JAVA 1

Iznimke

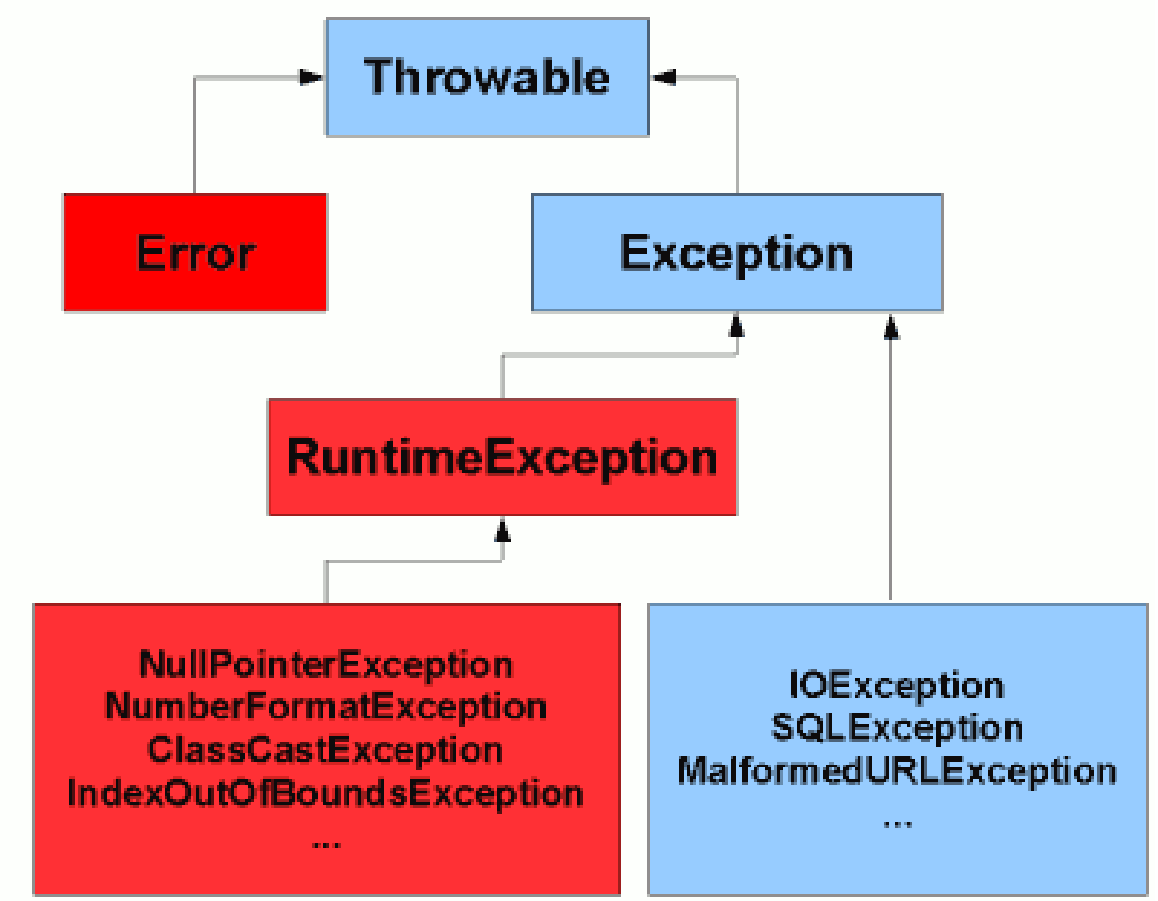
Teme

- Iznimke općenito
- Hijerarhija iznimaka
- *try-catch-finally* i *multicatch* blokovi
- Redoslijed obrade iznimaka
- *try-with-resources*
- Vlastite iznimke

Iznimke općenito

- Java, kao i mnogi objektno orijentirani programski jezici, greške koje se pojavljaju pri izvršavanju programa nazivaju iznimkama (*exceptions*)
- vrlo jasna sintaksa koja nam omogućuje rješavanje iznimaka
- *Exception* je objekt i stoga možemo napustiti paradigmu vraćanja vrijednosti kao status
- izbjegavati *Exception-Driven Development*!
- jasna podjela odgovornosti između pozivatelja (*caller*) i pozvanoga (*callee*)

Hijerarhije iznimaka



Izvor:https://www.javamex.com/tutorials/exceptions/exceptions_hierarchy.shtml

Errors

- iznimke koje su se dogodile uvjetovane vanjskim faktorima
- primjer - JVM zakaže radi nedostatka radne memorije
  - instancira se *OutOfMemoryError* koji je specijalizacija*VirtualMachineError* iznimke
- grešku se ne može ispraviti programski

Exceptions

- iznimke koje se mogu riješiti programskim putem
- *RuntimeException* i njene podklase – ne zahtjeva obradu
  - *ArithmeticException* – dijeljenje s 0
  - *ClassCastException* – kriva pretvorba u specifičniji tip
  - *NullPointerException* – poziv metode nad objektom koji je *null*
- Checked exceptions – sve koje nisu podklasa *RuntimeException*
  - *FileNotFoundException* – datoteka ne postoji
  - *MalformedURLException* – krivo specificiran URL
  - *SQLException* – općenita iznimka prilikom rada sa SQL

try-catch-finally blok

- iznimke se u kôdu rješavaju kroz *try-catch-finally* blokove
- kôd u kojem je moguća iznimka stavljamo u *try* blok
- ako se dogodi iznimka, izvršit će se kôd koji smo stavili u *catch* blok – nije obavezan u *try-finally* bloku
- *finally* blok - uvijek se izvršava i nije obavezan u *try-catch* bloku
  - omogućava zatvaranje resursa

try-catch-finally blok

```
try {

} catch (Exception e) {

} finally {

}
```

Multicatch blok

- ako je kôd koji se izvršava za različite iznimke identičan, možemo spriječiti redundanciju

```
try {

} catch (ArithmeticException | IndexOutOfBoundsException e) {

}
```

Redoslijed obrade iznimaka

```
try {

} catch (MostSpecificException e) {

} catch (LeastSpecificException e) {

}
```

try-with-resources

- znatno olakšava rad sa resursima
- resurs mora implementirati *AutoCloseable/Closeable* sučelje

```
Scanner scanner = null;
try {
    scanner = new Scanner(new File(PATH));
} finally {
    if (scanner != null) {
        scanner.close();
    }
}
```

```
try (Scanner scanner = new Scanner(new File(PATH))){

} catch (FileNotFoundException ex) {


}
```

Kreiranje vlastitih iznimaka

- ako naslijedimo *RuntimeException* ili neku podklasu
  - u metodi koja baca grešku (*callee*) nije nužno deklarirati iznimku već je dovoljno baciti grešku sa *throw* izrazom
  - u metodi koja poziva (*caller*) nije nužno obraditi iznimku

- ako nasljedimo klasu koja u hijerarhiji nije *RuntimeException*
  - u metodi koja baca grešku (*callee*) nužno je deklarirati iznimku *throws* izrazom u potpisu metode i baciti grešku sa *throw* izrazom
  - u metodi koja poziva (*caller*) nužno je obraditi iznimku
  - eksplicitno izražena odgovornost za iznimku

Demo

- Project

Izvor:http://www.jnhsolutions.com/contact-us/request-a-demo/

Hvala na pažnji!