Изузеци

Шта су изузеци и чему служе?

Изузецима се сигнализирају озбиљни проблеми приликом извршавања програма.

Главна корист од изузетака је што раздвајају кôд који обрађује грешке од кôда који се извршава када ствари теку глатко.
Други позитиван аспект изузетака је што нас присиљавају да реагујемо на одређене грешке.

Важно је схватити да не треба све грешке у програмима сигнализирати изузецима – само неуобичајене и катастрофалне. Разлог је што руковање изузецима укључује много додатног процесирања, па ће програм који рукује изузецима већи део времена бити знатно спорији него што би морао да буде.

У Јави, изузетак је **објекат** који се креира када се деси „изузетна“ ситуација у нашем програму.
Тај објекат садржи атрибуте у које се смештају информације о природи проблема.

За изузетак се каже да је **бачен** (енг. `thrown`).
Специфично парче кôда које се пише да би руковало одређеном врстом изузетка **хвата** (енг. `catch`) тај изузетак.

Изузетак је увек објекат неке поткласе стандардне класе **`Throwable`**. То важи и за изузетке које сами дефинишемо и за стандардне изузетке.
(Када се каже да је нешто стандардно, мисли се да је део Јавине библиотеке).

Класа Throwable

Објекат типа `Throwable` садржи 2 информације о изузетку:

- поруку о грешци
- запис са стека извршавања у тренутку креирања изузетка.
  Стек извршавања чува информације о свим методима који су у стању извршења у сваком датом тренутку.
  Он обезбеђује средство помоћу кога се извршењем наредбе `return` за неки метод извршавање наставља од одговарајуће наредбе позивајућег метода.
  Запис са стека извршавања који се чува у објекту изузетку садржи број линије изворног кôда у којој се десило избацивање изузетка, а затим податке о позивима метода који непосредно претходе тачки у којој се десио изузетак. Подаци о позивима метода (пуно квалификовано име метода плус број линије у којој је позив метода) уређени су тако што се најскорије позван метод појављује као први.

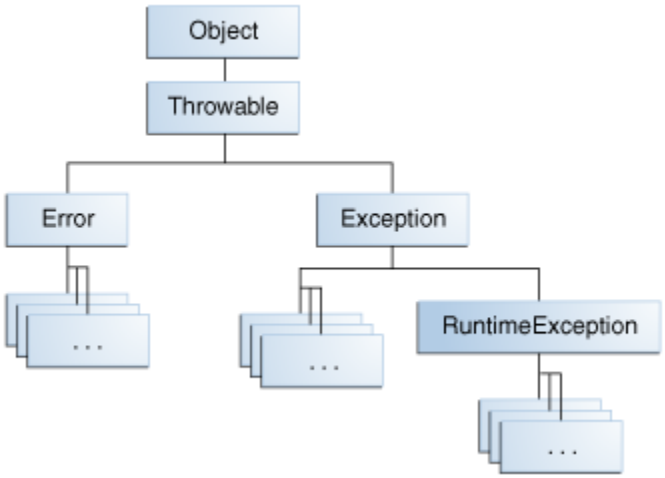
Класа има 2 конструктора: подразумевани и конструктор који прихвата аргумент типа `String`.
Тај стринг се користи да обезбеди опис природе проблема који је изазвао тај изузетак.

Јавни методи класе служе да приступимо поруци и стању стека извршавања:

- `getMessage()` – враћа садржај поруке која описује текући изузетак
  (типично садржи пуно квалификовано име класе изузетка и кратак опис изузетка)
- `printStackTrace()` – исписује поруку и стање стека на стандардни излаз за грешку – што је у случају конзолне апликације екран
- `printStackTrace(PrintStream s)` – исто као претходни метод, осим што се излазни ток задаје као аргумент. Позив претходног метода за објекат изузетак `e` је еквивалентан са:

```
e.printStackTrace(System.err);
```

Класа `Throwable` има две директне потклase: **`Error`** и **`Exception`**.
Ове класе имају потклase које описују конкретне изузетке.



Error изузеци

Изузеци дефинисани класом **`Error`** и њеним потклasама карактеришу се чињеницом да се од нас не очекује да предузимамо ништа, не очекује се да их хватамо. То су изузеци који се **не проверавају**.
Они су резултат катастрофалних догађаја и услова и у таквим ситуацијама обично све што можемо да урадимо јесте да прочитамо поруку о грешци која се генерише када се избаци изузетак и да покушамо да схватимо шта је у нашем кôду могло да изазове тај проблем.

Exception и RuntimeException изузеци

За скоро све изузетке представљене потклasама класе **`Exception`** морамо у наш програм укључити кôд који ће руковати њима уколико наш кôд може узроковати њихово избацивање. То су изузеци који **се проверавају**. (Речено је *скоро све* јер се ово не односи на `RuntimeException` изузетке)

Ако метод у нашем програму може да генерише изузетак типа који има `Exception` као суперкласу, морамо или руковати изузетком унутар тог метода (`try-catch`) или регистровати да наш метод може избацити такав изузетак (`throws`). У супротном, програм се неће искомпајлирати.

Разлог због ког се **`RuntimeException`** изузеци (изузеци који као базну имају класу `RuntimeException`, изведену из класе `Exception`) другачије третирају и компајлер допушта да их игноришемо је тај што они генерално настају због озбиљних грешака у нашем кôду и у већини ситуација можемо да урадимо мало да поправимо ситуацију.
Међутим, у неким контекстима, за неке од ових изузетака, то није увек случај и можда желимо да укључимо кôд за њихово препознавање. RuntimeException изузеци се **не проверавају**.

Поткласе `RuntimeException` дефинисане у стандардном пакету `java.lang` су:

- `ArithmeticException` (нпр. дељење `int`-а нулом)
- `IndexOutOfBoundsException` (нпр. за низ, `String` или `Vector` објекат)
- `NegativeArraySizeException` (покушај дефинисања низа негативне димензије)
- `NullPointerException` (коришћење промењиве која садржи `null` у случају када она треба да садржи референцу на објекат да би се позвао метод или приступило атрибуту)
- `ArrayStoreException` (покушај смештања у низ објекта који није одговарајућег типа)
- `ClassCastException` (покушај кастовања у неодговарајући тип)
- `IllegalArgumentException` (прослеђивање аргумента који не одговара типу параметра метода)
- `SecurityException` (нпр. покушај аплета да чита фајл на локалној машини)
- `IllegalMonitorStateException` (покушај нити да чека на монитор објекта који не поседује)
- `IllegalStateException` (покушај позива метода у тренутку када то није допуштено)
- `UnsupportedOperationException` (захтевање извршавања операције која није подржана)

Руковање изузецима

Ако кôд унутар метода може избацити изузетке који нису типа `Error` нити `RuntimeException` (и њихових потклаcа, што се подразумева), морамо нешто да предузмемо поводом тога.
Имамо избор:

- можемо обезбедити кôд који ће, унутар метода, руковати произвољним избаченим изузетком или
- нагласити да метод може избацити изузетак одговарајућег типа, чиме ће изузетак бити прослеђен позивајућем методу.

Задавање типова изузетака које метод може избацити

Ако се изузетак који се проверава не хвата унутар метода, мора се задати да метод може да доведе до његовог избацивања. То се чини клаузом `throws` која се пише након листе параметара метода, а у њој се, раздвојени запетама, наводе типови изузетака које метод може избацити. Нпр.

```
double mojMetod() throws IOException {…}
double mojMetod() throws IOException, FileNotFoundException {…}
```

Метод који позива `mojMetod()` мора узети у обзир изузетке које он може избацити па или их обрађивати или и он декларисати да избацује изузетке тог типа.

Руковање изузецима

Ако желимо да рукујемо изузецима тамо где се они десе, можемо укључити 3 врсте блокова кôда:
`try` блок – обухвата кôд у коме се може јавити један или већи број изузетака које желимо да ухватимо
`catch` блок – обухвата кôд који је намењен руковању изузецима одређеног типа који могу бити избачени у `try` блоку
`finally` блок – увек се изврши пре него што се метод заврши, без обзира да ли је било који изузетак избачен у `try` блоку или не

```
try{
      // kod koji moze izbaciti izuzetak tipa ArithmeticException
}catch(ArithmeticException e){
      // kod za rukovanje izuzetkom tipa ArithmeticException
}
```

catch блок ће процесирати изузетке типа класе која му је задата као параметар, али и типа свих потклаcа те класе.

ток извршавања
Када нема избацивања изузетка унутар `try` блока, извршава се читав `try` блок, а затим се прелази на прву наредбу након `catch` блока (кôд у `catch` блоку се не извршава).
У случају избацивања изузетка, контрола се непосредно преноси на прву наредбу `catch` блока. Након што се `catch` блок заврши, извршавање се наставља од наредбе која следи за `catch` блоком. Наредбе `try` блока након тачке у којој се десило избацивање изузетка се не извршавају.

Променљиве декларисане унутар `try` блока доступне су само унутар тог блока.
`catch` блок је одвојен опсег од `try` блока.

`try` и `catch` су неодвојиви (пример са вежби: ако желимо да се петља заврши избацивањем изузетка, читаву је ставимо у `try` блок)

Вишеструки `catch` блокови

Појављују се у случају када `try` блок може избацити већи број изузетака различитих типова.
Ако дође до избацивања изузетка, извршава се само први `catch` блок који може да га ухвати,
тј. онај `catch` блок чији је параметар истог типа као и избачени изузетак или типа неке његове натклase.
Стога је редослед којим се наводе `catch` блокови од значаја када постоји хијерархија: `catch` за руковање најизведенијим типом се наводи први, а за руковање најосновнијим типом последњи.
У супротном се јавља грешка при компајлирању.

multi-catch

Почев од Јаве 7, могуће је да један `catch` блок рукује већим бројем типова изузетака.
Тиме се може избећи понављање кôда у случајевима када се на исти начин рукује различитим типовима изузетака.
Типови изузетака којима `catch` блок рукује раздвајају се усправним цртама (|). Када `catch` блок рукује већим бројем изузетака, његов параметар је имплицитно `final` (унутар блока није му могуће променити вредност).

Нпр. вишеструки `catch` блок

```
catch (IOException ex) {
      logger.log(ex);
      throw ex;
}
catch (SQLException ex) {
      logger.log(ex);
      throw ex;
}
```

се може краће записати помоћу multi-catch:

```
catch (IOException|SQLException ex) {
      logger.log(ex);
      throw ex;
}
```

Не можемо имати само `try` блок. Њега увек мора да прати бар један од `catch` и `finally` блокова.

finally блок

Када дође до избацивања изузетка, остатак `try` блока се не извршава, без обзира колико је битан кôд који се ту налази. `finally` блок се извршава увек, пре краја метода, без обзира да ли је или не избачен изузетак у придруженом `try` блоку. Његова једина намена је да ослободи коришћене ресурсе (затвори фајлове, сокете, конекције са базама података,...). Уколико је нека вредност помоћу `return` враћена из `finally` блока, то поништава наредбу `return` која је евентуално извршена у `try` блоку. Иако је могуће користити контролу тока у `finally` блоку (`break`, `continue`, `return`), не треба то радити! Детаљније о томе у пратећем документу „finally.pdf“.

try-with-resources

У питању је наредба `try` која декларише један или већи број ресурса. Декларације ресурса се наводе унутар пара облих заграда које непосредно следе кључну реч `try`.
Ресурс је објекат који се мора затворити када програм заврши рад са њим. Наредба try-with-resources гарантује да је, на њеном крају, сваки ресурс затворен. Као ресурс се може користити сваки објекат класе која имплементира `java.lang.AutoCloseable`, што укључује објекте класа које имплементирају `java.io.Closeable`.

Унутар try-with-resources могуће је декларисати и већи број ресурса. Декларације се раздвајају тачка-запетом.
По завршетку кôда који непосредно следи, било нормално било због изузетка, аутоматски се позивају методи `close()` за ресурсе редоследом обрнутим од редоследа којим су креирани.

try-with-resources може имати `catch` и `finally` блокове, као и обичан `try`. За try-with-resources, `catch` и `finally` блокови се извршавају након затварања декларисаних ресурса.

Реизбацивање изузетка

У многим ситуацијама, када неки метод ухвати изузетак имплементирањем одговарајуће `catch` клаузе, позивајући метод можда мора да зна да се то десило. Ако је то потребно, можемо да реизбацимо изузетак из унутрашњости catch блока користећи наредбу `throw`. Нпр.

```
try{
      // ...
}catch(ArithmeticException e){
      // obrada ovog izuzetka
      throw e;
}
```

`throw` наредбу чини кључна реч `throw` за којом следи објекат изузетак који се избацује.