Informatika 2

Riešenie výnimočných stavov

doc. Ing. Ján Janech, PhD.
25. 3. 2024

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Pojmy zavedené v 5. prednáške (1)

- abstraktná trieda
  - v reálnom svete
  - v jazyku Java
  - v BlueJ/UML
- konkrétna trieda
- abstraktná metóda
  - v reálnom svete
  - v jazyku Java

Pojmy zavedené v 5. prednáške (2)

- vzťah is-a
- dedičnosť a extenzia triedy
- instanceof v hierarchii dedičnosti

Pojmy zavedené v 5. prednáške (3)

- Liskovej substitučný princíp
- riešenie problému štvorec-obdĺžnik pomocou abstraktnej triedy
- riziká porušenia LSP
  - znižovanie rizika
- trieda Object a LSP

- polymorfizmus vs. dedičnosť

Pojmy zavedené v 5. prednáške (4)

- vytváranie dedičnosti
  - princíp gen-spec
- implementačná závislosť v dedičnosti

Cieľ prednášky

- komunikácia klient-server
- defenzívne programovanie
- ošetrenie chýb
  - na strane servera
  - na strane klienta
- výnimky
- testovanie výnimiek

- príklad: KCalB

Behové chyby (1)

- chyby v kóde, ktoré spôsobia pád aplikácie
  - nekorektné ukončenie
  - aplikácia sa vymkne spod kontroly
  - nečakané ukončenie v lepšom prípade

Behové chyby (2)

- má padnúť internetový prehliadač, ak žiadame otvoriť neexistujúcu web stránku? (napr. sme pri písaní adresy urobili chybu)

- má sa zavrieť (ukončiť prácu) súborový manažér, ak chceme uložiť obrázok na USB kľúč, na ktorom už nie je dostatok miesta?

- má skončiť hra, ak hráč napíše nesprávny príkaz?

Behové chyby (3)

- problémy tohto typu chceme riešiť

Komunikácia klient-server (1)

- komunikácia dvojice objektov – odosielateľ správou žiada adresáta o službu
- odosielateľ – klient žiada o službu
  - klient je aktívny objekt
- adresát – server poskytuje službu
  - server nevykonáva žiadnu akciu z vlastnej vôle, len reakcia na žiadosť klienta

Komunikácia klient-server (2)

- správa – parametre
- správnosť parametrov – prípustné hodnoty

- kto je zodpovedný?

- klient – definícia hodnôt parametrov
- server – použitie hodnôt parametrov

Klient

- definícia hodnôt parametrov
  - doplňujúce informácie správy
  - podrobnejšia charakteristika žiadosti
  - klient vychádza zo svojho stavu

- spracovanie návratovej hodnoty

Server

- server – použitie hodnôt parametrov
  - parametre sú charakterizované typom
  - nie každá hodnota musí byť prípustná
  - určitá hodnota parametra môže vyžadovať určitý stav servera
  - parametre môžu vyžadovať zmenu stavu servera

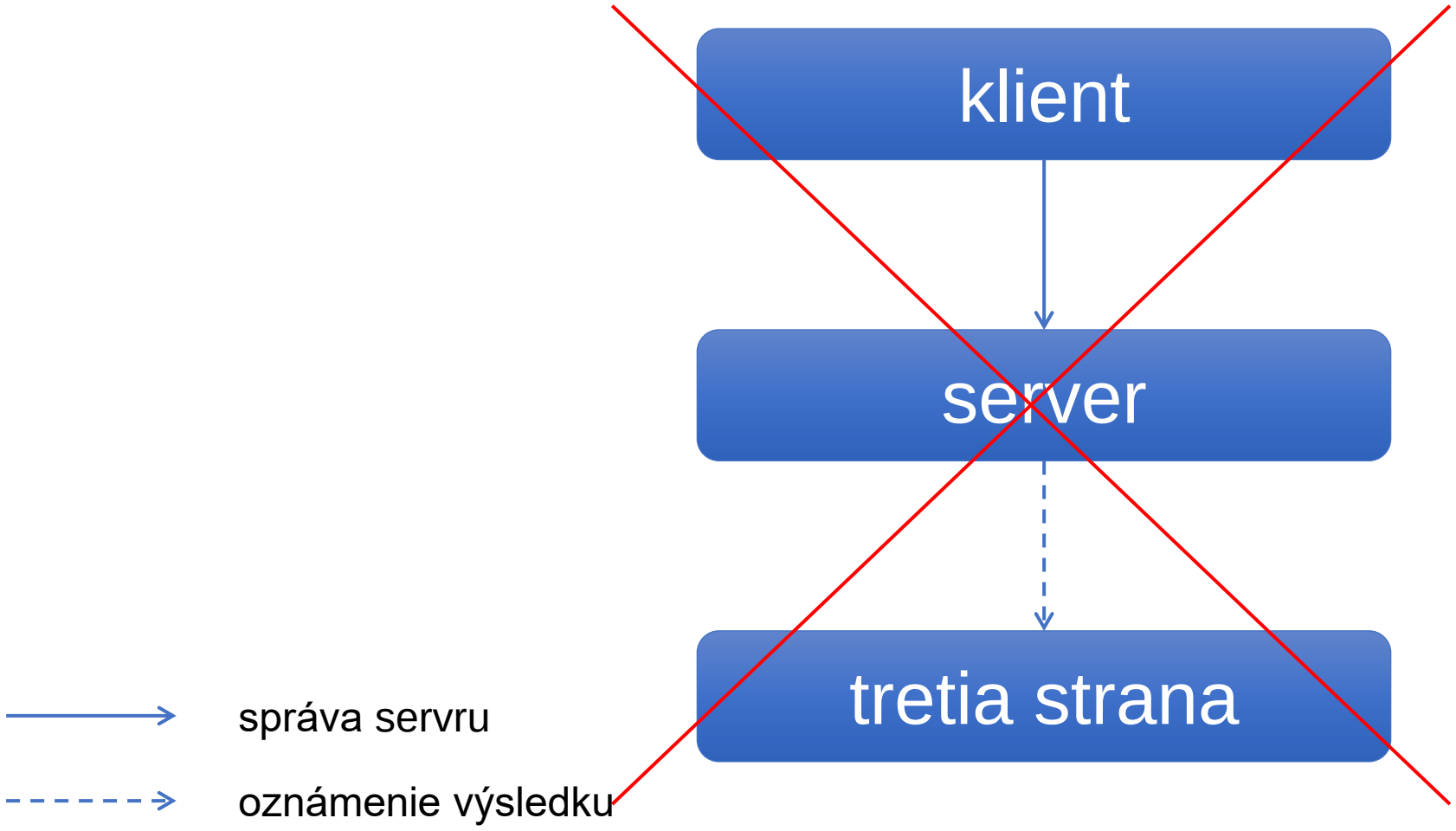
- server nesmie dopustiť, aby sa dostal do nekorektného stavu

Korektný vzťah klient-server

správa servru

oznámenie výsledku

Porušenie vzťahu klient-server



Príklad porušenia vzťahu klient-server

- výpis na terminál
  - výpis oznámenia o korektnom ukončení operácie
  - oznámenie chyby používateľovi
- klient sa nedozvie o výsledku operácie

Komunikácia klient-server

- kto je zodpovedný?

- extrémne (nie nesprávne) spôsoby riešenia
  - klient vie presne, čo chce a ako to má od servera korektne žiadať
  - server predpokladá, že sa môže stať čokoľvek a musí sám zabezpečiť, aby pracoval len korektne

Východiská

- klient nekladie nekorektné požiadavky zámerne

- existujúce nekorektné požiadavky – chyby
- existujú problémy mimo aplikácie – klient nemôže poznať

Riešenie – odpovede na otázky

- Aká veľká kontrola správ od klienta má byť v metódach servera?
- Ako má server oznamovať chyby klientovi?
- Ako má klient predchádzať nekorektným požiadavkám na server?
- Ako má klient reagovať na oznámenie o chybe?

Riešenie na strane servera

- kontrola požiadaviek
- reakcia na zistenú chybu
  - nevykoná akciu
  - nezmení svoj stav
  - informovanie o chybe

Príklad – Katalog.vymaz (1)

```
public void vymaz(String titul) {
    var polozka = this.vyhladaj(titul);
    this.zoznam.remove(polozka.get());
}
```

- titul
  - nesmie byť null
  - nesmie byť prázdny reťazec
  - musí existovať AudiovizualneDielo s takým titulom

Príklad – Katalog.vymaz (2)

```
public void vymaz(String titul) {
    if (titul == null || titul.isEmpty()) {
        // osetrenie chyby 1,2
    } else {
        var polozka = this.vyhladaj(titul);
        if (polozka.isEmpty()) {
            // osetrenie chyby 3
        } else {
            this.zoznam.remove(polozka.get());
        }
    }
}
```

Informovanie o chybe

- žiadne – ???
- informovanie používateľa – výpis
- informovanie klienta

Informovanie používateľa

- informácia o chybe sa zobrazí používateľovi

- textová forma – terminál, okno
- zvukový výstup – „pípanie“ pri stlačení nesprávneho klávesu
- ...

- doteraz v projektoch – výpis na terminál

Oznámenie chyby používateľovi

```
public void vymaz(String titul) {
    if (titul == null || titul.isEmpty()) {
        System.out.println("Titul musi byt zadany!");
    } else {
        var polozka = this.vyhladaj(titul);
        if (polozka.isEmpty()) {
            System.out.println("Titul nie je v katalogu!");
        } else {
            this.zoznam.remove(polozka);
        }
    }
}
```

Informovanie používateľa – použitie

- prečo/kedy áno
  - jednoduché na implementáciu
  - ak používateľ potrebuje vedieť výsledok operácie
- prečo/kedy nie
  - klient sa nedozvie výsledok operácie
    - porušenie princípu klient/server
    - nie vždy je klientom používateľ
  - nie o všetkých výsledkoch má byť informovaný používateľ

PocitaciKatalog.vymaz

- Príklad problému:

```
public void vymaz(String titul) {
    super.vymaz(titul);
    this.pocet--;
}
```

- počet odráta aj vtedy, keď sa nepodarí vymazať
  - nevie o tom

Informovanie klienta – návratová hodnota

- info o chybe

- boolean – false (nastala chyba), true (nenastala)
- int – číselný kód (1 – taká chyba; 2 – iná chyba;
  0 – bez chyby)
- enum – položky – rôzne typy chýb
- objekt – null v prípade chyby (napr. pri vyhľadávaní)

Návratová hodnota objekt

```
public Optional<AudiovizualneDielo> vyhladaj(String titul) {
    for (var polozka : this.zoznam) {
        if (polozka.getTitul().equals(titul)) {
            return Optional.of(polozka);
        }
    }
    return Optional.empty();
}
```

Návratová hodnota objekt – použitie

- štandardné riešenie pri vyhľadávaní
- hľadaný objekt neexistuje => Optional.empty(), resp. null

Návratová hodnota boolean

```
public boolean vymaz(String titul) {
    if (titul == null || titul.isEmpty()) {
        return false;
    }

    var polozka = this.vyhladaj(titul);
    if (polozka.isEmpty()) {
        return false;
    }

    this.zoznam.remove(polozka);
    return true;
}
```

Návratová hodnota boolean – použitie

- jediná informácia – nastala chyba
- vhodné pre jedinú možnosť
- nevhodné pre viac rôznych chýb v metóde – ak existuje viac možností ich ošetrenia

Návratová hodnota enum

```
public ChybaVymazania vymaz(String titul) {
    if (titul == null || titul.isEmpty()) {
        return ChybaVymazania.TITUL_NEZADANY;
    }

    var polozka = this.vyhladaj(titul);
    if (polozka.isEmpty()) {
        return ChybaVymazania.TITUL_SA_NENASIEL;
    }

    this.zoznam.remove(polozka);
    return ChybaVymazania.ZIADNA;
}
```

Návratová hodnota enum – použitie

- možnosť označenia konkrétnych chýb
- komplikované a prácne
- takmer nepoužívané

Informovanie klienta – výnimky (1)

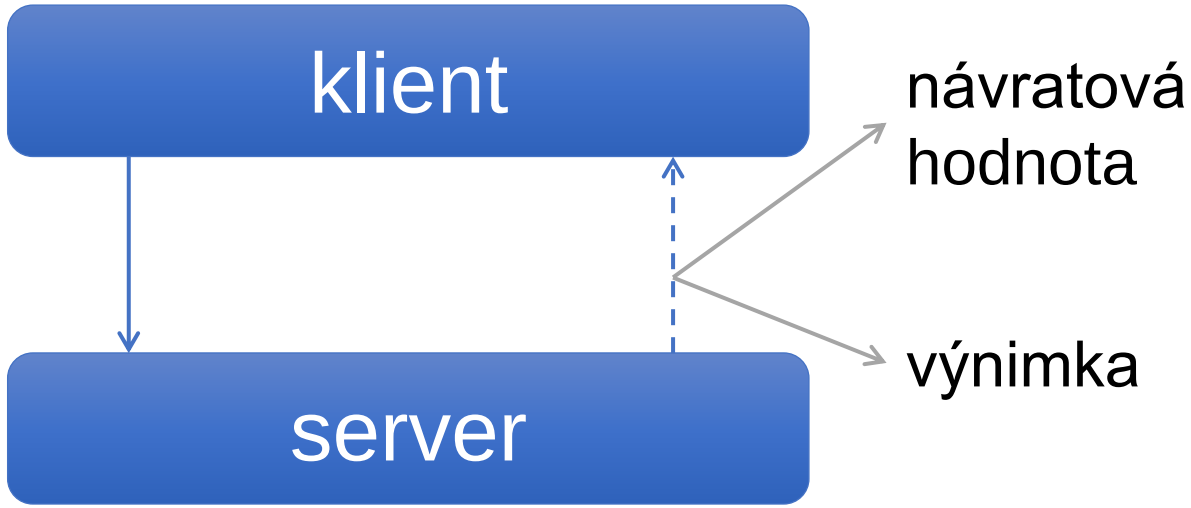
- kontrola výnimočných stavov
- oznamovanie chyby klientovi
- vo väčšine prípadov preferované riešenie

- doposiaľ známe ako behové chyby

Informovanie klienta – výnimky (2)

- výnimka – objekt
- informácia – názov triedy výnimky
- informácia – popis situácie

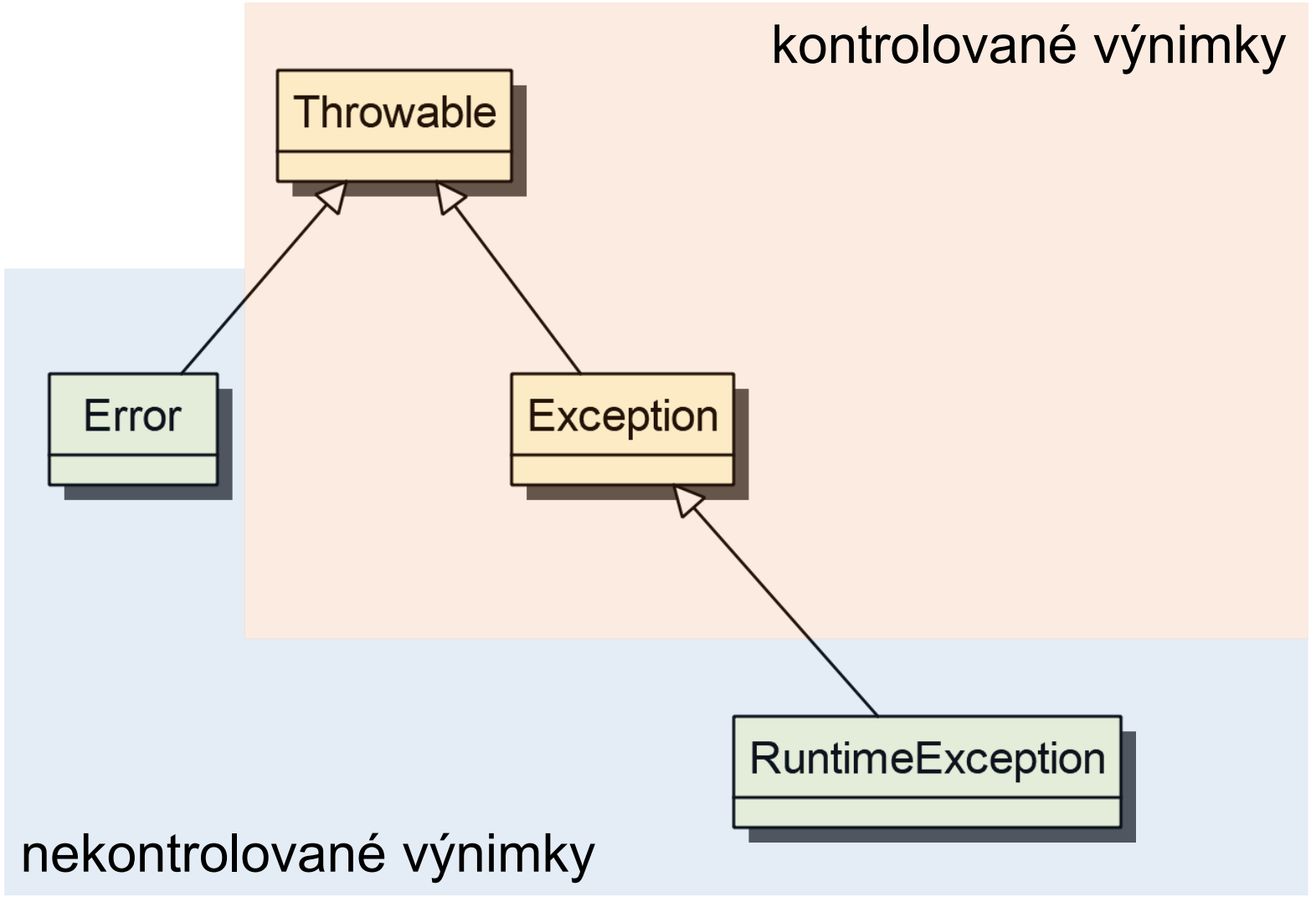
Klient-server – výnimky



Druhy výnimiek – Java

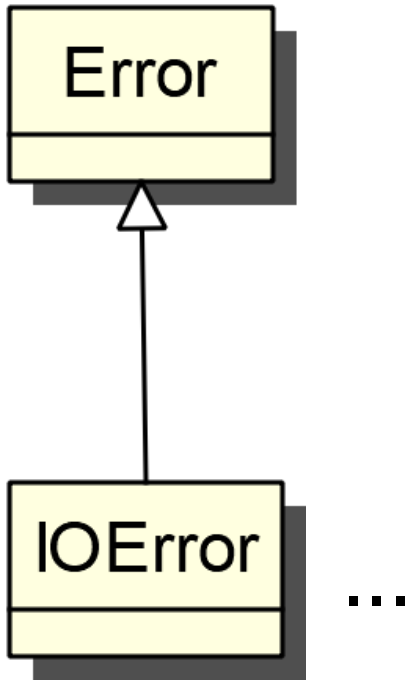
- kontrolované
  - klient nemôže ignorovať
  - kontroluje prekladač
- nekontrolované
  - klient môže ignorovať
  - spôsobí predčasné ukončenie aplikácie – pád
  - prekladač nerieši

Hierarchia výnimiek – koreň Throwable

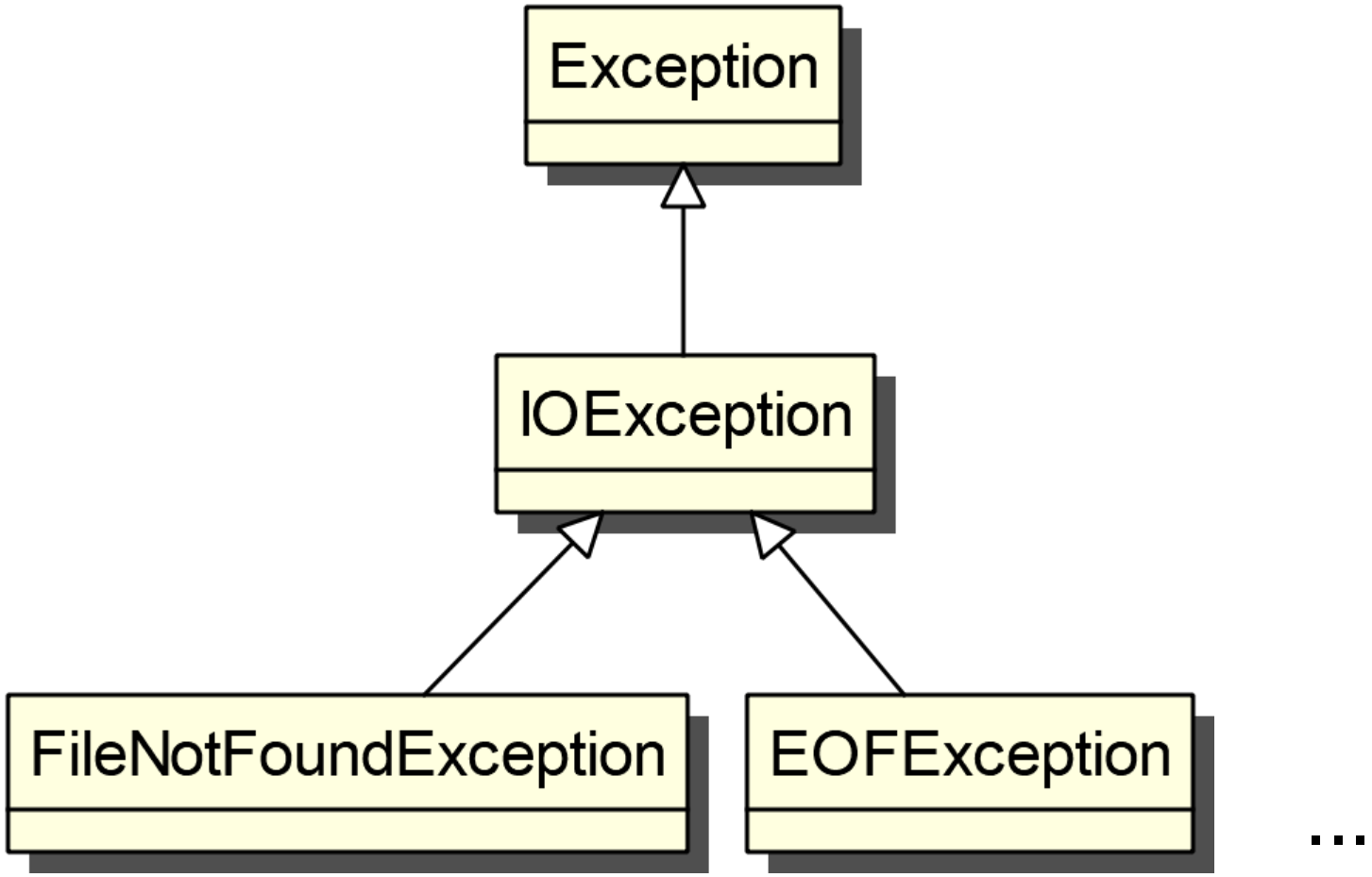


Chyby – Error

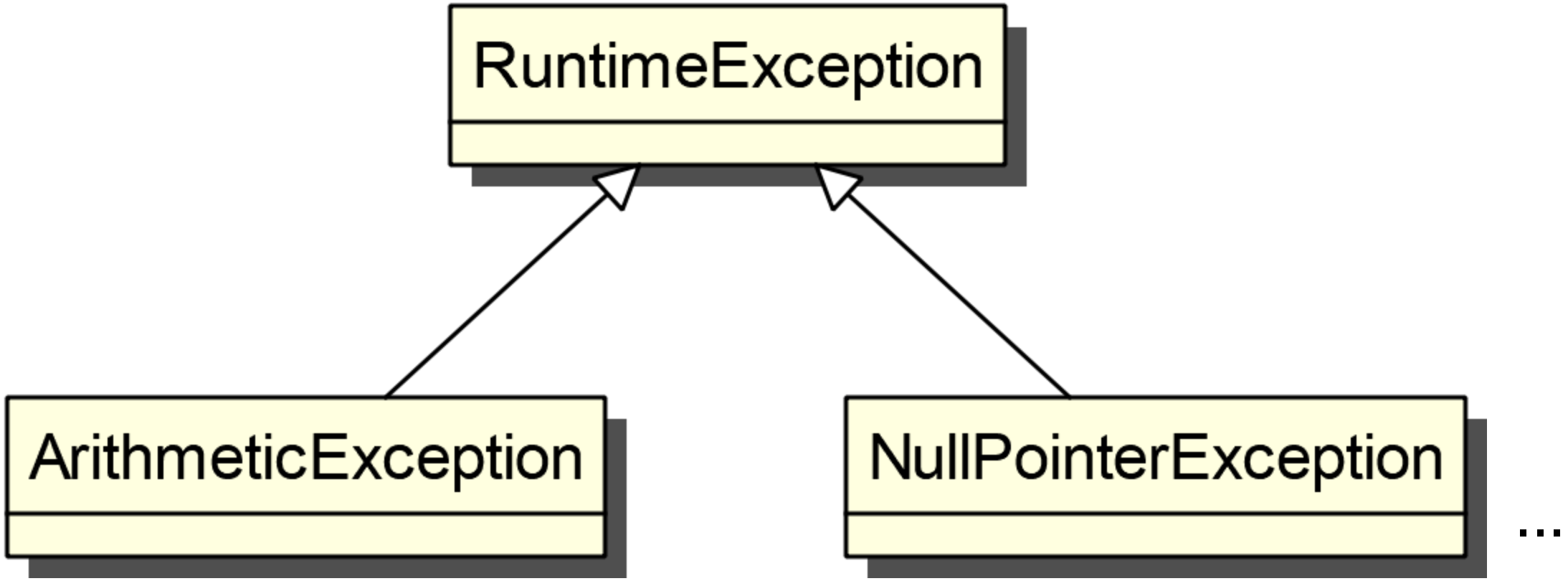
- systémové chyby
- neodchytávajú sa



Kontrolované výnimky



Nekontrolované výnimky



Výnimky a klient-server

- server výnimky vytvára – „vyhadzuje“
- klient výnimky zachytáva

Vyhodenie výnimky (1)

- príkaz throw

```
var vynimka = new TypVynimky("popis");
throw vynimka;
```

- skrátený – bežný zápis

```
throw new TypVynimky("popis");
```

Vyhodenie výnimky (2)

- príkaz throw ukončuje vykonávanie metódy
- server vracia riadenie klientovi
- informuje klienta o výnimke

Klauzula throws

- throws v hlavičke metódy
  - výnimky vyhadzované v metóde
  - čiarkou oddelený zoznam
  - kontrolované povinne
  - nekontrolované nepovinne

```
public void uloz(String nazovSuboru)
                                    throws IOException
```

Priklad – vyhodenie výnimky (1)

```
public void vymaz(String titul) {
    if (titul == null || titul.isEmpty()) {
        throw new IllegalArgumentException("titul je prazdny");
    }
    ...
}
```

Priklad – vyhodenie výnimky (2)

```
...
var polozka = this.vyhladaj(titul);
if (polozka.isEmpty()) {
    throw new IllegalArgumentException("titul sa nenasiel");
}

this.zoznam.remove(polozka);
...
```

Zachytávanie výnimky

- na strane klienta
- definujeme akcie, ktoré sa vykonajú v prípade výnimky
- príkaz try-catch

Príkaz try-catch (1)

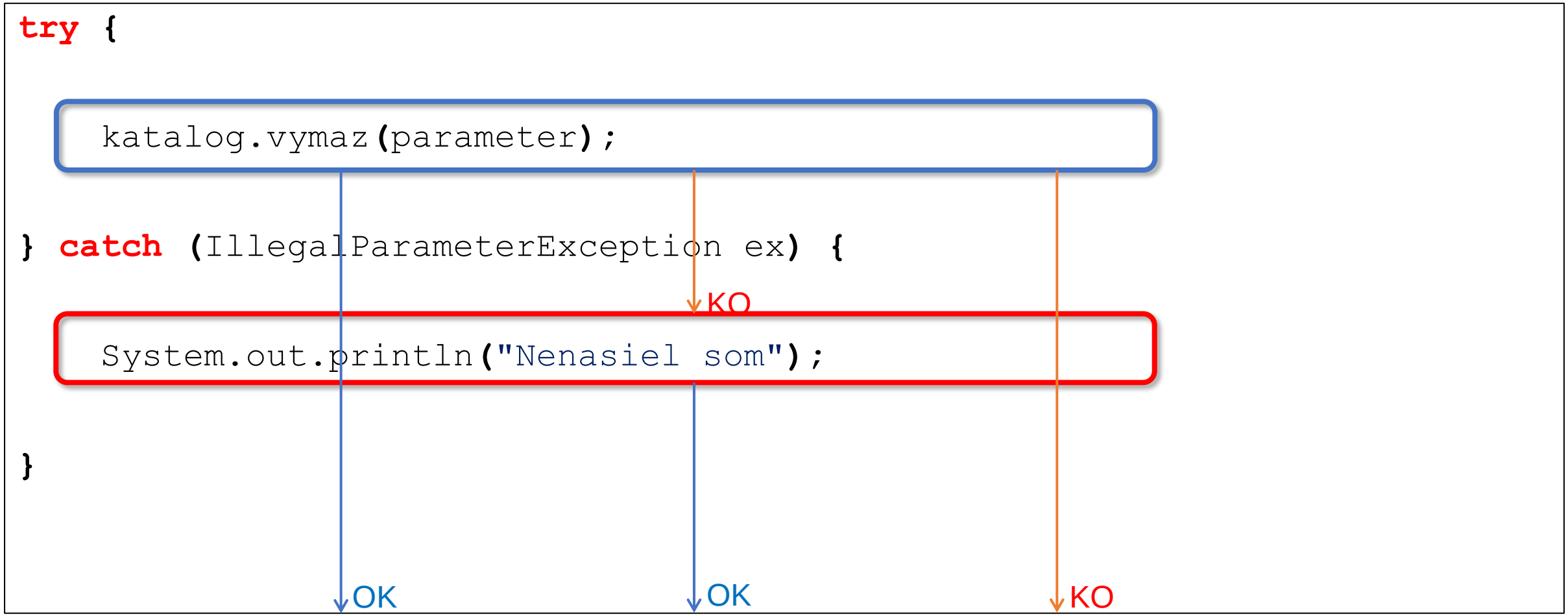
- bloky: try, catch
- samostatne catch pre každú triedu výnimiek

```
try {
    // chranene prikazy
} catch (TriedaVynimky vynimka) {
    // prikazy vykonane v pripade vynimky
}
```

Príkaz try-catch (2)

```
try {
    katalog.vymaz(parameter);
} catch (IllegalArgumentException ex) {
    this.terminal.vypisRiadok("Nenasiel som");
}
```

Príkaz try-catch (3)



Viac blokov catch

- postupné vyhľadávanie typu výnimky
- POZOR! len prvý vhodný blok catch

- bloky catch musia byť usporiadané
  - (najskôr potomok – potom predok)

- výnimka sa nenájde
  - informuje sa klient
  - behová chyba

Re-throw

- spracovanie výnimky
- Následné postúpenie výnimky ďalej (klientovi)
- anglicky (termínus-technikus) re-throw

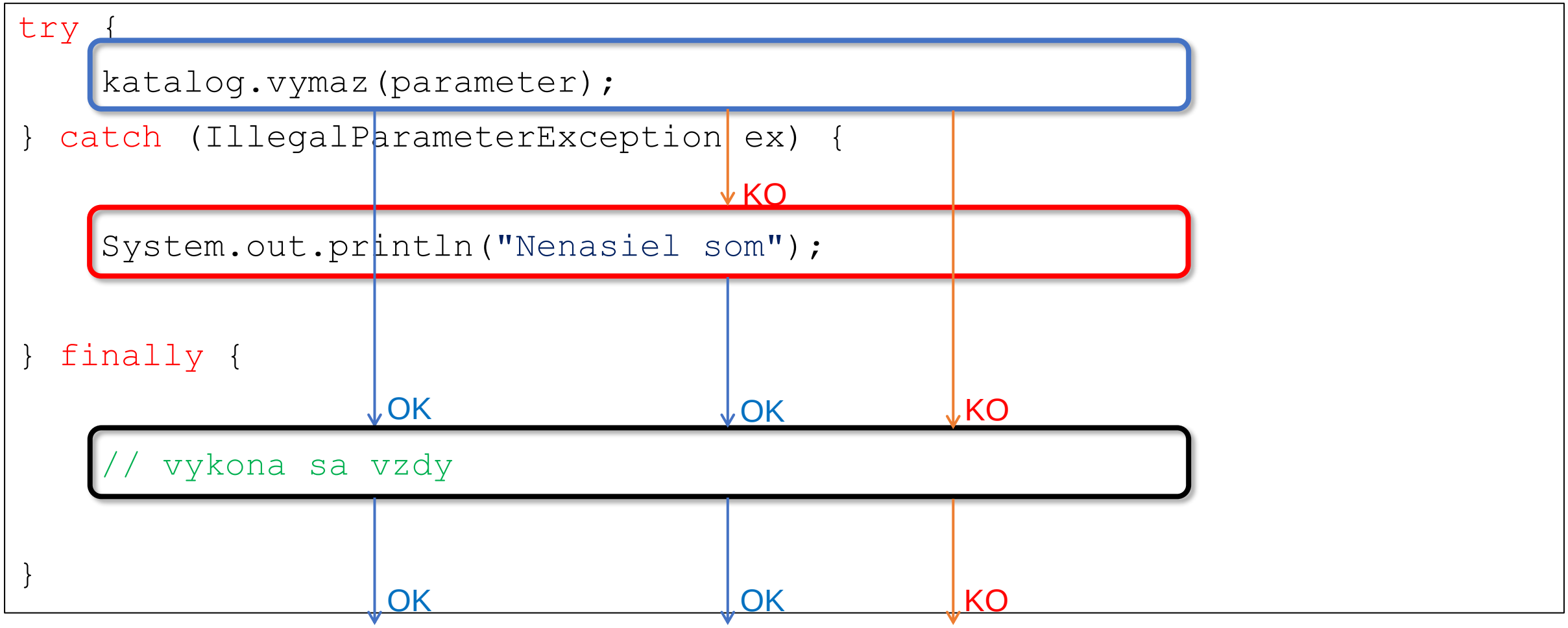
```
throw ex;
```

Príkaz try-catch-finally (1)

- blok finally – príkazy vykonávané vždy
  - ok – všetky príkazy bloku try + blok finally
  - ko – blok catch (ak taký je) + blok finally
- uzatvorenie systémových zdrojov – súbory

- verzia try-finally
- príkazy try môžu byť vnorené – catch, finally

Príkaz try-catch-finally (2)



Vlastné triedy výnimiek

- nová kontrolovaná výnimka – potomok triedy Exception (alebo od nej odvodenej triedy)
- nová nekontrolovaná výnimka – potomok triedy RuntimeException
- konvencia – názov končí slovom Exception

Dôvody použitia vlastných výnimiek

- neexistuje štandardná výnimka, ktorá dostatočne popisuje chybu
- vyčlenenie samostatnej výnimky
- existuje doplňujúca informácia, ktorá môže mať vplyv na riešenie chyby – kontrolované výnimky

Príklad triedy vlastnej výnimky

```
public class NeznameDieloException extends Exception {
    public NeznameDieloException(String nazov) {
        super("Dielo s nazvom " + nazov + " nebolo najdene.");
    }
}
```

Vlastná výnimka

- NeznameDieloException
- definícia triedy – odčlenenie výnimky od štandardných
- kontrolovaná výnimka
- zostavenie textu chybového hlásenia v konštruktore
- využitie konštruktora predka
- ostatné metódy získava od predka

Využitie vlastnej výnimky – Katalog

```
public void vymaz(String titul) throws NeznameDieloException {
    ...
    var polozka = this.vyhladaj(titul);
    if (polozka.isEmpty()) {
        throw new NeznameDieloException(titul);
    }
    this.zoznam.remove(polozka);
}
```

Metóda vymaz triedy Katalog

- klauzula throws – vyhadzuje kontrolovanú výnimku
- test výsledku vyhľadania
- vyhodenie výnimky

Zachytenie vlastnej výnimky (1)

```
try {
    katalog.vymaz(nazov);
} catch (NeznameDieloException ex) {
    System.out.println(ex.getMessage());
    ex.printStackTrace();
}
```

Zachytenie vlastnej výnimky (2)

- príkaz try pri poslaní správy
- chránený príkaz vymazania z katalógu
- zachytenie výnimky = zotavenie sa z chyby
  - informácia pre používateľa
- využitie zdedených metód
  - getMessage() – text správy o chybe
  - printStackTrace() – výpis textu na štandardný chybový výstup – objekt System.err

Kontrola stavu v konštruktore

- nesprávne parametre
- dve možnosti reakcie:
  - zmena počiatočného stavu na správny
  - vyhodenie výnimky

Zmena počiatočného stavu na správny

- riešenie používané doteraz
- napr. AutomatMHD
  - nekladná hodnota ceny lístka => použitie prednastavenej hodnoty
- klient nevie o probléme
  - možnosť informovať používateľa cez terminál
  - možnosť informovať klienta cez špeciálnu správu getChyba()

Vyhodenie výnimky v konštruktore

- oznámenie chyby klientovi
- inštancia sa vôbec nevytvorí, resp. sa zahodí

Príklad – server

```
public CD(String titul, String autor, int pocetSkladieb, int celkovyCas) {
    super(titul, celkovyCas);

    if (autor == null || autor.isEmpty()) {
        throw new IllegalArgumentException("autor musi byt nastaveny");
    }
    this.autor = autor;

    if (pocetSkladieb <= 0 ) {
        throw new IllegalArgumentException("pocetSkladieb musi byt vacsi ako nula");
    }
    this.pocetSkladieb = pocetSkladieb;
}
```

Príklad – klient

```
try {
    var cd = new CD(titul, autor, pocetSkladieb, cas);
    katalog.pridaj(cd);
} catch (IllegalArgumentException ex) {
    System.out.println("Nespravne parametre!");
}
```

Príklad – klient – nesprávne

```
CD cd;
try {
    cd = new CD(titul, autor, pocetSkladieb, cas);
} catch (IllegalArgumentException ex) {
    System.out.println("Nespravne parametre!");
}
katalog.pridaj(cd);
```

variable cd might not have been initialized

Príklad – klient – tiež nesprávne

```
CD cd = null;
try {
    cd = new CD(titul, autor, pocetSkladieb, cas);
} catch (IllegalArgumentException ex) {
    System.out.println("Nespravne parametre!");
}
katalog.pridaj(cd);
```

```
Nespravne parametre!
Exception in thread "main" java.lang.NullPointerException
        at fri.kcaib.katalog.Katalog.vypisPolozky(Katalog.java:29)
        at fri.kcaib.katalog.PocitaciKatalog.vypisPolozky(PocitaciKatalog.java:30)
        at fri.kcaib.Main.main(Main.java:37)
```

Príklad – klient – správne riešenie

```
CD cd = null;
try {
    cd = new CD(titul, autor, pocetSkladieb, cas);
} catch (IllegalArgumentException ex) {
    System.out.println("Nespravne parametre!");
}
if (cd != null) {
    katalog.pridaj(cd);
}
```

Možnosti práce s výnimkami v klientovi

- zotavenie sa z chyby
- predchádzanie chybám

Zotavenie sa z chyby

- príkazy v bloku catch
- riešenie situácie
- niekedy nový pokus s celým príkazom try
  - napr. vstupy
  - nemusí sa podariť
  - pozor na nekonečný cyklus

Príklad – čítanie dát cez internet (1)

```
…
var uspech = false;
var pocetPokusov = 0;
while (!uspech) {
    // pokus o stiahnutie dá z internetu – prikaz try
}
…
```

Príklad – zápis do súboru (2)

```
try {
    // stiahni dáta zo servera
    uspech = true; // posledny prikaz bloku
} catch (IOException e) {
    if (pocetPokusov < MAX_POCET) {
        pocetPokusov++;
        Thread.sleep(10000); // počkaj 10s
    } else {
        throw e; // vynimka sa posuva vyssie
    }
}
```

Predchádzanie chybám (1)

- vyžaduje spoluprácu klienta so serverom
- príklad – duplicita v katalógu CD a DVD
  - vyhladaj – vráti null, ak titul v katalógu nie je
  - ak sa skontroluje, nevyhodí sa výnimka
- server musí poskytnúť možnosť
- klient nemusí riešiť príslušnú výnimku

Predchádzanie chybám (2)

- nevýhody
  - zvyšuje sa implementačná závislosť
  - server sa poisťuje – dvojité testovanie

Výnimky a testovanie

- test – spôsobia neprípustné parametre výnimku?
- ! výnimka ukončí test chybou
  - nie je to, čo chceme

- dve možnosti riešenia
  - try-fail-catch
  - Assertions.assertThrows (JUnit 5.x)

try-fail-catch

```
@Test
public void newCDZlyTitul() {
    try {
        new CD(null, "Niekto", 5, 30);
        Assertions.fail("Neošetrená hodnota null");
    } catch (IllegalArgumentException ex) {
        // vynimka nastala = ok
    }
}
```

assertThrows (JUnit 5.x)

```
@Test
public void newCDZlyAutor() {
    Assertions.assertThrows(
        IllegalArgumentException.class,
        () -> {
            new CD("Nieco", null, 5, 30);
        }
    );
}
```