Ingineria Sistemelor de Programare

Gestionarea erorilor

mihai.hulea@aut.utcluj.ro

2017

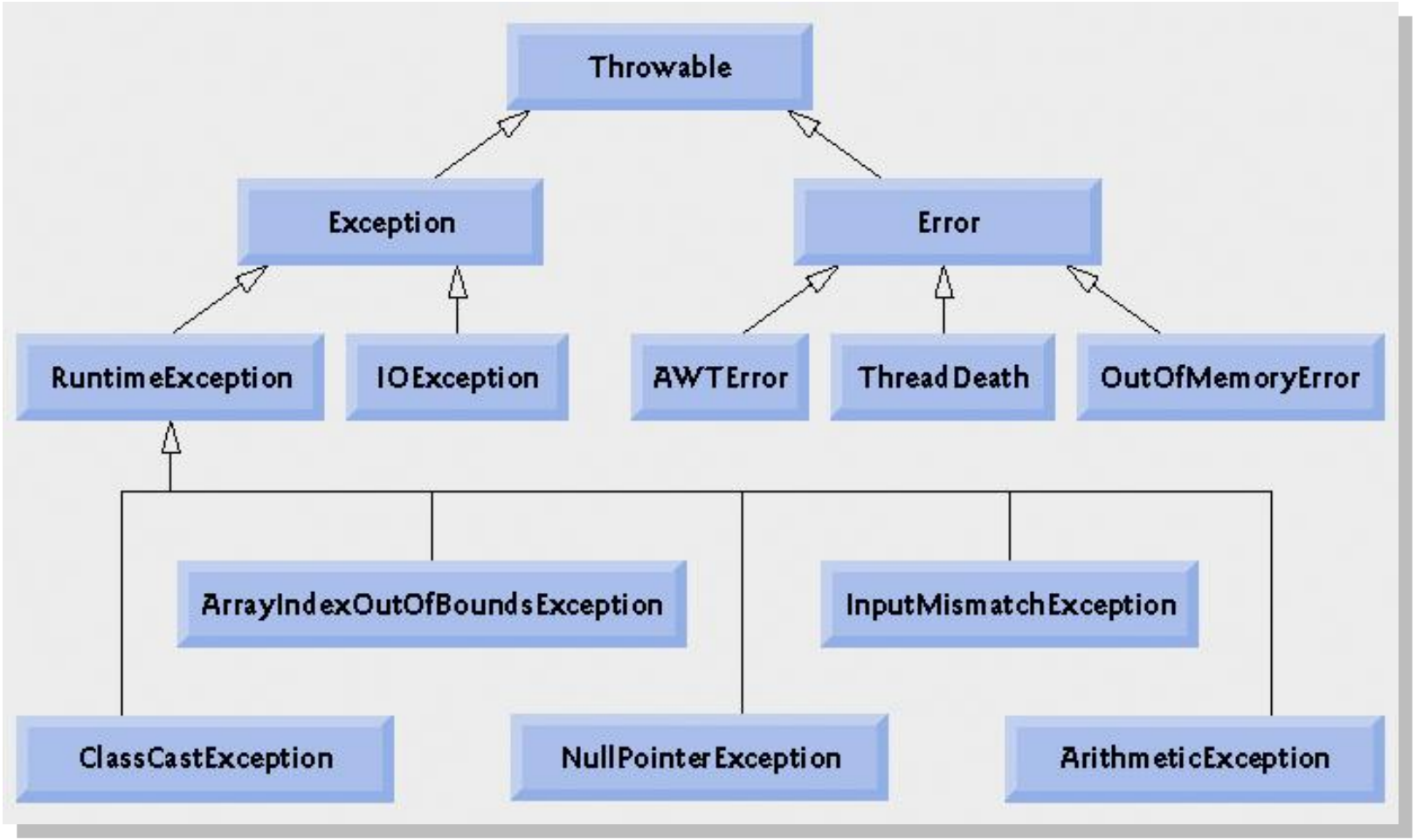
Introducere

- O eroare este o conditie care determina schimbarea fluxului normal al programului
- In Java erorile sunt numite exceptii
- Exceptiile sunt modelate sub forma de clase
- Se utilizeaza blocuri *try-catch-finally* pentru gestionarea erorilor
  - Codul care poate genera erori este pus in blocuri *try*
  - Codul care trateaza eroare este pus in blocul *catch*
  - Clauza *finally* este executata tot timpul

Terminologie

- “Thrown exception” – o exceptie care a avut loc
- “Stack trace” – numele exceptiei care a avut loc impreuna cu un mesaj descriptiv si secventa de apeluri de metode care a dus la apritia exceptiei
- “Thrown point” – locul de unde exceptia a fost aruncata initial

Ierarhia de clase pentru exceptii



Ierarhia de clase pentru exceptii

- Clasa Throwable
  - Subclasa *Exception* – se refera la erori care pot aparea in cadrul programului si pot fi prinse si gestionate de catre aplicatie
  - Subclasa *Error* – se refera la erori grave care apar in cadrul masinii virtuale java si care in general nu permit recuperarea aplicatiei

Ierarhia de clase pentru exceptii

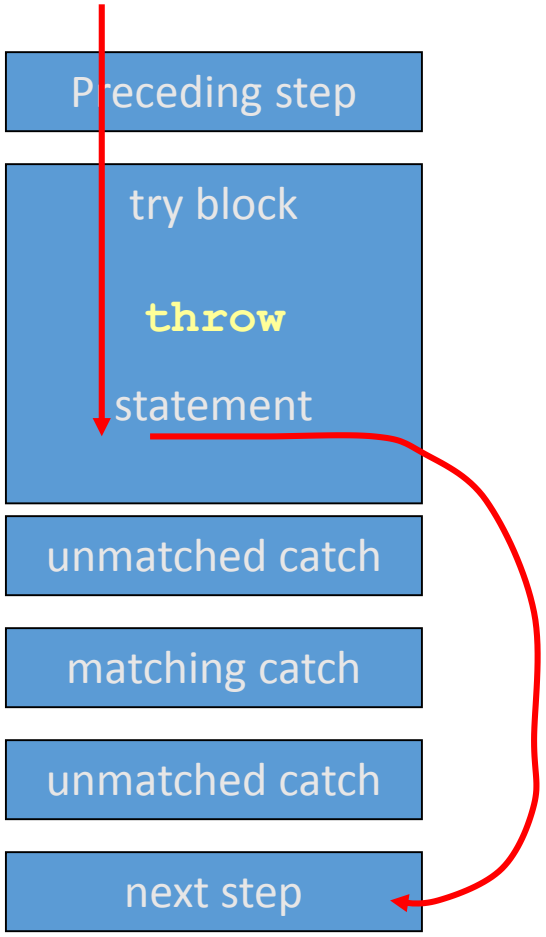
- Checked exceptions
  - Sunt mostenite din clasa Exception
  - Compilatorul forteaza prinderea acestor tipuri de erori
- Unchecked exceptions
  - Sunt mostenite din clasele RuntimeException si Error
  - Compilatorul nu forteaza prinderea acestor exceptii
  - Daca o astfel de eroare apare si nu este prinsa programul isi incheie executia

Prinderea exceptiilor

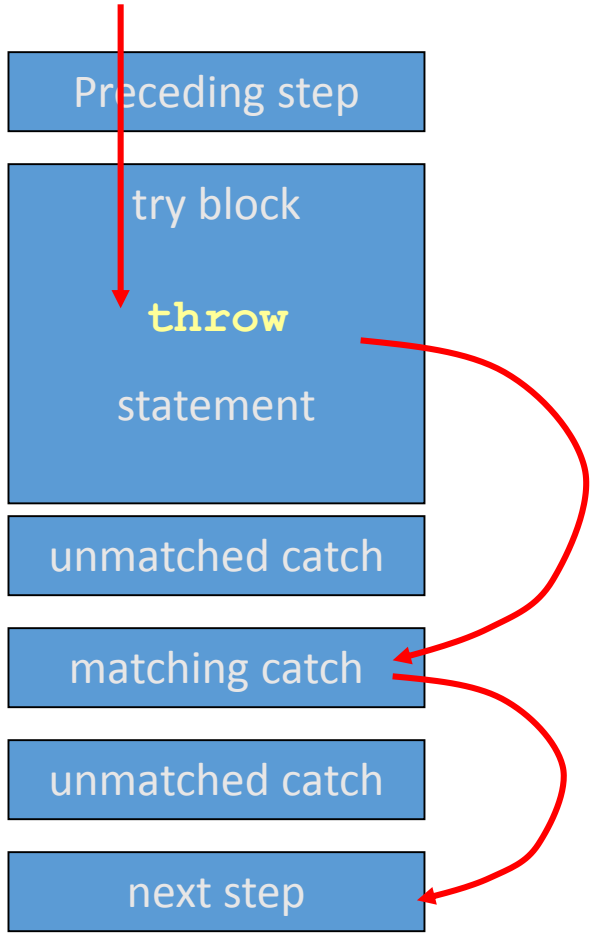
```
try {
        // statement that could throw an exception
   }
catch (ExceptionType1 e1) {
        // statements that handle the exception
}
catch (ExceptionType2 e2) { //e higher in hierarchy
        // statements that handle the exception
}
finally {
        // optional, execute evrytime
        //release resources
}
//other statements
```

Executia blocurilor try-catch

Executie fara erori



Executie cu aparitie de exceptie



```
public void writeList() {
   PrintWriter out = null;

   try {
      System.out.println("Entering" + " try statement");

      out = new PrintWriter(new FileWriter("OutFile.txt"));
      for (int i = 0; i < SIZE; i++) {
         out.println("Value at: " + i + " = " + list.get(i));
      }
   } catch (IndexOutOfBoundsException e) {
      System.err.println("Caught IndexOutOfBoundsException: "
                  +  e.getMessage());

   } catch (IOException e) {
      System.err.println("Caught IOException: " +  e.getMessage());

   } finally {
      if (out != null) {
         System.out.println("Closing PrintWriter");
         out.close();
      }
      else {
         System.out.println("PrintWriter not open");
      }
   }
}
```

Constuirea propriilor exceptii

- Se extinde clasa Exception sau RuntimeException

```
class TemperatureException extends Exception{
    int t;
    public TemperatureException(int t,String msg) {
        super(msg);
        this.t = t;
    }

    int getTemp(){
        return t;
    }
}//.class
```

Aruncarea exceptiilor

- Se utilizeaza *throws* pentru a defini exceptiile pe care o metoda le poate arunca
- Se utilizeaza instructiunea *throw* pentru a arunca exeptia

```
class CofeeDrinker{
    void drinkCofee(Cofee c) throws TemperatureException,
ConcentrationException{
        if(c.getTemp()>60)
            throw new
TemperatureException(c.getTemp(),"Cofee is to hot!");
        if(c.getConc()>50)
            throw new
ConcentrationException(c.getConc(),"Cofee concentration to
high!");
        System.out.println("Drink cofee:"+c);
    }
}//.class
```