LABORATOR 8 -EXCEPȚII

Excepțiile în Java sunt evenimente care apar în timpul execuției programului care perturbă fluxul normal de instrucțiuni. Ele reprezintă un mecanism de tratare a erorilor care permite programatorilor să gestioneze și să răspundă unor condiții neașteptate sau problematice într-un mod organizat și controlat.

Structura:

- Throwable (clasa de bază)
    - Error (erori grave, de obicei nu le tratăm)
    - Exception (excepții pe care le putem trata)
        - RuntimeException (excepții nechecked - ArithmeticException, NullPointerException)
        - Alte excepții (checked exceptions - IOException, SQLException)

Analogie: Este ca un arbore genealogic. Throwable este strămoșul, iar toate excepțiile sunt descendenții săi cu caracteristici moștenite.

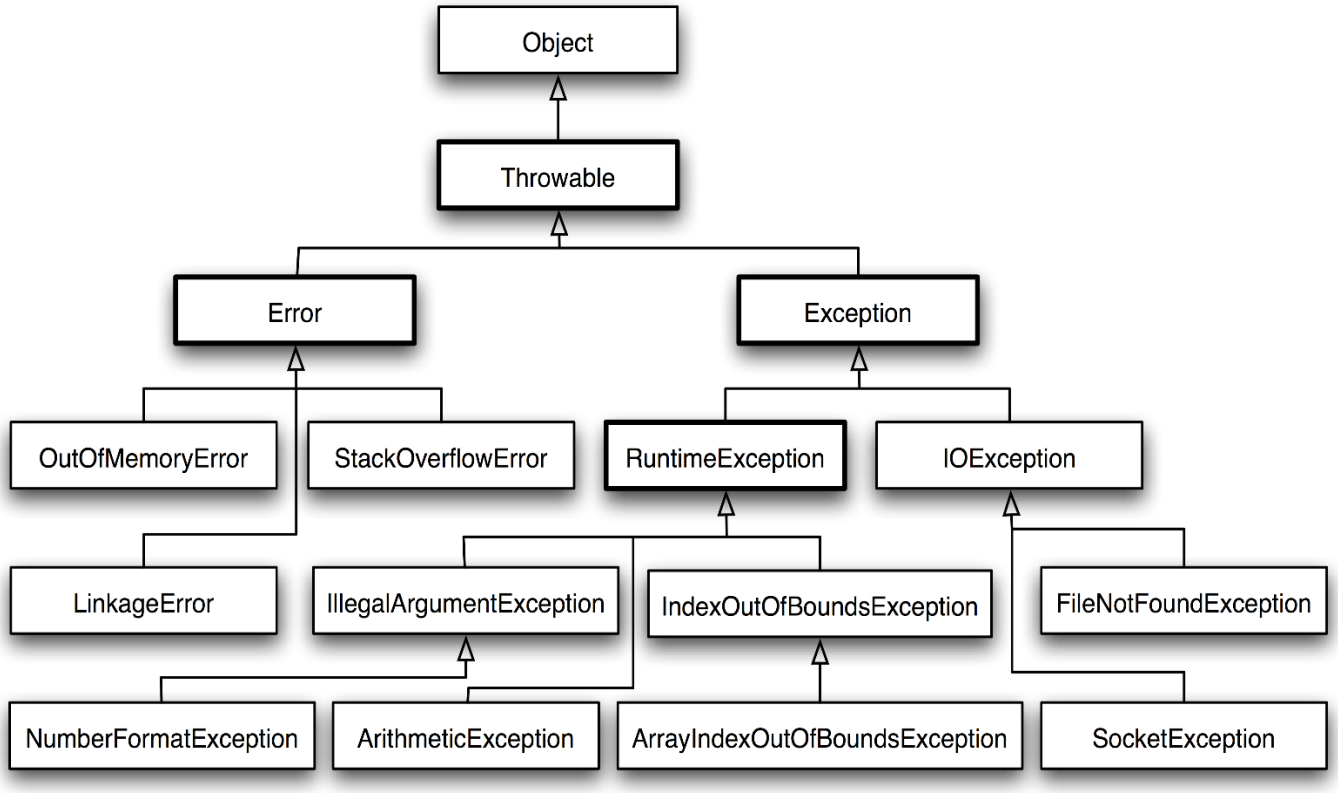

Figura 1. Ierahie

LABORATOR 8 -EXCEPȚII

Caracteristici principale ale excepțiilor:

1. Întreruperea fluxului normal de execuție: Atunci când apare o excepție, programul întrerupe secvența normală de instrucțiuni și transferă controlul către un bloc de gestionare a erorii.
2. Moștenire din clasa Throwable: În Java, toate excepțiile sunt obiecte care moștenesc direct sau indirect clasa Throwable, care este rădăcina ierarhiei de excepții.
3. Informații detaliate despre eroare: Fiecare excepție conține informații specifice despre natura și locația erorii, inclusiv traseul de execuție (stack trace).

1. Excepții Verificate (Checked Exceptions)

Excepțiile verificate sunt acele excepții care sunt verificate în momentul compilării. Programatorul TREBUIE să le trateze fie prin utilizarea blocului try-catch, fie prin declararea lor în semnătura metodei.

Exemple:

- IOException
- SQLException
- ClassNotFoundException

Caracteristici:

- Verificate de compilator
- Necesită tratare explicită
- Indică condiții externe care pot apărea în mod previzibil

```
class DemoChecked {

    // Metodă care aruncă o excepție verificată
    public static void testChecked() throws Exception {
        throw new Exception("Aceasta este o excepție verificată!");
    }

    public static void main(String[] args) {
        try {
            testChecked();   // compilatorul cere obligatoriu try-catch
        } catch (Exception e) {
            System.out.println("Excepție prinsă: " + e.getMessage());
        }
    }
}
```

Figura 2.Exceptie verificata

Acesta este un exemplu de checked exception deoarece metoda aruncă explicit o excepție de tip Exception, iar compilatorul obligă programatorul să o trateze folosind try-catch sau throws.

LABORATOR 8 -EXCEPȚII

2. Excepții Neverificate (Unchecked Exceptions)

Excepțiile neverificate, cunoscute și sub numele de Runtime Exceptions, nu sunt verificate în momentul compilării. Ele apar în mod tipic din erori de programare.

Exemple:

- NullPointerException
- ArrayIndexOutOfBoundsException
- ArithmeticException

Caracteristici:

- Nu sunt verificate la compilare
- Indică erori de programare
- Nu necesită tratare explicită, dar gestionarea lor poate îmbunătăți calitatea codului

```java
public static void main(String[] args) {
    int[] numbers = {1, 2, 3};
    System.out.println(numbers[5]);  // declanșează ArrayIndexOutOfBoundsException
}
```

Figura 3. Exceptie neverificata

Aceasta este o unchecked exception deoarece eroarea apare în timpul execuției când accesăm un index inexistent, iar Java nu obligă programatorul să trateze excepția cu try-catch.

3. Erori (Errors)

Erorile reprezintă situații grave care de obicei nu pot fi tratate de programator. Ele indică probleme sistemice sau de mediu.

Exemple:

- StackOverflowError
- OutOfMemoryError
- VirtualMachineError

Caracteristici:

- Indică probleme sistemice grave
- Nu se recomandă încercarea de recuperare
- Apar rar și semnalează condiții critice

4. **Excepțiile personalizate** sunt excepții create de programator pentru a descrie și semnala erori specifice aplicației, care nu sunt acoperite în mod adecvat de excepțiile standard din Java. Ele sunt folosite atunci când dorim să transmitem mesaje mai clare sau să impunem reguli de validare într-un mod controlat.

LABORATOR 8 -EXCEPȚII

O excepție personalizată este o clasă care extinde:

- Exception → dacă vrei ca excepția să fie checked (trebuie tratată obligatoriu)

- RuntimeException → dacă vrei ca excepția să fie unchecked (nu e obligatoriu să fie tratată)

```java
// Definirea unei excepții personalizate
class InvalidAgeException extends Exception {
    public InvalidAgeException(String mesaj) {
        super(mesaj);
    }
}
public class TestCustomException {
    public static void verificaVarsta(int varsta) throws InvalidAgeException {
        if (varsta < 0 || varsta > 150) {
            throw new InvalidAgeException("Vârsta introdusă nu este validă!");
        }
        System.out.println("Vârsta este corectă.");
    }

    public static void main(String[] args) {
        try {
            verificaVarsta(200);
        } catch (InvalidAgeException e) {
            System.out.println("Eroare: " + e.getMessage());
        }
    }
}
```

Figura 5. Exemplu de exceptie personalizata

O excepție personalizată este o clasă creată de programator care extinde Exception sau RuntimeException și permite semnalarea unor erori specifice aplicației cu mesaje clare și logica proprie.

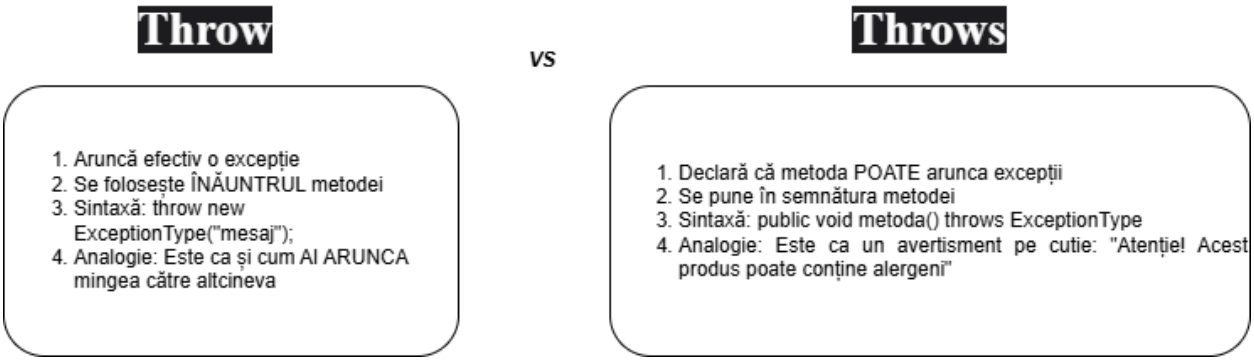

Figura 6. Throw vs Throws

LABORATOR 8 -EXCEPȚII

```java
public static void verificaVarsta(int varsta) throws Exception {
    if (varsta < 18) {
        throw new Exception("Persoana este minoră!"); // throw aruncă excepția
    }
    System.out.println("Acces permis.");
}

public static void main(String[] args) {
    try {
        verificaVarsta(15);
    } catch (Exception e) {
        System.out.println("Eroare: " + e.getMessage());
    }
}
```

Figura 7. Exemplu throw vs throws

throw aruncă efectiv o excepție în interiorul codului, în timp ce throws doar declară în semnătura metodei că metoda poate arunca acea excepție și obligă apelantul să o trateze.

1. Creează o clasă care validează vârsta unei persoane. Dacă vârsta este negativă sau mai mare de 150, aruncă o excepție personalizată InvalidAgeException.

Cerințe:

- Creează excepția personalizată InvalidAgeException
- Implementează metoda validateAge(int age)
- Tratează excepția în metoda main

LABORATOR 8 -EXCEPȚII

```java
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

class Person {
    private int age;

    public void setAge(int age) throws InvalidAgeException {
        if (age < 0 || age > 150) {
            throw new InvalidAgeException("Vârsta trebuie să fie între 0 și 150!");
        }
        this.age = age;
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person();
        try {
            person.setAge(-5);
        } catch (InvalidAgeException e) {
            System.out.println("Eroare: " + e.getMessage());
        }
    }
}
```

2. Creează un calculator care efectuează operații matematice. Tratează excepțiile pentru împărțirea la zero și pentru input-uri invalide.
   Cerințe:
   - Implementează metoda divide(int a, int b)
   - Tratează ArithmeticException pentru împărțirea la zero
   - Tratează NumberFormatException pentru conversii invalide
   - Folosește bloc finally pentru mesaj de finalizare

LABORATOR 8 -EXCEPȚII

```java
import java.util.Scanner;
class Calculator {
    public double divide(int a, int b) throws ArithmeticException {
        if (b == 0) {
            throw new ArithmeticException("Nu se poate împărți la zero!");
        }
        return (double) a / b;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Calculator calc = new Calculator();

        try {
            System.out.print("Introduceți primul număr: ");
            int num1 = Integer.parseInt(scanner.nextLine());

            System.out.print("Introduceți al doilea număr: ");
            int num2 = Integer.parseInt(scanner.nextLine());

            double result = calc.divide(num1, num2);
            System.out.println("Rezultat: " + result);

        } catch (NumberFormatException e) {
            System.out.println("Eroare: Introduceți doar numere întregi!");
        } catch (ArithmeticException e) {
            System.out.println("Eroare: " + e.getMessage());
        } finally {
            System.out.println("Operație finalizată.");
            scanner.close();
        }
    }
}
```

3. Creează o clasă BankAccount care gestionează un cont bancar. Implementează excepții personalizate pentru fonduri insuficiente și sume invalide.

Cerințe:

- Creează excepțiile InsufficientFundsException și InvalidAmountException
- Implementează metodele deposit(double amount) și withdraw(double amount)
- Tratează ambele tipuri de excepții
- Folosește excepții multiple în același bloc try-catch

LABORATOR 8 -EXCEPȚII

```java
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}
class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) throws InvalidAmountException {
        if (amount <= 0) {
            throw new InvalidAmountException("Suma trebuie să fie pozitivă!");
        }
        balance += amount;
        System.out.println("Depunere reușită. Sold: " + balance);
    }

    public void withdraw(double amount) throws InsufficientFundsException, InvalidAmountException {
        if (amount <= 0) {
            throw new InvalidAmountException("Suma trebuie să fie pozitivă!");
        }
        if (amount > balance) {
            throw new InsufficientFundsException("Fonduri insuficiente! Sold disponibil: " + balance);
        }
        balance -= amount;
        System.out.println("Retragere reușită. Sold: " + balance);
    }

    public double getBalance() {
        return balance;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000);

        try {
            account.deposit(500);
            account.withdraw(2000);
        } catch (InvalidAmountException e) {
            System.out.println("Eroare sumă: " + e.getMessage());
        } catch (InsufficientFundsException e) {
            System.out.println("Eroare sold: " + e.getMessage());
        }

        System.out.println("Sold final: " + account.getBalance());
    }
}
```

LABORATOR 8 -EXCEPȚII

4. Creează două excepții personalizate: InvalidAmountException și InsufficientStockException.
   - Implementează metoda procesareComanda(int stoc, int cantitate, double pret) care:
   - aruncă InvalidAmountException dacă prețul ≤ 0
   - aruncă InsufficientStockException dacă cantitatea > stoc
   - aruncă ArithmeticException dacă cantitatea = 0
   - altfel calculează și returnează pret / cantitate

   În main, apelează metoda și tratează toate cele trei tipuri de excepții cu mesaje corespunzătoare.

```java
// Excepție pentru preț invalid
class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}
// Excepție pentru stoc insuficient
class InsufficientStockException extends Exception {
    public InsufficientStockException(String message) {
        super(message);
    }
}
public class Comanda {
    public static double procesareComanda(int stoc, int cantitate, double pret)
            throws InvalidAmountException, InsufficientStockException {
        if (pret <= 0) {
            throw new InvalidAmountException("Prețul trebuie să fie pozitiv.");
        }
        if (cantitate > stoc) {
            throw new InsufficientStockException("Stoc insuficient pentru comandă.");
        }
        if (cantitate == 0) {
            throw new ArithmeticException("Împărțire la zero - cantitatea nu poate fi 0.");
        }
        double total = pret / cantitate;
        return total;
    }
```

LABORATOR 8 -EXCEPȚII

```java
    public static void main(String[] args) {
        try {
            double rezultat = procesareComanda(10, 0, 50);
            System.out.println("Total comandă: " + rezultat);

        } catch (InvalidAmountException e) {
            System.out.println("Eroare de preț: " + e.getMessage());

        } catch (InsufficientStockException e) {
            System.out.println("Eroare de stoc: " + e.getMessage());

        } catch (ArithmeticException e) {
            System.out.println("Eroare matematică: " + e.getMessage());

        } catch (Exception e) {
            System.out.println("Alt tip de eroare: " + e.getMessage());
        }
    }
}
```