

O altă metodă de rezolvare presupune prinderea excepției aruncate de împărțirea la `0`:

```
import java.util.Scanner;

public class Calcul {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Introduceti doua valori: ");
        int a = s.nextInt();
        int b = s.nextInt();
        try {
            System.out.println("Rezultat=" + (a/b));
        }
        catch (ArithmeticException ex) {
            System.out.println("Situatie de exceptie");
        }
        System.out.println("Executia continua ");
    }
}
```

O metodă poate arunca o excepție (en., `throw`). Instrucțiunea `throw` este asemănătoare unui apel de metodă, care apelează blocul `catch`.

Blocul `catch` este ca o definiție de metodă cu un parametru compatibil tipului valorii aruncate. După ce acesta este executat, controlul programului revine următoarei instrucțiuni de după blocul `catch`.

```
import java.util.Scanner;

public class Calcul2 {
    public static int calcul(int a, int b) {
        if (b == 0)
            throw new ArithmeticException("Situatie de exceptie ");
        return a / b;
    }
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Introduceti doua valori: ");
        int a = s.nextInt();
        int b = s.nextInt();
        try {
            int rez = calcul(a, b);
            System.out.println("Rezultat=" + rez);
        }
        catch (ArithmeticException ex) {
            System.out.println("Situatie de exceptie");
        }
        System.out.println("Executia continua");
    }
}
```

```
import java.util.Scanner;

import java.io.*;
public class Exemplu {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Nume fisier: ");
        String numeFisier = s.nextLine();
        try {
            Scanner sf = new Scanner(new File(numeFisier));
            System.out.println("Fisierul " + numeFisier + " exista deja ");
        }
        catch (FileNotFoundException ex) {
            System.out.println("Exceptie: " + numeFisier + " nu a fost gasit");
        }
    }
}
```

Excepțiile de tip `Runtime` apar atunci când la rulare este detectată o operație ce nu mai poate fi dusă la bun sfârșit. De exemplu, dacă se încearcă accesarea unui element de pe o poziție a unui tablou ce depășește limitele acestuia, va apărea excepția `ArrayIndexOutOfBoundsException`. Dacă se încearcă deschiderea unui fișier care nu există, excepția va fi `FileNotFoundException`.

Blocul `catch` conține cod care este executat atunci când `b` este `0`. În acest caz, programul aruncă o excepție.

Excepția este un obiect creat dintr-o clasă excepție. În acest caz, clasa excepție este `java.lang.ArithmeticException`.

Când o excepție este aruncată, fluxul execuției este întrerupt. Codul din blocul `catch` este executat pentru a manipula excepția.

Identificatorul `ex` din `catch (ArithmeticException ex)` se aseamănă unui parametru dintr-o metodă și este parametrul blocului `catch`.

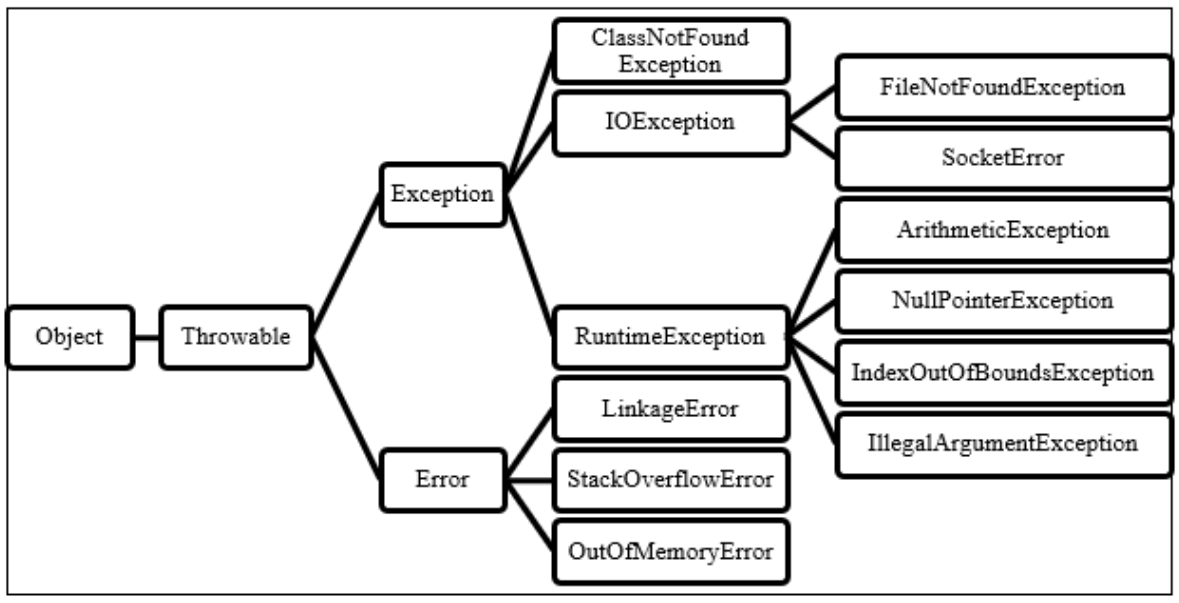
Tipul său, în acest caz `ArithmeticException`, arată ce tip de excepție poate prinde blocul `catch`.

O excepție poate fi aruncată prin utilizarea unei instrucțiuni `throw` în blocul `try`, sau prin invocarea unei metode ce aruncă o excepție.

Diverse metode din bibliotecile JDK aruncă deja excepții. Mai jos este prezentat un exemplu în care este manipulată excepția `FileNotFoundException`.

TIPURI DE EXCEPȚII

O clasificare a claselor de excepții în Java este afișată în figura de mai jos. Exemplele prezentate în figură nu sunt exhaustive, în Java existând o multitudine de clase de tip `Exception` sau `Error` predefinite.



`Throwable` este clasa de bază din care sunt derivate excepțiile!

Se pot crea noi clase prin extinderea lui `Exception` sau a unei subclase din `Exception`. Clasele de obiecte de tipul `Throwable` pot fi clasificate astfel: erori de sistem (**Error**) și excepții (**Exception**).

- Erorile de sistem sunt aruncate de JVM și reprezentate de clasa **Error**. Aceste erori apar rar. În situația în care apar, utilizatorul trebuie notificat, iar programul se va încheia. De exemplu, `VirtualMachineError` apare atunci când JVM este blocat sau când nu mai are resurse pentru a rula programul.
- Excepțiile sunt reprezentate de clasa **Exception**, și pot fi prinse în program. Două exemple de excepții sunt `ClassNotFoundException` și `IOException`, legate de operații invalide de input și output. Câteva subclase cunoscute ale lui `IOException` sunt `InterruptedIOException`, `EOFException` sau `FileNotFoundException`.

Excepțiile de *runtime* (**RuntimeException**) sunt o subcategorie aparte de excepții și cuprind excepții care pot prinde erori de programare precum: împărțirea la zero sau acces incorect la o zonă de memorie nealocată a unui tablou. Excepțiile de runtime sunt aruncate de JVM. Câteva exemple sunt: `ArithmeticException`, `NullPointerException`, `IndexOutOfBoundsException`, `IllegalArgumentException`.

Excepțiile `RuntimeException`, respectiv erorile de sistem `Error`, precum și subclasele lor sunt cunoscute ca ***unchecked exceptions***. Toate celelalte excepții sunt ***checked exceptions***, ceea ce înseamnă că compilatorul forțează programatorul să le rezolve.

> *În contrast cu excepțiile checked, o excepție unchecked reprezintă o eroare în logica de programare (și nu o situație eronată), care ar putea apărea în mod rezonabil atunci când programul rulează (la runtime).*

Excepțiile runtime (fiind unchecked) nu trebuie (în mod obligatoriu) prinse folosind `try-catch`. Se poate arunca o excepție `RuntimeException` dintr-o metodă fără a o menționa în clauza `throws` din antetul metodei (tocmai pentru că în metoda apelantă nu este obligatoriu ca excepția să fie prinsă):

```
public void f(Object o) {
    if (o == null)
        throw new NullPointerException("o este null");
```

DECLARAREA EXCEPȚIILOR

Fiecare metodă trebuie să precizeze tipul de excepție pe care o aruncă (valabil pentru excepțiile de tip ***checked*)**. Acest lucru mai este cunoscut ca ***declararea excepțiilor***.

Java nu impune declararea excepțiilor ***unchecked*,** așa cum s-a mai specificat anterior. În schimb, toate celelalte excepții aruncate de metodă trebuie să fie explicit declarate în antetul metodei. De exemplu:

```
public void metoda() throws Exceptia1, Exceptia2, ...
```

Dacă o metodă suprascrisă nu declară excepțiile în clasa de bază, atunci nu poate fi suprascrisă să declare excepțiile în subclasă.

ARUNCAREA EXCEPȚIILOR

Un program care detectează o eroare poate crea o instanță a unei excepții și să o arunce.

```
throw new IllegalArgumentException("Argument greșit");
```

`IllegalArgumentException` este o clasă excepție în Java. Fiecare clasă excepție are doi constructori [1] – unul fără parametri și unul cu un parametru de tipul `String` ce descrie excepția. Parametrul este numit exception message și poate fi obținut folosind apelul `getMessage()` [2].

PRINDEREA EXCEPȚIILOR

Când o excepție este aruncată, ea poate fi prinsă și manipulată astfel:

```
try {
// secventa de cod
}
catch (Exceptia1 ex1) {
// prindere Exceptia1
}
catch (Exceptia2 ex2) {
// prindere Exceptia2
}
catch (Exceptia3 ex3) {
// prindere Exceptia3
}
```

Dacă nu apar excepții în timpul execuției blocului `try`, atunci nu se intră în niciunul dintre blocurile `catch`. Dacă una dintre secvențele din interiorul blocului `try` aruncă o excepție, sunt ignorate celelalte linii din interiorul blocului `try` și se caută blocul `catch` ce prinde acea excepție, numit *exception handler*. Fiecare bloc `catch` este examinat pe rând și se execută codul din blocul corespunzător găsit. Dacă nu este găsit niciun bloc `catch` compatibil excepției, programul se finalizează (se închide) cu un mesaj de eroare la consolă.

Diverse clase de excepții pot fi derivate dintr-o superclasă comună. Dacă un bloc `catch` prinde obiecte de tip excepție ale superclasei, atunci poate prinde și excepțiile claselor derivate din aceasta.

Ordinea în care excepțiile sunt specificate în blocurile catch este foarte importantă! O eroare de compilare va apărea dacă blocul catch al superclasei apare înainte de blocul catch al subclasei.

Exemplul următor este greșit:

```
try {
...
}
catch (Exception ex) {
...
}
catch (RuntimeException ex) {
...
}
```

Deoarece clasa `RuntimeException` este o subclasă a clasei `Exception`, ordinea corectă a blocurilor `catch` este următoarea (în caz contrar, apare eroare de compilare):

```
try {
...
}
catch (RuntimeException ex) {
...
}
catch (Exception ex) {
...
}
```

Evitați scrierea unei secvențe care prinde orice excepție, deoarece astfel se pierde din specificitatea programului!

```
try {
    ...
}
catch (Exception e) {
    e.printStackTrace();
}
```

Exemplul următor prezintă o modalitate corectă de a deschide un fișier pentru a fi citit. `FileNotFoundException` este o subclasă a lui `IOException`.

Dacă în clauza `catch(FileNotFoundException e)` apare o altă excepție, cum ar fi `EOFException`, atunci ea va fi prinsă de `catch(IOException e)`.

```
import java.io.*;

public class CitireDate {
    public static void main(String args[]) {
        try {
            RandomAccessFile f = new RandomAccessFile("numeFisier.txt", "r");
        }
        catch(FileNotFoundException e) {
            System.out.println("Fisierul nu exista");
            System.out.println(e.getMessage());
        }
        catch(IOException e) {
            System.out.println("Eroare IO");
            System.out.println(e.toString());
        }
    }
}
```

CLAUZA `finally`

Este posibil să se dorească execuția unei secvențe de cod, indiferent de apariția unei excepții (atât dacă apare, câte și dacă nu apare excepție). Clauza `finally` permite acest lucru și se folosește astfel:

```
try {
    ...
}
catch (Exceptie ex) {
    ...
}
finally {
    ...
}
```

Codul din blocul `finally` este executat indiferent dacă apare o sau nu excepție în blocul `try` și este prinsă sau nu. Deci, chiar dacă apare o excepție și nu este prinsă, blocul `finally` tot se execută. Mai mult, se execută chiar și atunci când înainte este prezentă o instrucțiune `return`. Într-o singură situație nu va funcționa `finally` – atunci când JVM se oprește și se apelează `System.exit()` dintr-un bloc `try` sau `catch`. Codul din blocul `finally` poate arunca și el o excepție sau apela `System.exit()`.

```
try {
     deschidereFisier();
     citireFisier();
...
} catch (FILE_NOT_FOUND) {
     // gestioneaza exceptia
}  catch (INUFFICIENT_PERMISSIONS) {
     // gestioneaza exceptia
}  catch (DISK_ERROR) {
     // gestioneaza exceptia
}  finally {
     // inchide fisierul
}
```

În exemplul de mai sus, se poate observa că, indiferent de execuția programului, blocul `finally` mereu o să închidă fișierul. Putem să vedem `finally` și ca o modalitate de a curăța resursele.

- Blocurile `try` pot funcționa și fără `catch`, utilizând doar `finally`, dar nu pot fi de sine stătătoare fără `finally` sau `catch`. O astfel de utilizare ar rezulta într-o eroare de compilare.
- Nu se poate insera nicio instrucțiune de cod între `try-catch` sau `try-finally` sau `try-catch-finally`.
- O excepție poate fi aruncată de mai multe ori, în așa fel încât să fie manipulată de alte blocuri `catch` din program.

```
try {
    ...
}
catch (Exceptie ex) {
    ...
    throw ex;
}
```

Uneori este nevoie să aruncăm o nouă excepție alături de excepția inițială. Acest proces se numește *chained exceptions*.

```java
public class InlantuireExceptii {
    public static void main(String[] args) {
        try {
            metoda1();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    public static void metoda1() throws Exception {
        try {
            metoda2();
        }
        catch (Exception ex) {
            throw new Exception("Informatie din metoda1", ex);
        }
    }
    public static void metoda2() throws Exception {
        throw new Exception("Informatie din metoda2");
    }
}
```

Excepția aruncată de `metoda2()` este prinsă în blocul `catch` din `metoda1()` și îmbinată cu o nouă excepție definită și aruncată în `metoda1()`. Noua excepție este aruncată și prinsă apoi în blocul `catch` din metoda `main`.

CREAREA PROPRIILOR CLASE DE EXCEPȚII

Atunci când programăm, ideal este să creăm excepții specifice, care descriu motivul pentru care execuția programului nu s-a realizat cu succes. Crearea acestora se face prin extinderea clasei `Exception` sau a oricărei clase din ierarhia de clase de excepții. Acestea sunt excepții de tip ***checked*.**

`Exception` moștenește clasa `Throwable`, din care preia metodele:

- `getMessage()` - returnează mesajul obiectului;
- `toString()`– returnează o concatenare a numelui clasei și a mesajului returnat de `getMessage();`
- `printStackTrace()` - afișează apelurile pe stivă ale metodelor.

Clasa `Exception` are 4 constructori [3], dintre care cei mai des utilizați sunt constructorul fără parametri și cel cu parametru de tip `String`.

Fie următoarele clase:

```java
class ExceptieA extends Exception {}
class ExceptieB extends ExceptieA {}
class ExceptieC extends ExceptieA {}
```

In aceste condiții, trebuie acordată atenție ordinii în care se vor defini blocurile `catch`. Acestea trebuie precizate de la clasa excepție cea mai particulară, până la cea mai generală (în sensul moștenirii). De exemplu, pentru a întrebuința excepțiile de mai sus, blocul `try-catch` ar trebui să arate ca mai jos:

```java
try {
    ...
} catch (ExceptieB e) {
    ...
} catch (ExceptieA e) {
    ...
} catch (Exception e) {
    ...
}
```

Excepțiile trebuie prinse de la cea mai specifică la cea mai generală. `ExceptieB` este subclasă a clasei `ExceptieA`, care la rândul ei este derivată din clasa de bază `Exception`.

APLICAȚII:

Clasa ContBancar

Implementați clasa `ContBancar` cu următoarele variabile membru:

- `private int numarCont;`
- `private int depozit;`

Clasa `ContBancar` va conține:

- Un constructor cu parametri
- Metoda `public void depune(int suma)`, ce realizează depunerea unei sume `(depozit = depozit + suma)`
- Metoda `public void retrage(int suma)`, ce retrage o suma din depozit. Această metodă va arunca excepția `ExceptieFonduriInsuficiente`, definită de voi, ce extinde clasa `Exception` și afișează mesajul *Fonduri Insuficiente*, în situația în care suma retrasă este mai mare decât suma existentă în depozit
- Metoda `public int numarCont()`, ce returnează numărul contului
- Metoda `public int depozit()`, ce returnează suma acumulată în depozit

Clasa Persoana

Implementați clasa `Persoana` cu următoarele variabile membru:

- `private static int nrPersoane;`
- `private ContBancar cont;`
- `private String nume, cnp;`

Clasa `Persoana` va conține

- Metoda `public void seteazaDate(String nume, String cnp, ContBancar cont)`, ce setează numele, CNP-ul și contul persoanei. Această metodă aruncă excepția: `ExceptieCnpInvalid` (definită de voi, ce extinde clasa `Exception` și afișează mesajul *CNP Invalid* atunci când primul caracter din CNP este diferit de 1 sau 2 și în situația când lungimea CNP-ului este diferită de 13).
- Constructorul cu parametri `public Persoana(String nume, String cnp, ContBancar cont)`, ce folosește un apel al metodei `seteazaDate` și incrementează variabila statică `nrPersoane`.
- Metoda public `ContBancar cont()`, ce returnează contul
- Metoda public `void afiseazaInformatii()`, ce afișează informațiile despre persoană – nume, CNP și cont

Testarea programului

În metoda `main` (definită într-una dintre cele două clase sau într-o altă):

- Creați un tablou cu n obiecte de tip `Persoana` și citiți de la tastatură datele despre ele. Variabila `n` se va genera aleatoriu cu o valoare întreagă cuprinsă între 2 și 9. Rulați programul folosind diverse CNP-uri valide (puteți folosi un generator ca acesta [4]), dar și CNP-uri greșite (variind câmpurile unui CNP valid).
- Afișați informațiile despre aceste persoane, prin apelul metodei `afiseazaInformatii()`
- Retrageți din contul fiecărei persoane suma de 1000 RON, prin apelul metodei `retrage()`
- Afișați din nou datele despre aceste persoane, după retragere, prin apelul metodei `afiseazaInformatii()`

Referințe:

1. Oana Balan, Mihai Dascalu. **Programarea orientata pe obiecte in Java**. Editura Politehnica Press, 2020
2. Bert Bates, Kathy Sierra. **Head First Java: Your Brain on Java - A Learner's Guide 1st Edition**. O'Reilly Media. 2003
3. Herbert Schildt. **Java: A Beginner's Guide**. McGraw Hill. 2018
4. Barry A. Burd. **Java For Dummies**. For Dummies. 2015
5. Joshua Bloch. **Effective Java**. ISBN-13978-0134685991. 2017
6. Harry (Author), Chris James (Editor). **Thinking in Java: Advanced Features (Core Series) Updated To Java 8**
7. Learn Java online. Disponibil la adresa: https://www.learnjavaonline.org/
8. Java Tutorial. Disponibil la adresa: https://www.w3schools.com/java/

   The Java Tutorials. Disponibil la adresa: https://docs.oracle.com/javase/tutorial/