

Tratarea excepțiilor

1. Scopul lucrării

Obiectivele de învățare ale acestei sesiuni de laborator sunt:

- Înțelegerea noțiunii de excepție și învățarea modului corect de folosire a excepțiilor
- Acumularea experienței de programare a excepțiilor existente în bibliotecile Java și a celor definite de către programator

2. Tratarea excepțiilor

Tratarea excepțiilor constituie un mecanism care permite programelor Java să gestioneze diferitele situații excepționale, cum sunt violările semantice ale limbajului și erorile definite în programe într-un mod robust. La apariția unei situații excepționale se aruncă o excepție. Dacă mașina virtuală Java sau mediul de execuție detectează o violare semantică, mașina virtuală sau mediul de execuție vor arunca implicit o excepție. După aruncarea excepției controlul va fi transferat din punctul curent la o clauză **catch** corespunzătoare blocului **try** în care a apărut excepția. Clauza **catch** se numește tratarea excepției deoarece tratează excepția prin executarea de acțiuni care sunt necesare recuperării din situația respectivă.

O instrucțiune **try** conține un bloc de cod de executat. Plasarea unui bloc într-o instrucțiune **try** indică faptul că orice excepții sau alte ieșiri anormale din bloc urmează să fie tratate corespunzător. O instrucțiune **try** poate avea orice număr de clauze **catch** opționale care slujesc ca blocuri de tratare a excepțiilor pentru blocul **try** respectiv. O instrucțiune **try** poate avea și o clauză **finally**. Blocul **finally** este executat întotdeauna înainte de părăsirea instrucțiunii **try**; el "face curățenie" după blocul **try**. Remarcați că o instrucțiune **try** trebuie fie să aibă o clauză **catch** fie o clauză **finally**, fie ambele. Iată un exemplu de instrucțiune **try** care are o clauză **catch** și una **finally**:

```
try
{
    out.write(b);
}
catch (IOException e)
{
    System.out.println("Output Error");
}
finally
{
    out.close();
}
```

Dacă `out.write()` aruncă o `IOException`, excepția este interceptată de clauza **catch**. Indiferent dacă `out.write()` revine normal sau aruncă o excepție, blocul **finally** este executat, ceea ce asigură că `out.close()` este apelată întotdeauna.

Instrucțiunea **try** execută blocul care urmează după cuvântul cheie **try**. Dacă este aruncată o excepție din blocul **try** și instrucțiunea **try** are clauze **catch**, atunci respectivele clauze vor fi cercetate, în ordine, pentru a găsi una care poate trata excepția. Dacă există o clauză **catch** care poate trata excepția, atunci se execută respectivul bloc **catch**. Dar dacă instrucțiunea **try** nu are nici o clauză **catch** care poate trata excepția (sau nu are deloc clauze **catch**), excepția se propagă mai departe în blocurile exterioare celui în care a apărut în metoda curentă. Dacă metoda curentă nu conține o instrucțiune **try** cu un bloc de tratare corespunzător, excepția se propagă în sus spre metoda apelantă. Dacă această metodă nu conține o instrucțiune **try** cu un bloc de tratare corespunzător, excepția se propagă iarăși în sus. În sfârșit, dacă nu se găsește o instrucțiune **try** cu bloc de tratare corespunzător, programul în curs de execuție se termină.

Clauza **catch** se declară cu un parametru care specifică ce tip de excepție poate trata. Parametrul dintr-o clauză **catch** trebuie să fie de tipul **Throwable** sau al uneia dintre subclasele sale. La apariția unei excepții, clauzele **catch** sunt cercetate pe rând pentru a o găsi pe prima care are parametrul de același fel ca tipul excepției aruncate sau este o superclasă a excepției aruncate. La executarea blocului **catch** corespunzător, obiectul excepție actual este transmis ca argument al blocului **catch**. Codul din blocul **catch** trebuie să facă ceea ce este necesar pentru a trata situația excepțională.

Clauza **finally** a unei instrucțiuni **try** este executată întotdeauna, indiferent de modul în care s-a părăsit instrucțiunea **try**. De aceea clauza respectivă este un loc potrivit pentru a efectua operații de "curățare", cum sunt: *închiderea fișierelor, eliberarea resurselor și închiderea conexiunilor de rețea*.

3. Declararea excepțiilor

Dacă se așteaptă ca o metodă să arunce excepții, atunci metoda respectivă trebuie să declare acest lucru în clauza **throws**. Dacă implementarea unei metode conține o instrucțiune **throw**, atunci este posibil să fie aruncată o excepție din metodă. În plus, dacă metoda apelează o alta declarată cu clauză **throws**, există posibilitatea să fie aruncată o excepție din metoda respectivă. Dacă excepția nu este interceptată în metoda respectivă într-un **try-catch**, atunci ea va fi aruncată afară spre apelant. Orice excepție care poate fi aruncată în afara unei metode în acest mod, trebuie să fie listată într-o clauză **throws** în declarația metodei. Clasele listate într-o clauză **throws** trebuie să fie **Throwable** sau o subclasă a sa; clasa **Throwable** este superclasa tuturor obiectelor care pot fi aruncate în Java.

Există anumite tipuri de **Throwable** care nu trebuie listate în clauze **throws**. Mai specific, dacă excepția este o instanță a lui **Error**, **RuntimeException**, sau o subclasă a acestora, atunci nu trebuie listată în clauza **throws**. Subclasele clasei **Error** corespund unor situații care nu pot fi prevăzute ușor, cum este epuizarea memoriei disponibile. Subclasei lui **RuntimeException** corespund mai multor probleme uzuale care apar la execuție, cum sunt conversiile/forțările (casts) de tip și problemele legate de indecșii tablourilor. Motivul tratării speciale a acestora este că ele pot fi aruncate dintr-un număr atât de mare de locuri încât aproape fiecare metodă ar trebui să le declare. Considerați următorul exemplu:

```
import java.io.IOException;
class ThrowsExample
{
    char[] a;
    int position;
    ...
    // Metoda care arunca explicit o excepție
    int read() throws IOException
    {
        if (position >= a.length)
            throw new IOException();
        return a[position++];
    }

    // Metoda care implicit arunca o excepție
    String readUpTo(char terminator) throws IOException
    {
        StringBuffer s = new StringBuffer();
        while (true)
        {
            int c = read(); // Poate arunca o IOException
            if (c == -1 || c == terminator)
            {
                return s.toString();
            }
            s.append((char)c);
        }
        return s.toString();
    }
}
```

```

// Metoda care intercepțează intern excepția aparută
int getLength()
{
    String s;
    try
    {
        s = readUpTo(':');
    }
    catch (IOException e)
    {
        return 0;
    }
    return s.length();
}

// Metoda care poate arunca o RuntimeException
int getAvgLength()
{
    int count = 0;
    int total = 0;
    int len;
    while (true)
    {
        len = getLength();
        if (len == 0)
            break;
        count++;
        total += len;
    }
    return total/count; // Poate arunca o ArithmeticException
}
}

```

Metoda `read()` poate arunca o **IOException**, așa că declară acest lucru în clauza sa **throws**. Fără o clauză **throws**, compilatorul ar avertiza că metoda trebuie fie să declare **IOException** în clauza sa **throws** fie să o intercepțeze. Deși metoda `readUpTo()` nu aruncă explicit nici o excepție, ea apelează metoda `read()` care aruncă o **IOException**, așa că declară aceasta în clauza sa **throws**. Fie că excepția este aruncată implicit sau explicit, cerința ca să fie declarată sau interceptată există. Metoda `getLength()` intercepțează **IOException** aruncată de `readUpTo()`, așa că nu trebuie să o declare. Ultima metodă, `getAvgLength()`, poate arunca o **ArithmeticException** dacă variabila `count` are valoarea zero. Deoarece **ArithmeticException** este o subclasă a lui **RuntimeException**, faptul că ea poate fi aruncată afară din `getAvgLength()` nu trebuie declarat în clauza **throws**.

4. Generarea (aruncarea) excepțiilor

Un program Java poate folosi mecanismul de tratare a excepțiilor pentru a trata erori de program anume de o manieră curată. Pur și simplu programul folosește instrucțiunea **throw** pentru a semnala excepția. Instrucțiunea **throw** trebuie urmată de un obiect de tipul **Throwable** sau al unei subclase a sa. Pentru excepțiile definite prin program, dorim de obicei ca obiectul excepție să fie o instanță a unei subclase a clasei **Exception**. În majoritatea cazurilor are sens să definim o nouă subclasă a lui **Exception**, specifică programului nostru.

Considerați exemplul care urmează:

```

class WrongDayException extends Exception
{
    public WrongDayException () {}
    public WrongDayException(String msg)
    {
        super(msg);
    }
}
}

```

```

public class ThrowExample
{
    void doIt() throws WrongDayException
    {
        int dayOfWeek = (new java.util.Date()).getDay();
        if (dayOfWeek != 2 && dayOfWeek != 4)
            throw new WrongDayException("Tue. or Thur.");
        System.out.println("Did it");
    }
    public static void main (String [] argv)
    {
        try
        {
            (new ThrowExample()).doIt();
        }
        catch (WrongDayException e)
        {
            System.out.println("Sorry, can do it only on " + e.getMessage());
        }
    }
}

```

Codul din acest exemplu definește o clasă numită `WrongDayException` pentru a reprezenta tipul de excepție specific aruncat în exemplu. Clasa **Throwable** și majoritatea subclaselor sale au cel puțin doi constructori. Un constructor ia ca argument un **String** folosit ca mesaj textual care explică excepția, iar celălalt nu ia nici un argument. De aceea clasa `WrongDayException` definește doi constructori.

În clasa `ThrowExample`, dacă ziua curentă nu este nici Tuesday (marți) nici Thursday (joi), metoda `doIt()` aruncă o `WrongDayException`. Remarcați că obiectul `WrongDayException` este creat la momentul aruncării sale. Este o practică des întâlnită să se ofere informație despre excepție la aruncarea sa, așa că se folosește un argument șir de caractere în instrucțiunea de alocare a lui `WrongDayException`. Declarația metodei `doIt()` conține o clauză **throws** pentru a indica faptul că ea poate arunca o `WrongDayException`.

Metoda `main()` din `ThrowExample` include apelul la metoda `doIt()` într-o instrucțiune **try**, astfel că ea poate intercepta orice `WrongDayException` aruncat de către `doIt()`. Blocul **catch** afișează un mesaj de eroare folosind metoda `getMessage()` din obiectul excepție. Această metodă regăsește șirul transmis constructorului la crearea obiectului excepție.

4.1. Afișarea trasărilor de stivă

La interceptarea unei excepții poate fi util să tipărim o trasare a stivei pentru a ne da seama de unde provine excepția. O trasare a stivei arată în modul următor:

```

java.lang.ArithmeticException: / by zero
    at t.cap(t.java:16)
    at t.doit(t.java:8)
    at t.main(t.java:3)

```

Putem tipări trasarea stivei apelând metoda **printStackTrace()** pe care toate obiectele **Throwable** o moștenesc din clasa **Throwable**. Spre exemplu:

```

int cap(x) {
    return 100/x;
}
try {
    cap(0);
}
catch(ArithmeticException e) {
    e.printStackTrace();
}

```

Putem tipări trasarea stivei oriunde în aplicații fără a arunca de fapt o excepție. Exemplu:

```
new Throwable().printStackTrace();
```

4.2. Re-aruncarea excepțiilor

După interceptarea unei excepții ea poate fi re-aruncată dacă este cazul. Decizia pe care trebuie să o luăm la re-aruncarea unei excepții este despre locația de unde trasarea stivei să spună ca a fost aruncat obiectul. Putem face ca excepția re-aruncată să pară a fi fost aruncată din locația excepției inițiale sau din locația re-aruncării curente.

Pentru locația inițială, tot ce trebuie făcut este să re-aruncăm excepția:

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

Pentru a arăta locația reală e nevoie să apelăm metoda **fillInStackTrace()** a excepției. Această metodă setează informația din excepție pe baza contextului de execuție curent. Iată un exemplu care folosește metoda **fillInStackTrace()**:

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

Este important să apelăm **fillInStackTrace()** pe aceeași linie cu instrucțiunea **throw**, pentru ca numărul precizat de trasarea stivei să corespundă cu linia în care apare instrucțiunea **throw**. Metoda **fillInStackTrace()** întoarce o referință la clasa **Throwable**, așa că este nevoie să forțăm tipul (cast) referinței la tipul real al excepției.

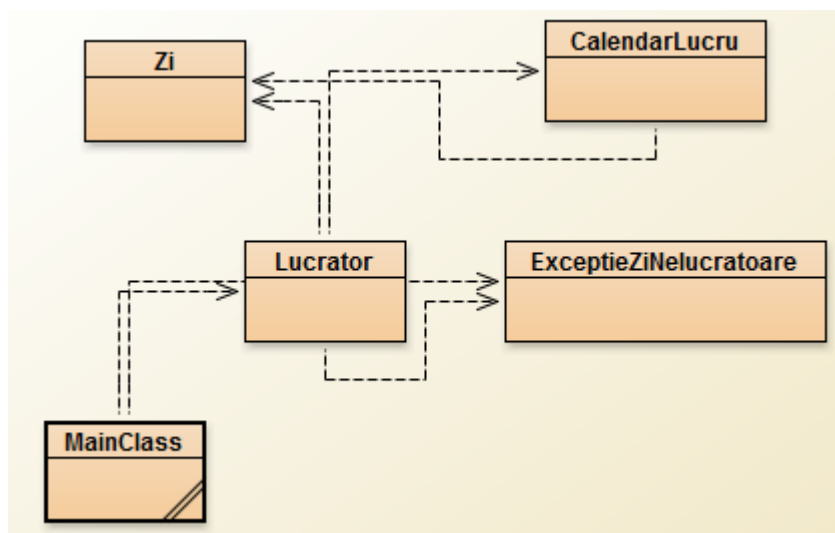
5. Sfaturi referitoare la excepții

Folosiți excepțiile pentru:

1. A trata problemele la nivelul corespunzător. (Evitați să interceptați excepțiile în afara cazului că știți ce să faceți cu ele.)
2. Rezolvați problema și apelați din nou metoda care a cauzat excepția.
3. Corectați lucrurile și continuați fără a re-încerca metoda.
4. Calculați un rezultat alternativ în locul celui care ar fi trebuit produs de către metodă.
5. Faceți tot ce puteți în contextul curent și apoi re-aruncați aceeași excepție spre un context mai sus.
6. Faceți tot ce puteți în contextul curent și apoi aruncați o excepție diferită spre un context mai sus.
7. Terminați execuția programului.
8. Simplificați. (Dacă schema voastră de excepții complică lucrurile, atunci e dificil și enervant de folosit.)
9. Faceți-vă bibliotecile și programele mai sigure. (Aceasta este o investiție pe termen scurt d.p.d.v. al depanării și una pe termen lung d.p.d.v. al robusteții aplicației.)

6. Mersul lucrării

6.1. Implementați o aplicație pentru a simula activitatea de lucru pentru angajați conform unui calendar pre-stabilit. Fiecare lucrător va fi identificat printr-un nume și va avea un calendar de lucru în care se vor specifica zilele lucrătoare și nelucrătoare. Atunci când un lucrător este pus la lucru într-o zi nelucrătoare, se va genera o excepție. Diagrama de clase este prezentată mai jos:



Detalii de implementare:

- Clasa **Zi** va implementa o zi oarecare din săptămână, fiind caracterizată de un nume (Luni, Marți, Miercuri,...) și de o proprietate în care se specifică dacă ziua este sau nu lucrătoare. Inițializarea, modificarea și accesul la cele două proprietăți se va face prin intermediul constructorilor, metodelor mutatoare și metodelor accesoare (de exemplu: `void setNume(String nume)`, `String getNume()`, `void setLucratoare()`, `void setNelucratoare()`, `boolean esteLucratoare()` etc...
- Clasa **CalendarLucru** va conține un tablou de zile. Implicit această clasă trebuie să specifice toate zilele săptămânii, iar zilele de sâmbătă și duminică să fie setate ca zile "nelucrătoare". Accesul și modificarea valorilor se va face, la fel, prin intermediul metodelor accesoare și mutatoare.
- Clasa **Lucrator**, va descrie un lucrător oarecare. Fiecare lucrător va fi identificat printr-un nume (numele lucrătorului) și un calendar de lucru – de tip **CalendarLucru**. Clasa **Lucrator** va implementa metoda **public void lucreaza(String zi)**, care va afișa un mesaj dacă ziua dată ca argument este una lucrătoare și va arunca o excepție în cazul când ziua corespunzătoare din calendar este "nelucrătoare".
- **ExceptieZiNelucratoare** este clasa în care se va implementa o excepție (extinde clasa **Exception**), cu un mesaj de excepție personalizat.
- În clasa **MainClass**, în metoda `... main(...)` se vor crea lucrători, calendare de lucru, se vor asigna la fiecare lucrător un calendar de lucru și se vor pune lucrătorii la muncă. Atunci când un lucrător trebuie să lucreze într-o zi nelucrătoare, se va genera o excepție de tip **ExceptieZiNelucratoare**. Aceste excepții vor trebui tratate corespunzător cu `try{...}-catch(...)`.
- Exemplu rulare, pentru un lucrător cu numele „Gigel”, pentru care zilele de luni – vineri sunt lucrătoare iar zilele de sâmbătă și duminică sunt zile nelucrătoare, pentru un apel de metode:

```

1.lucreaza("Luni");
1.lucreaza("Marte");
1.lucreaza("Duminica");
  
```

se va afișa:

```

Lucratorul Gigel lucreaza Luni
Marte nu este o zi a saptamanii
Exceptie: lucru in zi nelucratoare
  
```

- 6.2. Definiți o referință la un obiect și inițializați-o la **null**. Încercați să apelați o metodă folosind această referință. Apoi includeți codul într-o clauză **try-catch** pentru a intercepta excepția de tipul **NullPointerException**.
- 6.3. Creați o clasă cu o metodă **main()** care să genereze și să trateze cu clauza **try-catch** o excepție de tipul **ArrayIndexOutOfBoundsException**.
- 6.4. Creați o clasă cu o metodă **main()** care aruncă un obiect de clasa **Exception** în interiorul unui bloc **try**. Dați constructorului pentru **Exception** un argument **String**. Interceptați excepția într-o clauză **catch** și afișați argumentul **String**. Adăugați o clauză **finally** și tipăriți un mesaj pentru a dovedi că s-a ajuns acolo.
- 6.5. Creați propria voastră clasă excepție folosind cuvântul cheie **extends**. Scrieți un constructor pentru clasă care ia ca argument un **String** și îl stochează în obiect într-o referință la **String**. Scrieți o metodă care afișează șirul stocat. Creați o clauză **try-catch** pentru a exersa noua excepție.
- 6.6. Creați o clasă cu două metode, **f()** și **g()**. În **g()**, aruncați o excepție de un tip nou, definit de voi. În **f()**, apelați **g()**, interceptați excepția sa și, în clauza **catch**, aruncați o excepție diferită (de un al doilea tip, definit de voi). Testați-vă codul în **main()**.
- 6.7. Creați o ierarhie de excepții cu trei nivele. Apoi creați o clasă de bază **A** cu o metodă care aruncă o excepție la baza ierarhiei definite de voi. Moșteniți **B** din **A** și suprascrieți metoda astfel ca metoda să arunce o excepție la nivelul al doilea al ierarhiei. Repetați procedeul moștenind clasa **C** din **B**. În **main()**, creați **C** și convertiți-o (upcast) la **A**, apoi apelați metoda.