Programare orientată pe obiecte

1. Erori şi excepţii în Java

Probleme în cursul execuţiei

- Un program întâlneşte adesea probleme în cursul execuţiei sale:
  - poate avea probleme la citirea datelor,
  - pot exista caractere nepermise în date sau
  - indexul unui tablou poate depăşi limitele acestuia.
- Excepţiile şi erorile Java permit programatorului să trateze astfel de probleme.
  - Putem scrie programe care îşi revin la întâlnirea erorilor şi îşi continuă execuţia.
  - *Programele nu trebuie să eşueze atunci când utilizatorul face o greşeală!*
- În special intrarea şi ieşirea sunt susceptibile la erori.
- Tratarea excepţiilor este esenţială pentru programarea I/E

Exemplu de apariţie a unei excepţii

- Programul:

```
import java.util.Scanner;
public class InputMismatchExceptionDemo {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter one integer:");
        int inputNumber = keyboard.nextInt();
        System.out.println("The square of  " + inputNumber + " is " +
  inputNumber * inputNumber);
    }
}
```

- Cu intrarea: `Enter one integer:h1`
- Dă rezultatul:

```
java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:819)
  at java.util.Scanner.next(Scanner.java:1431)
  at java.util.Scanner.nextInt(Scanner.java:2040)
  at java.util.Scanner.nextInt(Scanner.java:2000)
  at InputMismatchExceptionDemo.main(InputMismatchExceptionDemo.java:11)
```


Discuţie asupra exemplului

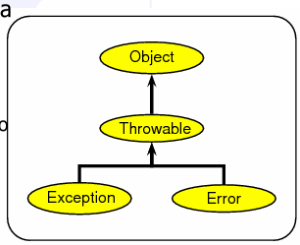
- Programul nu este greşit.
  - Problema este că **`nextInt`** nu poate converti şirul de caractere "**`h1`**" la un **`int`**.
  - În momentul în care **`nextInt`** a întâlnit problema, metoda a **aruncat** o excepţie de tipul **`InputMismatchException`**.
  - Sistemul de execuţie Java a interceptat (a "prins") excepţia, a oprit programul şi a tipărit mesajele de eroare

Excepţii şi erori

- O **excepţie:** o problemă care apare în cursul execuţiei unui program.
  - La apariţia unei excepţii, JVM creează un obiect de clasa **`Exception`** care conţine informaţii despre problema apărută.
  - Însuşi programul Java poate **intercepta** (**catch**) o excepţie. Apoi poate folosi obiectul de tipul excepţie pentru aşi reveni după problemă.
- Şi o **eroare** este o problemă care apare la rularea unui program.
- O eroare este reprezentată de un obiect de clasa **`Error`**.
  - Dar o eroare este prea severă pentru a fi tratată de un program. Programul trebuie să-şi *înceteze execuţia.*

Ierarhia Throwable ("aruncabil")

- Atât clasa **`Exception`** cât şi clasa **`Error`** descind din **`Throwable`**.
  - O metodă Java poate "arunca" un obiect de clasa **`Throwable`**.
  - D.e. Integer.parseInt("zzz") aruncă o excepţie atunci când încearcă să convertească "zzz" într-un întreg.
- Excepţii != Erori: se pot scrie programele astfel încât să-şi revină după excepţii, dar nu se pot scrie astfel încât să-şi revină după erori



Introducere în tratarea excepţiilor

- Software de bibliotecă Java (sau codul definit de către programator) furnizează un mecanism care semnalizează când se întâmplă *ceva neobişnuit*
  - Aceasta se numeşte *aruncarea unei excepţii*
- În alt loc din program, programatorul trebuie să scrie cod care *tratează* cazul excepţional
  - Aceasta se numeşte *tratarea excepţiei*

Mechanismul `try-throw-catch`

- Calea fundamentală pentru tratarea excepţiilor în Java constă din trio-ul ***`try-throw-catch`***
- Blocul `try` conţine codul pentru algoritmul implementat
  - Acest cod spune ce se face atunci când totul merge şnur
  - Se numeşte bloc `try` deoarece el "încearcă" să execute cazul în care totul merge aşa cum a fost planificat
  - De asemenea acest bloc poate conţine cod care aruncă o excepţie dacă se întâmplă ceva neobişnuit

```
try {
   CodCarePoateAruncaOExceptie
}
```


Mechanismul `try-throw-catch`

```
throw new
   NumeleClaseiExceptie(PosibilArgumente);
```

- La aruncarea unei excepţii, execuţia blocului `try` în care a fost aruncată excepţia se opreşte
  - Normal, controlul este transferat unei alte porţiuni de cod, blocul `catch` (blocul de interceptare)
- Valoarea aruncată este argumentul operatorului `throw`; ea este întotdeauna un obiect aparţinând unei clase excepţie
  - Execuţia unei instrucţiuni `throw` se numeşte *aruncare a unei excepţii*

Mechanismul `try-throw-catch`

- O instrucţiune `throw` seamănă cu un apel de metodă:

```
throw new NumeClasaExceptie(UnString);
```

  - În exemplul de mai sus, obiectul de clasa ***`NumeClasaExceptie`*** este creat folosind ca argument un şir de caractere
  - Acest obiect, care este argument pentru operatorul `throw`, este obiectul excepţie aruncat
- În loc să apeleze o metodă, instrucţiunea `throw` apelează un bloc `catch`

Mechanismul `try-throw-catch`

- La aruncarea unei excepţii se începe executarea blocului `catch`
  - Blocul `catch` are *un parametru*
  - Obiectul excepţie aruncat este transmis ca parametru al blocului `catch`
- Execuţia blocului `catch` se numeşte *interceptarea/"prinderea" excepţiei*, sau *tratarea excepţiei*
  - Ori de câte ori se aruncă o excepţie, ea trebuie până la urmă tratată (interceptată – "prinsă") de un bloc `catch`

Mechanismul `try-throw-catch`

```
catch(Exception e) {
   CodDeTratareAExceptiei
}
```

- Un bloc `catch` arată ca o definiţie de metodă care are un parametru de tipul clasei ***`Exception`***
  - Dar nu este, totuşi, o definiţie de metodă
- Un bloc `catch` este o porţiune de cod separată care se execută atunci când un program întâlneşte şi execută o instrucţiune `throw` în blocul `try` precedent
  - Un bloc `catch` este numit adesea *bloc de tratare a excepţiei*
  - El poate avea cel mult un parametru

Mechanismul `try-throw-catch`

`catch(Exception e) { . . . }`

- Identificatorul `e` din blocul `catch` de deasupra se numeşte parametru al blocului `catch`
- Parametrul blocului `catch` îndeplineşte două roluri:
  1. Specifică tipul de obiect excepţie aruncat pe care blocul `catch` îl poate intercepta (d.e., mai sus este un obiect de clasa `Exception`)
  2. Oferă un nume (pentru obiectul care este interceptat) care să fie folosit în blocul `catch`
     - Observaţie: adesea se foloseşte identificatorul `e` prin convenţie, dar se poate folosi orice identificator care nu este cuvânt cheie

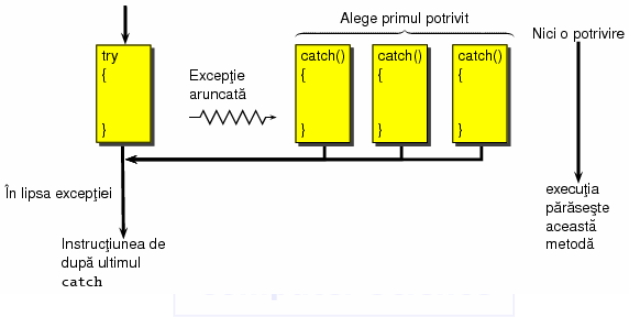
Mechanismul `try-throw-catch`

- La executarea unui bloc `try` se pot întâmpla două lucruri:
  1. Nu este aruncată nici o excepţie în blocul `try`
     - Codul din blocul `try` este executat până la sfârşitul blocului
     - Blocul `catch` este sărit
     - Execuţia continuă de la codul amplasat după blocul `catch`

Mechanismul `try-throw-catch`

2. Este aruncată o excepţie în blocul `try` şi interceptată în blocul `catch`
   - Restul codului din blocul `try` este sărit
   - Controlul se transferă la un bloc `catch` următor (în cazurile simple)
   - Obiectul aruncat este transmis ca parametru al blocului `catch`
   - Se execută codul din blocul `catch`
   - Se execută codul care urmează după blocul `catch` respectiv (dacă există)

Mechanismul `try-throw-catch`



Exemplu cu două excepţii

```
public class DublaGreseala {
    public static void main(String[] args) {
      int num = 5, denom = 0, result;
      int[] arr = {7, 21, 31};
      try
      {
          result = num / denom;
          result = arr[num];
      }
      catch (ArithmeticException ex)   {
          System.out.println("Eroare aritmetica");
      }
      catch (IndexOutOfBoundsException ex) {
          System.out.println("Eroare de indice");
      }
    }
}
```

Observaţie. Cea de a doua excepţie nu va fi aruncată niciodată. De ce?

Mechanismul `try-throw-catch`

- La aruncarea unei excepţii de către o instrucţiune blocul try{}, blocurile catch{} sunt examinate unul câte unul începând cu primul.
- *Un singur bloc* catch{} este ales.
- Dacă nici un bloc catch{} nu se potriveşte cu excepţia, atunci nu este ales nici unul, iar execuţia părăseşte metoda respectivă (exact ca în lipsa blocului catch{}.)
- Primul bloc catch{} care se potriveşte cu tipul de excepţie aruncată obţine controlul.
- Cele mai specifice tipuri de excepţie trebuie să apară la început, urmate de tipurile mai generale de excepţie.
- Instrucţiunile din blocul catch{} ales sunt executate secvenţial. După executarea ultimei instrucţiuni, controlul ajunge la prima instrucţiune care urmează după structura try/catch.
- Controlul *nu se întoarce* în blocul try.

Exemplu de intrare "prietenoasă"

```
import java.lang.* ;
import java.io.* ;

public class SquareUser
{
  public static void main ( String[] a ) throws
     IOException
  {
    BufferedReader stdin =
       new BufferedReader ( new
     InputStreamReader( System.in ) );
    String  inData = null;
    int     num = 0;
    boolean inputOK = false;
    while ( !inputOK )
    {
      System.out.print("Introduceti un
     intreg:");
      inData = stdin.readLine();
      try
      {
        num     = Integer.parseInt( inData
      );
        inputOK = true;
      }
      catch (NumberFormatException ex )
      {
        System.out.println("Ati introdus
      date invalide." );
        System.out.println("Va rog sa
      reincercati.\n" );
      }
    }
    System.out.println("Patratul lui " +
     inData + " este " + num*num );
  }
}
```


Clauza `finally`

- Excepţia provoacă terminarea metodei curente
- Pericol: se poate sări peste o porţiune de cod esenţială
- Exemplu:

```
reader = new FileReader(filename);
Scanner in = new Scanner(reader);
readData(in);
reader.close();
// s-ar putea sa nu ajunga aici
// niciodata
```

- Trebuie executat `reader.close()` chiar dacă apare o excepţie
- Folosim clauza `finally` pentru codul care trebuie executat "indiferent de ce se întâmplă" (necondiţionat)

Clauza `finally`

- Exemplu BlueJ (`ExceptFinallyEx`)

Clauza `finally`

- Executată la ieşirea din blocul `try`:
  - După ultima instrucţiune din blocul `try`
  - După ultima instrucţiune din blocul `catch`, dacă în acest bloc `try` a apărut o excepţie
  - La aruncarea unei excepţii în blocul `try`, excepţie care nu a fost interceptată
- Cay Horstmann recomandă: nu amestecaţi clauzele `catch` şi `finally` în acelaşi bloc `try`

Blocuri `catch` multiple şi clauza `finally`

- **Dacă există clauze `catch` asociate blocului `try`, atunci trebuie să punem clauza `finally` după toate clauzele `catch`. Exemplu:**

```
try {
    // Bloc de cod cu puncte de iesire multiple
}
catch (OneException e) {
    System.out.println("Am interceptat OneException!");
}
catch (OtherException e) {
    System.out.println(" Am interceptat OtherException!");
}
catch (AnotherException e) {
    System.out.println(" Am interceptat AnotherException!");
}
finally {
    // Bloc de cod executat intotdeauna la iesirea din blocul,
    // indiferent de cum s-a iesit din try.
    System.out.println("Finally este executat intotdeauna");
}
```


Clase excepţie

- Există mai multe clase excepţie pe lângă clasa `Exception`
  - Există mai multe clase excepţie în bibliotecile standard Java
  - Pot fi definite noi clase excepţie exact ca orice alte clase
- Toate clasele excepţie predefinite au următoarele proprietăţi:
  - Posedă un constructor cu un singur argument de tipul `String`
  - Clasa are o metoda accesoare, `getMessage()`, care poate recupera şirul dat ca argument constructorului la crearea obiectului excepţie
- Toate clasele definite de programator ar trebui să aibă aceleaşi proprietăţi

Clase excepţie din pachetele standard

- Există numeroase clase excepţie predefinite care sunt incluse în pachetele standard livrate cu Java. Spre exemplu:
  - `IOException`
  - `NoSuchMethodException`
  - `FileNotFoundException`
  - Multe clase excepţie trebuie importate pentru a le putea utiliza. Exemplu:
    ```
    import java.io.IOException;
    ```
- Clasa predefinită **`Exception`** este clasa rădăcină pentru toate excepţiile
  - Fiecare clasă excepţie este descendentă din clasa **`Exception`**
  - Deci clasa **`Exception`** poate fi folosită: direct sau pentru a defini o clasă derivată
  - Se află în pachetul **`java.lang`** şi, nu trebuie clauză **`import`**

Folosirea metodei `getMessage`

```
. . . // codul metodei
try
{
  . . .
  throw new
   Exception(StringArgument);
  . . .
}
catch(Exception e)
{
  String message =
   e.getMessage();
  System.out.println(message);
  System.exit(0);
}  . . .
```

- Fiecare excepţie are o variabilă instanţă de tipul **`String`** care conţine un mesaj
  - Acest şir identifică de obicei motivul apariţiei excepţiei
- **`StringArgument`** este folosit ca valoare pentru variabila instanţă de tip şir a excepţiei ***`e`***
  - De aceea, apelul de metodă **`e.getMessage()`** returnează acest şir

Definirea claselor excepţie

- O instrucţiune **`throw`** poate arunca un obiect excepţie de oricare clasă excepţie
- Clasele excepţie pot fi şi *definite de către programator*
  - Acestea pot fi croite astfel încât să conţină exact tipurile de informaţie necesare în blocul **`catch`**
  - Putem defini un tip de excepţie diferit pentru a identifica fiecare situaţie excepţională

Definirea claselor excepţie

- Fiecare clasă excepţie care urmează să fie definită trebuie să fie o clasă derivată dintr-o clasă excepţie deja definită
  - derivată din oricare clasă excepţie definită în bibliotecile standard Java sau definită de către programator
- Constructorii sunt membrii cei mai importanţi în definirea unei clase excepţie
  - Constructorii trebuie să se comporte corespunzător în raport cu variabilele şi metodele moştenite din clasa de bază
  - Adesea, nu există alţi membri cu excepţia celor moşteniţi din clasa de bază
- Clasa care urmează nu efectuează decât aceste sarcini fundamentale

O clasă excepţie definită de către programator

```
public class DivisionByZeroException extends Exception
{
   public DivisionByZeroException()
   {
       super("Division by zero.");
   }
   public DivisionByZeroException(String message)
   {
       super(message);
   }
}
```

Se poate face mai mult într-un constructor de excepţie, dar aceasta este o formă uzuală.

super **invocă constructorul clasei de bază `Exception`**

Caracteristicile obiectului Exception

- Cele mai importante două lucruri referitoare la un obiect excepţie sunt tipul său (adică, clasa excepţie) şi mesajul pe care îl poartă
  - Mesajul este transmis împreună cu obiectul excepţie ca variabilă instanţă
  - Acest mesaj poate fi recuperat cu metoda accesoare **`getMessage`**, astfel că blocul **`catch`** poate folosi mesajul

Indicaţii pentru clasele excepţie definite de programator

- Clasele excepţie pot fi definite de către programator, dar fiecare asemenea clasă trebuie să fie derivată dintr-o clasă excepţie existentă deja
- Clasa **`Exception`** poate fi folosită pe post de clasă de bază, cu excepţia cazului în care o altă clasă excepţie este mai potrivită
- Trebuie definiţi cel puţin doi constructori, iar uneori mai mulţi
- Excepţia trebuie să ţină seama că metoda **`getMessage()`** este moştenită

Să păstreze `getMessage`

- Pentru toate clasele excepţie predefinite, **`getMessage`** returnează şirul de caractere transmis ca argument constructorului său
  - Sau să returneze un şir implicit dacă nu s-a transmis nici un argument constructorului
- Acest comportament trebuie păstrat în toate clasele excepţie definite de către programator
  - Trebuie inclus un *constructor* care are *un parametru şir* de caractere şi al cărui corp începe cu un apel la **`super.`** Apelul la **`super`** trebuie să folosească parametrul ca argument al său
  - Trebuie inclus şi un *constructor fără argumente* al cărui corp începe cu un apel la **`super`**. Acest apel la **`super`** trebuie să folosească *şirul implicit* ca argument

Blocuri `catch` multiple

- Un bloc **`try`** poate arunca potenţial orice număr de valori excepţie, iar acestea pot fi de tipuri diferite
  - În oricare execuţie a unui bloc **`try`**, poate fi aruncată cel mult o excepţie (de vreme ce instrucţiunea **`throw`** termină execuţia blocului **`try`**)
  - La execuţii diferite ale blocului **`try`** pot fi aruncate valori diferite
- Fiecare bloc **`catch`** poate intercepta valorile de tipul de clasă excepţie date în antetul blocului **`catch`**
- Se pot intercepta tipuri diferite de excepţii punând mai multe blocuri **`catch`** după un bloc **`try`**
  - Se pot pune oricâte blocuri **`catch`**, dar în ordinea corectă

Capcană: Interceptaţi mai întâi cea mai specifică excepţie

- La interceptarea de excepţii multiple, ordinea blocurilor **`catch`** este importantă
  - **La aruncarea unei excepţii într-un bloc `try`, blocurile `catch` sunt examinate în ordinea apariţiei**
  - **Este executat primul bloc care se potriveşte cu tipul de excepţie aruncat**

```
catch (Exception e)
{ . . . }
catch (NegativeNumberException e)
{ . . . }
```

- Deoarece **`NegativeNumberException`** este un tip de **`Exception`**, toate **`NegativeNumberExceptions`** vor fi interceptate de către primul bloc **`catch`** înainte de a ajunge vreodată la cel de-al doilea
  - **Blocul catch pentru `NegativeNumberException` nu va fi folosit!**
- Pentru ordine corectă, inversaţi cele două blocuri

Aruncarea unei excepţii într-o metodă

- Uneori are sens să se arunce o excepţie într-o metodă fără a o intercepta în metoda respectivă
  - Unele programe care folosesc o anume metodă ar trebui să se termine pur şi simplu la aruncarea unei excepţii, iar altele nu
  - În atare cazuri, programul care foloseşte invocarea metodei ar trebui să o includă într-un bloc **`try`** şi să intercepteze excepţia într-un bloc **`catch`** care urmează
- În acest caz, metoda în sine nu va include blocuri **`try`** şi **`catch`**
  - Totuşi, trebuie să conţină o *clauză* ***`throws`***

Declararea excepţiilor în clauza `throws`

- Dacă o metodă poate arunca o excepţie, dar nu o interceptează, atunci ea trebuie să furnizeze un avertisment
  - Acest avertisment se numeşte *clauză* ***`throws`***
  - Procesul de includere a unei clase excepţie într-o clauză throws se numeşte *declararea excepţiei*

```
throws OExceptie  //clauza throws
```

  - Următorul cod declară că invocarea lui **`oMetoda`** poate cauza aruncarea lui **`OExceptie`**

```
public void oMetoda() throws OExceptie
```

- **`main( )`** este o metodă care poate avea şi ea specificarea unei excepţii:

```
public static void main(String[] args) throws Exception
```

Declararea excepţiilor în clauza `throws`

- Dacă o metodă poate arunca mai mult de un fel de excepţie, atunci tipurile se separă prin virgule

```
public void oMetoda() throws
   OExceptie, AltaExceptie
```

- Dacă o metodă aruncă o excepţie şi nu o interceptează, atunci apelul metodei se termină imediat

Regula "prinde sau declară"

- Cele mai obişnuite excepţii care ar putea fi aruncate într-o metodă trebuie tratate în unul dintre următoarele două moduri:
  1. Codul care poate arunca o excepţie este pus într-un bloc `try`, iar excepţia care poate apărea este interceptată într-un bloc `catch` din aceeaşi metodă
  2. Excepţia posibilă poate fi declarată la începutul definiţiei metodei punând numele clasei excepţie într-o clauză `throws`

Regula "prinde sau declară"

- Prima dintre tehnici tratează o excepţie într-un bloc `catch`
- Cea de a doua tehnică este o modalitate de a deplasa răspunderea pentru tratarea excepţiei la metoda care a invocat-o pe cea care a aruncat excepţia
- Metoda apelantă trebuie să trateze excepţia, cu excepţia cazului în care foloseşte aceeaşi tehnică de "pasare"
- Într-un sfârşit, fiecare excepţie ar trebui interceptată de un bloc `catch` din vreo metodă care nu numai declară într-o clauză `throws` ci şi interceptează clasa de excepţie respectivă

Regula "prinde sau declară"

- În oricare metodă, ambele tehnici pot fi amestecate
  - Unele excepţii pot fi interceptate, iar altele declarate în clauza `throws`
- Cu toate acestea, tehnicile menţionate trebuie folosite consistent pentru o excepţie dată
  - Dacă o excepţie nu este declarată, atunci ea trebuie tratată în metodă
  - Dacă este declarată excepţia, atunci responsabilitatea pentru tratarea ei este pasată unei alte metode care o apelează
  - Observaţi că dacă definiţia unei metode include invocarea unei a doua metode, iar cea de a doua poate arunca o excepţie şi nu o interceptează, atunci prima metodă trebuie să o declare sau să o intercepteze

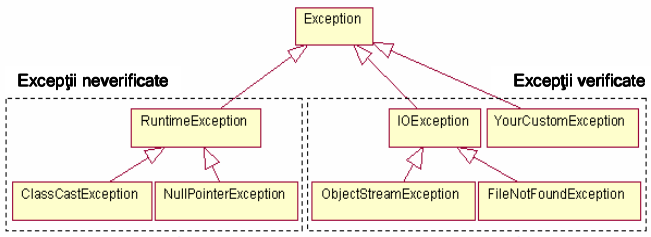
Excepţii verificate şi neverificate

- Excepţiile care sunt supuse regulii "prinde sau declară" sunt numite excepţii *verificate*
  - Compilatorul verifică pentru a vedea dacă excepţiile sunt luate în considerare fie într-un bloc `catch`, fie într-o clauză `throws`
  - Clasele `Throwable`, `Exception`, precum şi toţi descendenţii clasei `Exception` constituie excepţii verificate
- Toate celelalte excepţii sunt *neverificate*
- Clasa `Error` şi toate clasele care descind din ea sunt numite *clase eroare*
  - Clasele eroare *nu sunt* supuse regulii "prinde sau declară"

Excepţii de la regula "prinde sau declară"

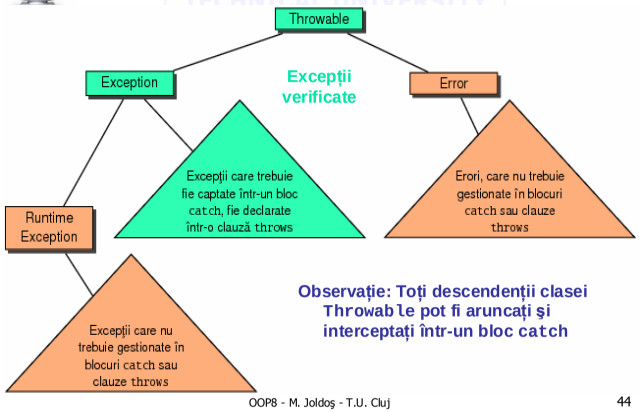
- Excepţiile *verificate* trebuie să respecte regula "prinde sau declară"
  - Programele în care pot fi aruncate aceste excepţii nu se vor compila până când excepţiile respective nu sunt tratate corespunzător
- Excepţiile *neverificate* nu sunt supuse regulii "prinde sau declară"
  - Programele în care apar astfel de excepţii trebuie pur şi simplu corectate întrucât au erori de alt fel, dacă compilatorul semnalează erori

Excepţii verificate şi neverificate



Notă. Aici este o mică parte a ierarhiei.

Ierarhia obiectelor Throwable (aruncabile)



Observaţie: Toţi descendenţii clasei Throwable pot fi aruncaţi şi interceptaţi într-un bloc catch

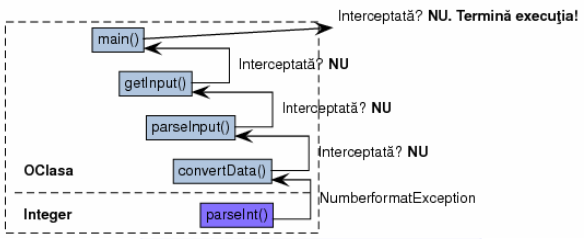
Clauza `throws` în clase derivate

- La suprascrierea unei metode într-o clasa derivată, aceasta trebuie să aibă aceleaşi clase excepţie precum cele listate în clauza **`throws`** din clasa de bază
  - sau un subset al acestora
- O clasă derivată *nu poate adăuga* excepţii la clauza **`throws`**
  - dar poate şterge câteva

Ce se întâmplă dacă o excepţie nu este interceptată?

- Dacă fiecare metodă până la, şi inclusiv, metoda **`main`** include o clauză **`throws`**, excepţia respectivă poate fi aruncată, dar poate să nu fie interceptată niciodată
  - Într-un program GUI (adică un program cu o interfaţa cu ferestre, grafică) nu se întâmplă nimic – atâta doar că utilizatorul poate fi lăsat într-o situaţie ne-explicată, iar programul poate să nu mai fie sigur
  - În programe non-GUI, aceasta face ca programul să se termine cu un mesaj de eroare care dă numele clasei excepţie
- Fiecare program bine scris trebuie în cele din urmă să intercepteze fiecare excepţie printr-un bloc **`catch`** în una dintre metodele sale

Propagarea excepţiei

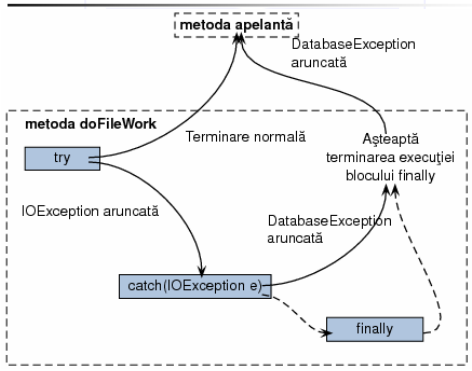


Un alt exemplu

```
public void doFileWork(String
   filename) throws
   DatabaseException{
 FileOutputStream fos = null;
 ObjectOutputStream oos = null;
 try{
   fos = new
    FileOutputStream(filename);
   oos = new
    ObjectOutputStream(fos);
   oos.writeObject(obj);
 }
 catch(IOException e){
   throw new DatabaseException(
    "Problem while working with
   "+filename+": "
    +e.getMessage());
 }
```

```
 finally{
   try{
    if(oos!=null){
     oos.close();
    }
    if(fos!=null){
     fos.close();
    }
   }
   catch(IOException e){
    throw new DatabaseException(
     "Problem while working with
  "+filename+": "
     +e.getMessage());
   }
  }
 }
```

Explicaţii pentru exemplu



Când să folosim excepţiile

- Excepţiile trebuie rezervate pentru situaţiile în care o metodă întâlneşte *un caz neobişnuit sau neaşteptat, care nu poate fi tratat cu uşurinţă în vreun alt mod*
- Atunci când trebuie folosită tratarea excepţiilor, folosiţi aceste recomandări:
  - Includeţi instrucţiuni **`throw`** şi precizaţi clasele excepţie într-o clauză **`throws`** din definiţia metodei respective
  - Plasaţi blocurile **`try`** şi **`catch`** într-o metodă diferită

Când să folosim excepţiile

- Iată un exemplu de metodă din care este aruncată o excepţie:

```
public void oMetoda()
    throws OExceptie {
  . . .
  throw new

  OExceptie(UnArgument);
  . . .
}
```

- Atunci când **`oMetoda`** este folosită de **`altaMetoda`**, **`altaMetoda`** trebuie să trateze excepţia:

```
public void altaMetoda()
{
  try {
    oMetoda();
    . . .
  }
  catch (OExceptie e) {
CodPentruTratareaExceptiei
  }
  . . .
}
```


Ghid pentru excepţii

- Dacă metoda întâlneşte o condiţie anormală pe care nu o poate trata, atunci trebuie să arunce o excepţie.
- Evitaţi folosirea excepţiilor pentru a indica situaţii care pot fi aşteptate ca parte a funcţionării normale a metodei.
- Dacă metoda descoperă că clientul şi-a încălcat obligaţiile contractuale (spre exemplu, prin transmiterea de date de intrare neconforme specificaţiei), atunci aruncaţi o excepţie neverificată.

Ghid pentru excepţii

- Dacă metoda nu-şi poate îndeplini contractul, atunci aruncaţi fie o excepţie verificată, fie una neverificată.
- Dacă aruncaţi o excepţie pentru o situaţie anormală despre care consideraţi că programatorii trebuie să decidă în mod conştient cum să o trateze, atunci aruncaţi o excepţie verificată.
- Definiţi sau alegeţi o clasă excepţie care există deja pentru fiecare fel de condiţie anormală care poate face ca metoda Dvs. să arunce o excepţie.

Re-aruncarea excepţiilor

- După interceptarea unei excepţii, ea poate fi re-aruncată dacă e cazul.
- La re-aruncarea unei excepţii putem alege locaţia din care se va vedea ca aruncat obiectul în trasarea stivei de execuţie.
  - Putem face ca excepţia re-aruncată să pară a fi aruncată din locul excepţiei originale sau
  - din locul re-aruncării.
- Pentru a re-arunca o excepţie cu indicarea locaţiei originale, pur şi simplu o aruncăm din nou:

```
try {
    cap(0);
} catch(ArithmeticException e) {
    throw e;
}
```

Re-aruncarea excepţiilor

- Pentru a avea locaţia reală din care a fost re-aruncată apelăm metoda **`fillInStackTrace()`** a excepţiei.
  - Metoda setează informaţia din trasarea stivei pe baza contextului de execuţie curent. Exemplu:

```
try {
    cap(0);
}
catch(ArithmeticException e) {
    throw (ArithmeticException)e.fillInStackTrace();
}
```

- Apelăm **`fillInStackTrace()`** pe linia cu instrucţiunea **`throw`** – astfel numărul de linie din trasare este la fel cu cel unde apare instrucţiunea throw.
  - Metoda **`fillInStackTrace()`** returnează o referinţă la clasa **`Throwable`**, aşa că e nevoie de o conversie de tip la tipul real de excepţie.

- Exemplu BlueJ (`DataSetReader`)

Rezumat

- Excepţii şi erori
- Excepţii verificate şi neverificate
- Instrucţiunile try şi catch
- Clauza finally
- Regula "prinde sau declară"
- Clauza throws
- Instrucţiunea throw