



LATVIJAS  
UNIVERSITĀTE

ANNO 1919

NACIONĀLAIS  
ATTĪSTĪBAS  
PLĀNS 2020



EIROPAS SAVIENĪBA

Eiropas Sociālais  
fonds

IEGULDĪJUMS TAVĀ NĀKOTNĒ

# Java programmēšanas pamati



VADĪBAS UN  
UZŅĒMĒJDARBĪBAS  
MĀCĪBU CENTRS

# 10. NODARBĪBA

# IZŅĒMUMI

## KAS IR IZŅĒMUMS (EXCEPTION)?

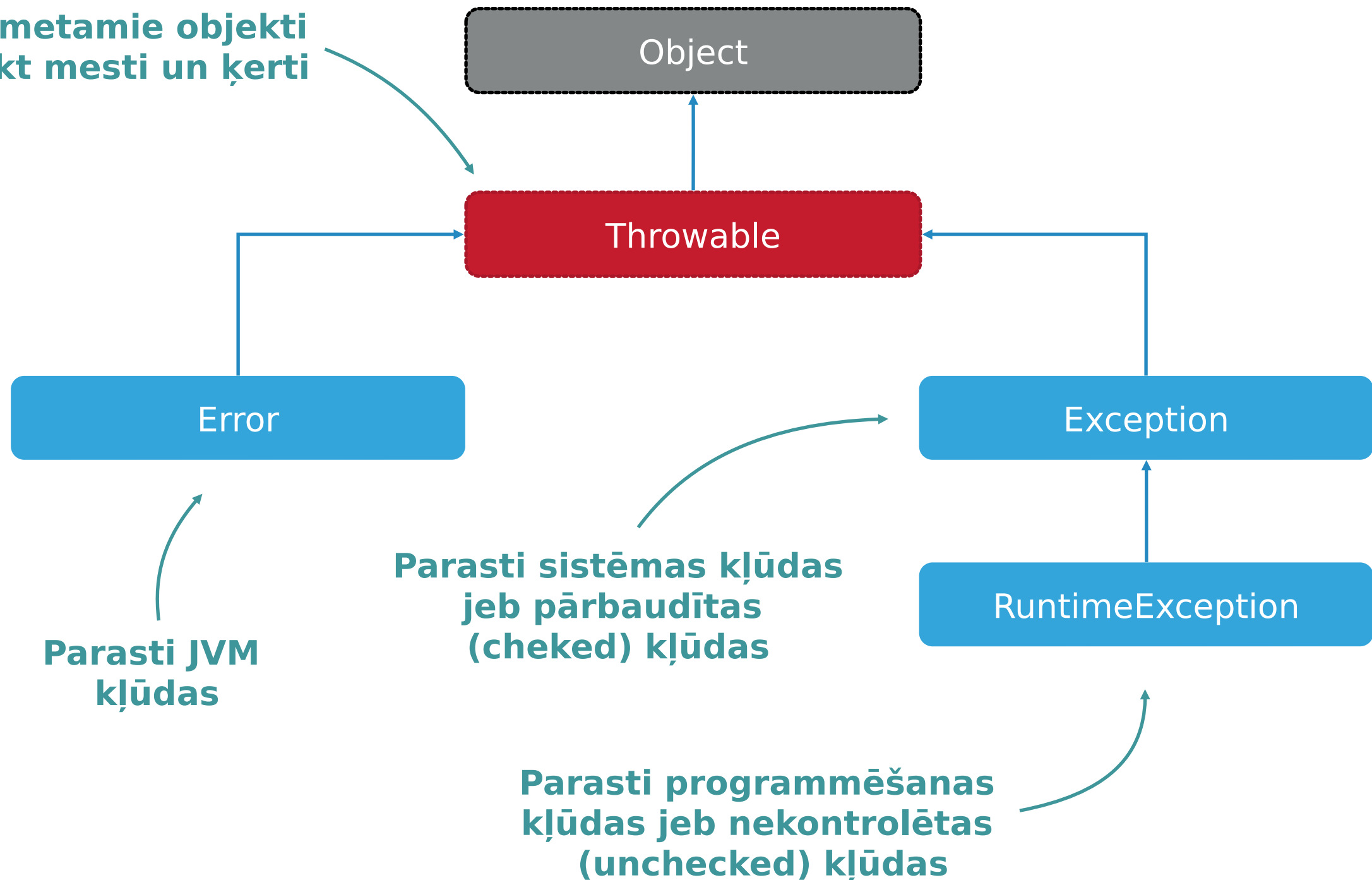
- ▶ Izņēmums ir **notikums**, kas notiek, ja programmas normālo instrukciju izpildi **negaidīti pārtrauc** (piem. piekļuve null norādei, ārpus masīva robežām, u.c.)
- ▶ Izņēmums ir **objekts**, kas apzīmē **kļūdu**, kas radusies metodē, un satur:
  - ▶ Informāciju par kļūdu, tai skaitā **tipu**
  - ▶ Programmas **stāvokli**, kad kļūda notika
- ▶ Izņēmuma objektu var **mest** (throw) un **ķert** (catch)

# KĻŪDU KLASIFIKĀCIJA UN IZŅĒMUMI

- ▶ JVM kļūdas
  - ▶ OutOfMemoryError, StackOverflowError, u.c.
- ▶ Sistēmas kļūdas
  - ▶ FileNotFoundException, IOException, u.c.
- ▶ Programmēšanas kļūdas
  - ▶ NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, u.c.

# IZŅĒMUMU HIERARHIJA

Tikai metamie objekti  
var tikt mesti un ķerti



# KĀPĒC LIETOT IZŅĒMUMUS?

- ▶ Izņēmumi **atdala** kļūdu apstrādes **loģiku** no **biznesa** loģikas
  - ▶ Algoritms kļūst **tīrāks** un **vienkāršāks**
- ▶ Izņēmumi **izplata** kļūdu uz augšu pa **izsauktajām** metodēm
  - ▶ Izsauktajām metodēm nav **atsevišķi** jāapstrādā kļūdas, tās tiek nodotas uz augšu automātiski
- ▶ Izņēmumu klases **grupē** un **izšķir** kļūdu tipus
  - ▶ Kļūdas var grupēt pēc to **vecāka** klases (polimorfisms)
  - ▶ Kļūdas var šķirot pēc to **faktiskās** klases
- ▶ Izņēmumi **standartizē** kļūdu apstrādi

## PĀRBAUDĪTI (CHECKED) IZŅĒMUMI

- ▶ Parasti lieto, lai apzīmētu paredzamās sistēmas kļūmes ar saprātīgu atkopšanu (piemēram, failu sistēmā trūkstoša faila vai savienojuma izveides kļūme)
- ▶ Kompilators liek tās atsevišķi apstrādāt
- ▶ Metodēm, kuras var mest pārbaudītu kļūdu, ir tas ir jādefinē metodes parakstā
- ▶ Metodēm, kuras izsauc metodes, kas met pārbaudītus izņēmumus, ir divas iespējas:
  - ▶ Apstrādāt tos (no tiem ir iespējams atkopties)
  - ▶ Deklarēt metodes parakstā, ka tā met pārbaudītu izņēmumu. Tādejādi nodot to uz augšu izsaucošajai metodei
- ▶ Klase Exception objekti un tās atvasinājumi ir pārbaudīti (izņemot RuntimeException)



# NEKONTROLĒTI (UNCHECKED) IZŅĒMUMI

- ▶ Parasti lieto, lai apzīmētu **negaidītas programmēšanas** vai **loģiskas** kļūdas
- ▶ Kompilators **neliek** tās atsevišķi **apstrādāt**
- ▶ Tiek pieņemts, ka programma **nespēj atgūties** no šiem izņēmumiem
- ▶ Error un RuntimeException un to atvasinājumi ir **nekontrolēti** izņēmumi



## NEKONTROLĒTS IZŅĒMUMS: PIEMĒRS

Simbolu virknei  
nav norādes

```
String emptyString = null;  
emptyString.isEmpty();
```

Izsaucot metodi kaut kam, kas  
neeksistē saņems klūdas  
objektu **NullPointerException**


## PĀRBAUDĪTS IZŅĒMUMS: PIEMĒRS

Definē, ka metode  
var mest izņēmumu



```
public String readFile(String path) throws IOException {  
    byte[] bytes = Files.readAllBytes(Paths.get(path));  
    return new String(bytes);  
}
```

Nedroša operācija, jo  
datne var būt salauzta vai  
to nav iespējams nolasīt



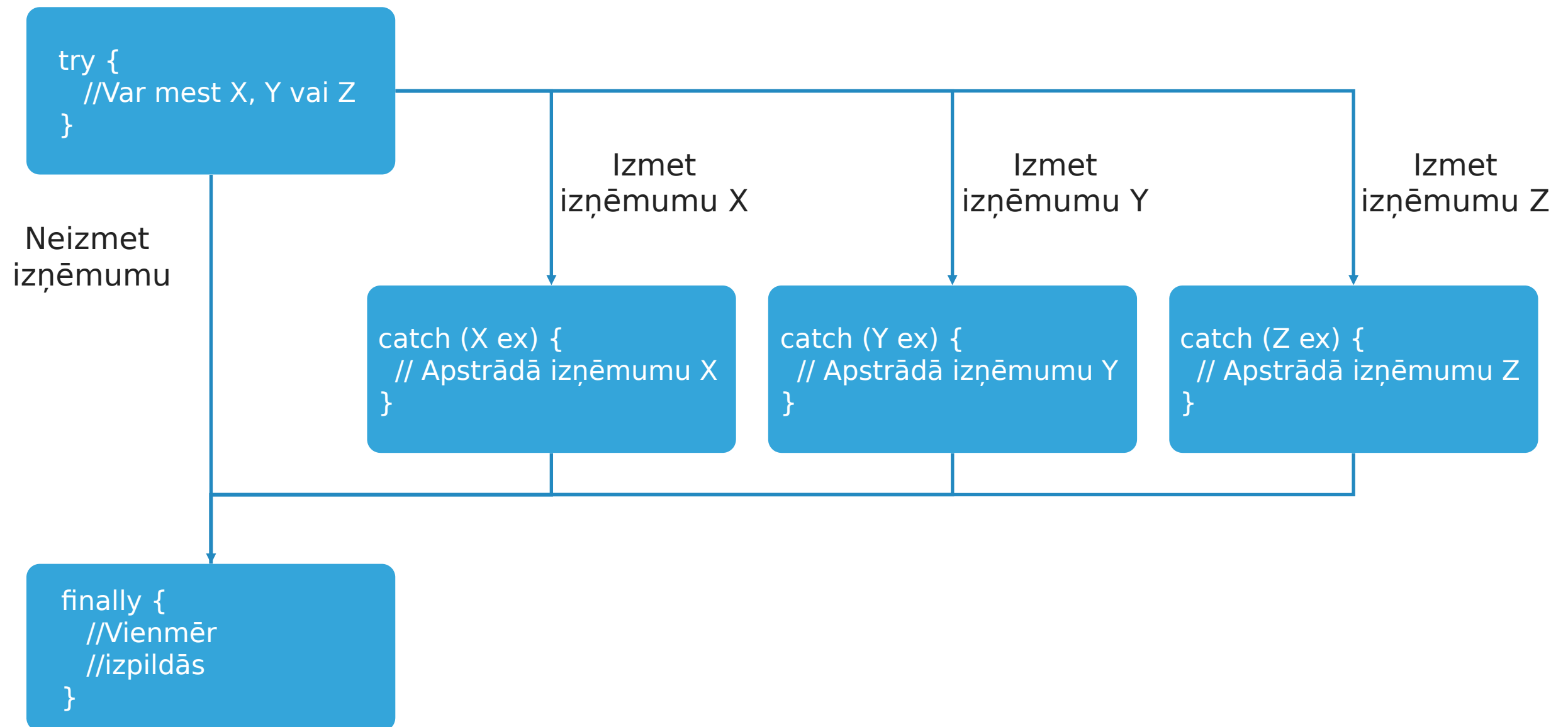
## IZŅĒMUMA DZĪVES CIKLS

1. Pēc izņēmuma objekta izveides tas **tiek nodots** izpildlaika sistēmai (runtime system) jeb izmests
2. Izpildlaika sistēma mēģina **atrast** izņēmuma apstrādātāju, izsekojot **izsaukto** metožu sakārtotajam sarakstam jeb izsaukumu kaudze (call stack)
3. Ja apstrādātājs tiek **atrasts**, tad izņēmums ir *noķerts*:
  - I. Izņēmums tiek **apstrādāts** vai **izmests** (re-thrown) tālāk iepriekšējai izsaucošajai metodei
4. Ja apstrādātājs **nav atrasts** (izpildlaiks atgriežas līdz pat main() metodei) :
  - I. Izņēmuma “metožu izsaukumu pēdas” (stack trace) tiek izdrukāts **kļūdu** kanālā
  - II. Programma pārtrauc tās izpildi

## IZŅĒMUMU APSTRĀDE

- ▶ Java darbam ar izmestu izņēmumu, izmanto **try-catch-finally** struktūru
- ▶ Sastāv no **trīs blokiem**:
  - ▶ **try** bloks
    - ▶ Obligāts, ja tiek izsaukta metode, kas var izmest nekontrolētu izņēmumu
    - ▶ Nosaka, kā izņēmums tiks apstrādāts, ja tāds tiks izmests
  - ▶ **catch** bloks
    - ▶ Obligāts un var tikt deklarēts tik reizes, cik tas ir nepieciešams
    - ▶ Nosaka, kā tiks apstrādāts noteiktais izņēmums vai to grupa
  - ▶ **finally** bloks
    - ▶ Nav obligāts un tiek deklarēts tikai vienu reizi (obligāts, ja nav catch bloka)
    - ▶ Vienmēr tiks izpildīts pēc try un visiem catch blokiem

# TRY-CATCH-FINALLY: PLŪSMAS DIAGRAMMA



## TRY-CATCH-FINALLY: PIEMĒRS

Nedroša metode ir tā,  
kuras parakstā ir  
atslēgvārds “throws”

```
try {  
    //Izsauc nedrošu metodi  
} catch (Exception1 ex) {  
    //apstrādā Exception1 tipa izņēmumu  
} catch (Exception2 ex) {  
    //apstrādā Exception2 tipa izņēmumu  
} finally {  
    //Šis bloks tiek vienmēr izpildīts  
    //pēc try un catch blokiem  
}
```

Visu parakstā deklarēto  
izņēmumu apstrāde

# ATSAUCES

- ▶ <https://docs.oracle.com/javase/tutorial/essential/exceptions/>
- ▶ <https://beginnersbook.com/2013/04/try-catch-in-java/>
- ▶ [https://www.tutorialspoint.com/java/java\\_exceptions.htm](https://www.tutorialspoint.com/java/java_exceptions.htm)
- ▶ <https://stackify.com/best-practices-exceptions-java/>





LATVIJAS  
UNIVERSITĀTE  
ANNO 1919