Objektovo orientované programovanie

(výnimky)

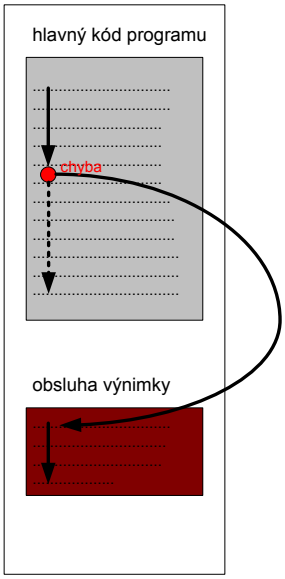
7. prednáška

Vladislav Novák
FEI STU v Bratislave
28.10.2014
(oprava 4.11 na str. 9)

Obsah

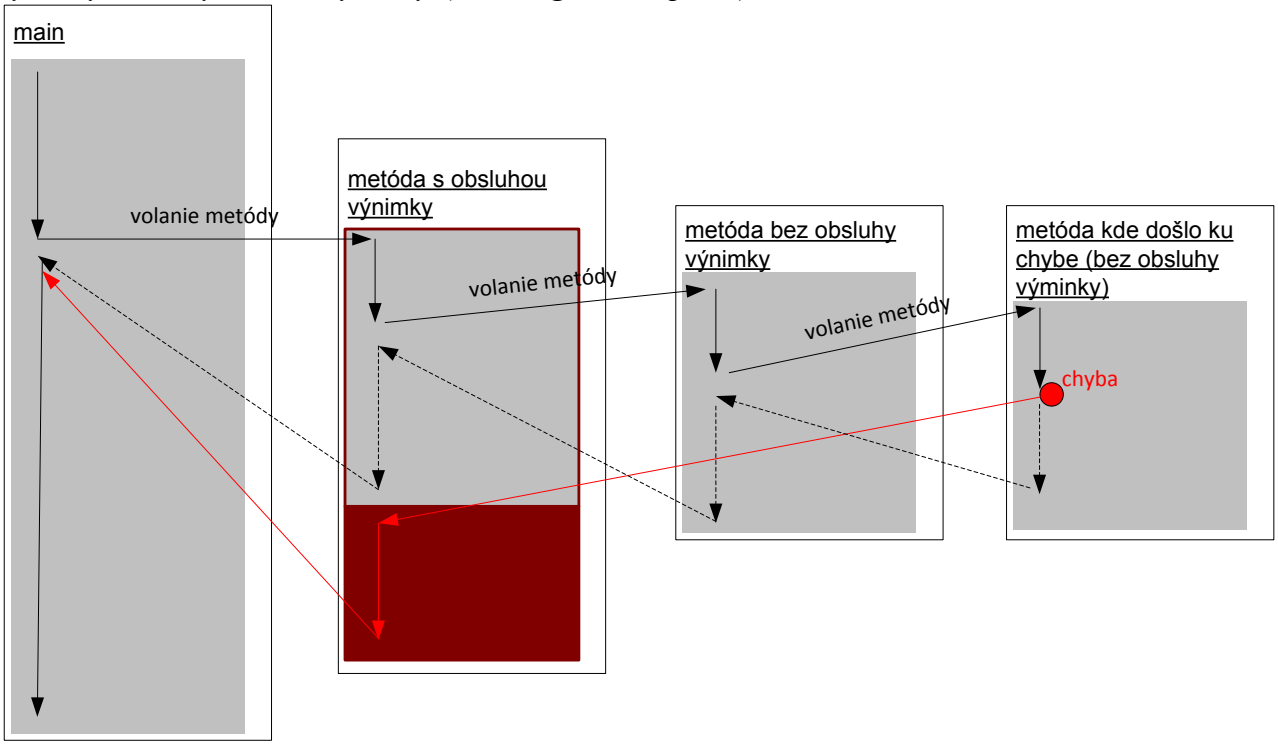
Výnimky (exceptions)

Výnimka (*exception*) je udalosť, ktorá narušuje normálny priebeh poradia vykonávaných inštrukcii programu. Používa sa pre obsluhu chýb.
Termín výnimka je skrátenie pre „výnimočná udalosť“ (exceptional event). Nesúvisí však s koncepciou udalosťami riadených programov.



Pri vzniku chyby je vytvorený *objekt výnimky*. Ten je predaný do runtime systému. Objekt výnimky obsahuje informácie o chybe.

Vytvorenie objektu výnimky a jeho predanie runtime systému sa označuje ako *spôsobenie výnimky* alebo *vyhodenie výnimky* (*throwing an exception*).



Po vyhodení výnimky runtime systém hľadá najbližší blok kódu určený pre spracovanie výnimky. Tento blok kódu sa nazýva *obsluha výnimky* (*exception handler*).
Každá obsluha je určená na spracovanie určitého typu výnimky – typ výnimky je daný typom objektu výnimky.
Hovoríme, že obsluha výnimky *zachytáva výnimku* (*catch exception*).
Zvyšok bloku kódu kde vznikla chyba sa vynechá.

Ak chyba vznikla v metóde ktorá neobsahuje príslušnú obsluhu výnimky, tak sa obsluha hľadá v niektorej aktuálne nadradenej metóde (z hľadiska volania metód). Na to sa využíva *zásobník volaní* (*call stack*).

Zachytenie a spracovanie výnimiek

```
try {
      //kód kde môže vzniknúť chyba
}
catch(TypVynimky1 objektVynimky){
      // obsluha výnimky typu (alebo podtypu) TypVynimky1
}
catch(TypVynimky2 objektVynimky){
      // obsluha výnimky typu (alebo podtypu) TypVynimky2
}
finally{
      //kód ktorý sa vykonaná vždy po bloku try (aj v prípade, že bloky catch nezachytia výminku)
}
```

Blok `try` – uzaviera časť kódu, kde môžu vzniknúť výnimky ktoré chceme zachytávať.

Za blokom `try` musí nasledovať blok `catch` (prípadne viac blokov `catch`), alebo blok `finally`, alebo obidva druhy blokov s ktorých musia byť prvé bloky `catch`.

Bloky `catch` – ku bloku `try` môže byť pridružený jeden, alebo viac blokov `catch`, ktoré slúžia na obsluhu výnimiek vyhodených z bloku `try`. Medzi koncom bloku `try` a začiatkom bloku `catch` nesmie byť žiadny kód. Každý blok `catch` slúži na obsluhu určitého typu výnimky. Typ výnimky je daný triedou. Trieda je priamym, alebo nepriamym potomkom triedy `Throwable`.

Ak sa v niektorej časti bloku `try` vyhodí výnimka, tak sa nepokračuje ďalej vo vykonávaní príkazov bloku `try`, ale začne sa vykonávať obsluha výnimky v bloku `catch`. Zvyšok bloku `try` sa už nevykoná. Po obsluhe výnimky sa pokračuje za blokmi `catch`.

Blok `finally` – Slúži na „upratanie/čistenie“. Príkazy v bloku `finally` sa vykonajú vždy po ukončení bloku `try`, bez ohľadu na to či bola vyhodená výnimka, alebo nie. A to aj v prípade ak sa v bloku `try` vykonal príkaz `return`, `continue`, alebo `break`.
Umiestnenie „upratovacieho“ kódu do bloku `finally` patrí medzi dobré programátorské postupy. Výhodou bloku `finally` je to, že sa vykoná aj v prípade, ak je vyhodená výnimka obslúžená v bloku `catch` umiestenom v inej metóde (nadradenej v zásobníku volaní).

Postup vykonávanie blokov:

1. blok `try` (dokonca, alebo po vyhodenie výnimky)
2. blok `cath` – vykoná sa ak v bloku `try` dôjde k vyhodeniu výnimky a typ objektu výnimky je rovnaký ako typ uvedený v bloku `catch`, alebo je podtypom typu uvedeného v bloku `catch`.
3. blok `finally` (vykoná sa vždy, ak je uvedený)
4. pokračuje sa vykonávaním kódu uvedenom za týmito blokmi

príklad:

```
System.out.println("zaciatok programu");
try{
    System.out.println("blok try (zaciatok)");
    //String retazec = "male pismena";
    String retazec = null;
    System.out.println(retazec.toUpperCase());//vyhodí výnimku
    System.out.println("blok try (dokonceny)");
}
catch(NullPointerException objektVynimky) {
    System.out.println("blok catch");
}
finally{
    System.out.println("blok finally");
}
System.out.println("dalsi kod programu");
```

výstup:

```
zaciatok programu
blok try (zaciatok)
blok catch
blok finally
dalsi kod programu
```

príklad (viac blokov `catch`, použitie objektu výnimky v obsluhe výnimky):

```
System.out.println("zaciatok programu");
try {
    System.out.println("blok try (zaciatok)");
    String retazec = "male pismena";
    //retazec = null;
    System.out.println(retazec.toUpperCase()); //teraz OK
    System.out.println(retazec.charAt(50));  //vyhodí výnimku
    System.out.println("blok try (dokonceny)");
}
catch (NullPointerException objektVynimky) {
    System.out.println("blok catch (nulovy pointer)");
    System.out.println("sprava: " + objektVynimky.getMessage());

}
catch (IndexOutOfBoundsException objektVynimky) {
    System.out.println("blok catch (index mimo rozsahu)");
    System.out.println("sprava: " + objektVynimky.getMessage());
}
finally {
    System.out.println("blok finally");
}
System.out.println("dalsi kod programu");
```

výstup:

```
zaciatok programu
blok try (zaciatok)
MALE PISMENA
blok catch (index mimo rozsahu)
sprava: String index out of range: 50
blok finally
dalsi kod programu
```

Ak je uvedených viacero blokov `catch` za sebou, obsluha výnimky sa hľadá v takom poradí v akom sú bloky `catch` uvedené.

príklad (vnorenie `try-catch-finally` v inom `try`, obsluha výnimky vyššie):

```
public static void main(String[] args) {
    try {
        System.out.println("blok try - vonkajsi (zaciatok)");
        String retazec = "male pismena";
        retazec = null;
        pracujSRetazcom(retazec);
        System.out.println("blok try - vonkajsi (dokonceny)");
    }
    catch (NullPointerException objektVynimky) {
        System.out.println("blok catch - vonkajsi (nulovy pointer)");
    }
    finally {
        System.out.println("blok finally - vonkajsi");
    }
    System.out.println("dalsi kod programu");
}

public static void pracujSRetazcom(String retazec) {
    try{
        System.out.println("blok try - vnutorny (zaciatok)");
        System.out.println(retazec.charAt(50));//NullPointerException
        System.out.println("blok try - vnutorny (dokonceny)");
    }
    catch (IndexOutOfBoundsException objektVynimky) {
        System.out.println("blok catch - vnutorny (index mimo rozsahu)");
    }
    finally {
        System.out.println("blok finally - vnutorny");
    }
    System.out.println("koniec metódy");
}
```

výstup:

```
blok try - vonkajsi (zaciatok)
blok try - vnutorny (zaciatok)
blok finally - vnutorny
blok catch - vonkajsi (nulovy pointer)
blok finally - vonkajsi
dalsi kod programu
```

príklad (blok `finally` a príkaz `return`):

```
public static void main(String[] args) {
  System.out.println("zaciatok programu");
  try{
    System.out.println("blok try (zaciatok)");
    String retazec = "male pismena";
    //retazec = null;
    System.out.println(retazec.toUpperCase());//nevyhodí výnimku
    System.out.println("blok try (dokonceny)");
    return;
  }
  catch(NullPointerException objektVynimky) {
    System.out.println("blok catch(NullPointerException)");
  }
  finally{
    System.out.println("blok final - tento kod sa vykona");
  }
  System.out.println("tento kod sa uz nevykona");
}
```

výstup:

```
zaciatok programu
blok try (zaciatok)
MALE PISMENA
blok try (dokonceny)
blok final - tento kod sa vykona
```

Blok `finally` sa vykonal aj keď bol pre ním uvedený `return`.

príklad (bez bloku `catch`):

```
try{
    System.out.println("blok try (zaciatok)");
    String retazec = null;
    System.out.println(retazec.toUpperCase());//vyhodí výnimku
    System.out.println("blok try (koniec)");
}
finally {
    System.out.println("blok finally");
}
```

Po vzniku výnimky sa vykoná blok `finally` a obsluha výnimky sa hľadá v nadradených blokoch `try-catch`, ktoré sú buď v tej istej metóde, alebo v nadradenej metóde z hľadiska volania metód.

Určenie výnimky spôsobenej metódou

Ako sme si ukázali, metódy nemusia obsluhovať všetky výnimky, ktoré v nich môžu vzniknúť. V prípade že v metóde môže vzniknúť chyba (následkom nej výnimka), ale používateľa metódy chceme iba informovať o chybe (vyhodením výnimky z metódy) a nechať ho aby rozhodol ako spracovať chybu (teda výnimku) v ním definovanom bloku `catch`, môžeme použiť *určenie výnimky* spôsobenej metódou. Vtedy pomocou kľúčového slova `thorws` vymenujeme zoznam výnimiek, ktoré môže metóda vyhodiť.

Nie všetky typy výnimiek musia byť takto určené. Ako bude uvedené neskôr, povinné je určovať iba kontrolované výnimky.

Príklad:

```
//zoznam výnimiek informuje používateľa metódy, že metóda môže vyhodiť
//výnimky typu NullPointerException a IndexOutOfBoundsException

public static void pracujSRetazcom(String retazec)
throws NullPointerException, IndexOutOfBoundsException {
    System.out.println("metoda - zaciatok");
    System.out.println(retazec.charAt(50)); //NullPointerException
    System.out.println("metoda - dokoncena");
}

public static void main(String[] args) {
    try {
        System.out.println("blok try (zaciatok)");
        String retazec = "male pismena";
        retazec = null;
        pracujSRetazcom(retazec);
        System.out.println("blok try (dokonceny)");
    }
    catch (NullPointerException objektVynimky) {
        System.out.println("blok catch (NullPointerException)");
    }
    catch (IndexOutOfBoundsException objektVynimky) {
        System.out.println("blok catch (IndexOutOfBoundsException)");
    }
    //blok final nie je povinný
}
```

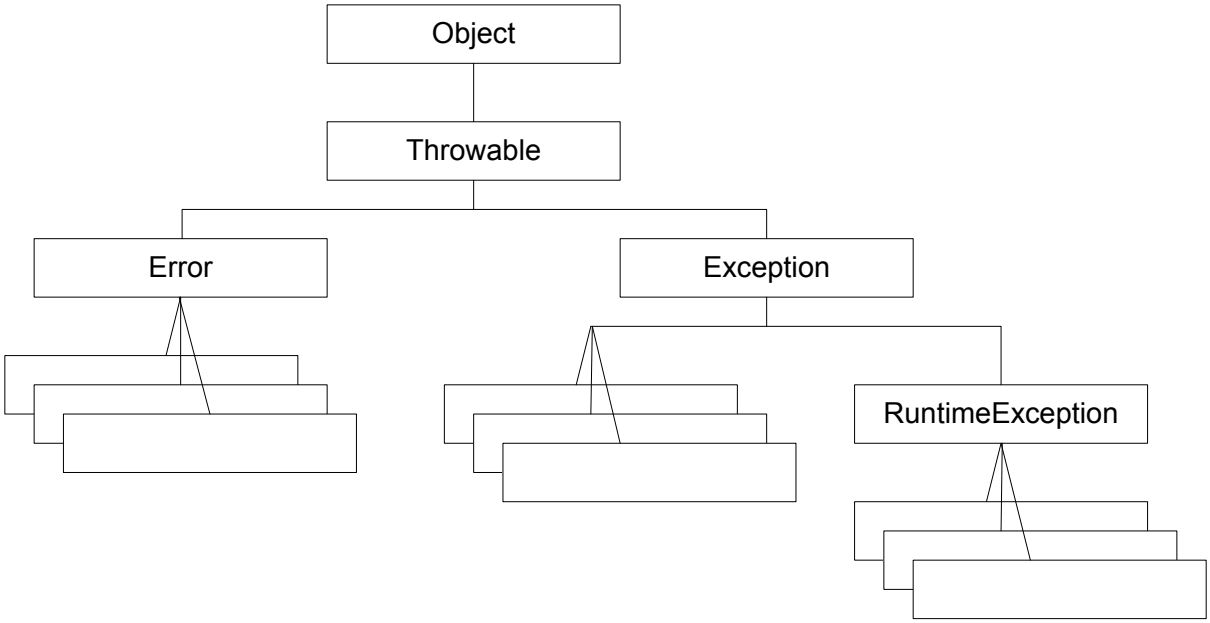
Tento príklad nie je celkom dobrý pretože: Výnimky typu `NullPointerException` a `IndexOutOfBoundsException` sú nekontrolované výnimky, preto ich v tomto príklade nie je nutné uvádzať za kľúčovým slovom `throws`.

výstup:

```
blok try (zaciatok)
metoda - zaciatok
blok catch (NullPointerException)
```

Trieda Throwable a jej podtriedy

Typ výnimky je daný triedou objektu výnimky. Všetky triedy výnimiek sú priamymi, alebo nepriamymi potomkami triedy `Throwable (java.lang.Throwable)`.



Trieda `Error`
Ak dôjde k chybe dynamického prepojenia, alebo inej závažnej chybe modulu JVM, spôsobí modul výnimku typu `Error`.
Jednoduché programy bežne nezachytávajú, ani nespôsobujú výnimky typu `Error`.

Trieda `Exception`
Väčšina programov vyvoláva a zachytáva objekty odvodené od triedy `Exception`. Objekt typu `Exception` oznamuje, že došlo k problému, ale nejde o závažný systémový problém.
Príkady:

- `IllegalAccessException` – nepodarilo sa nájsť konkrétnu metódu
- `NegativeArraySizeException` – pokus o vytvorenie poľa zo zápornou veľkosťou

Trieda `RuntimeException`
Je vyhradená pre výnimky oznamujúce nesprávne použitie rozhrania API.
Príklady:

- `NullPointerException` – nastáva pri pokuse o prístup k objektu pomocou referencie s hodnotou `null`
- `IndexOutOfBoundsException` – index mimo rozsahu

Zachytiť alebo určiť požiadavku (catch or specify requirement)

Kód ktorý by mohol spôsobiť výnimku, je niekedy nutné uzavrieť jedným z dvoch spôsobov:
- uzavrieť do bloku `try`, za ktorým nasleduje obsluha výnimky
- uzavrieť do metódy, ktorá uvádza (pomocou `throws`), že môže spôsobiť výnimku

Toto pravidlo sa nazýva „*zachytiť, alebo určiť*“. Niektoré výnimky musia splňovať toto pravidlo, iné ho splňovať nemusia.

Tri druhy výnimiek

1) *kontrolovaná výnimka* (*checked exception*) – výnimočné stavy, s ktorými by sa mala aplikácia vedieť vyrovnať (napr. práca s I/O)
Kontrolované výnimky podliehajú požiadavke „zachytiť, alebo určiť“. Medzi kontrolované výnimky patria všetky výnimky okrem tých, ktoré sú inštanciami tried `Error`, `RuntimeException`, alebo ich podtriedami.

2) *chyba* (*error*) – tieto výnimočné stavy majú obvykle príčinu mimo aplikácie. Aplikácia ich obvykle nemôže predpokladať a vyriešiť.
Napr. aplikácia úspešne otvorí súbor na čítanie, ale nemôže ho čítať kvôli poruche hárdveru, alebo poruche systému.
Programátor môže, ale nemusí takúto výnimku zachytávať.
Na tento druh výnimiek sa požiadavka „zachytiť, alebo určiť“ nevzťahuje.
Tieto výnimky sú označené triedou `Error` a jej podtriedami.

3) *výnimka za behu* (*runtime exception*) – tieto výnimočné stavy majú príčinu v aplikácii a aplikácia ich obvykle nemôže predpokladať a vyriešiť. Obvykle znamenajú programátorskú chybu. Napríklad logické chyby, alebo nesprávne použitie rozhrania API.
Napríklad ak je na niektorom mieste programu očakávaná „nenulová“ referencia na objekt, ale pri vykonávaní má táto referencia hodnotu `null`.
Aplikácia môže zachytiť túto výnimku, ale lepšie je odstrániť chybu v kóde programu, kvôli ktorej k výnimke došlo.
Tieto výnimky nepodliehajú požiadavke „zachytiť, alebo určiť“.
Výnimky sú označené triedou `RuntimeException`, alebo jej podtriedami.

Nekontrolované výnimky (*unchecked exception*) – spoločné označenie pre tieto druhy výnimiek: „chyba“ a „výnimka za behu“.

Ako spôsobiť (vyhodiť) výnimku

Výnimky sa vyhadzujú príkazom `throw`. Príkaz `throw` vyžaduje jediný argument: objekt ktorý možno vyhodiť. Tieto objekty sú inštanciami ľubovoľnej podtriedy triedy `Throwable`.
Príklad (spôsobenia výnimky):

```
throw objektVynimky;
```

Príklad (vytvorenie vlastnej výnimky, spôsobenie výnimky):
Definujeme triedu reprezentujúcu zásobník a výnimky reprezentujúce chyby pri práci so zásobníkom.
Väčšinou budeme vytvárať kontrolované výnimky, teda dediť od triedy `Exception`. Názov typu výnimky sa v tomto prípade končí "Exception" (konvencia). Trieda `Exception` definuje správu obsahujúcu textový reťazec s popisom chyby. Túto správu môžeme nastaviť napr. pomocou parametra konštruktora a získať volaním metódy `getMessage()`.

```
//Pre všetky výnimky definujeme spoločnú nadtriedu, aby sme mohli spoločne narábať s výnimkami
//zásobníka.
public abstract class AbstractStackException extends Exception {
    public AbstractStackException(String message) {
        super(message); //správu s popisom chyby predáme konštruktoru nadtriedy
    }
}
```

```
//Chybne zadaná kapacita zásobníka (pri vytváraní)
public class CapacityStackException extends AbstractStackException {
    public CapacityStackException(int capacity) {
        super("chybne udana kapacita (" + capacity + ")");
    }
}
```

```
//Chyba pri pokuse o výber hodnoty z prázdneho zásobníka
public class EmptyStackException extends AbstractStackException {
    public EmptyStackException() {
        super("prazdny zasobnik");
    }
}
```

```
//Chyba pri vkladaní hodnoty do plného zásobníka
public class FullStackException extends AbstractStackException {
    public FullStackException(double value, int capacity) {
        super("zasobnik plny (hodnota = " + value
              + ", kapacita = " + capacity + ")");
    }
}
```

```
public class Stack {
    private double[] data;
    private int counter;

    public Stack(int capacity) throws CapacityStackException {
        if( capacity <= 0 ) {
            throw new CapacityStackException(capacity);
        }
        data = new double[capacity];
        counter = 0;
    }

    public double pop() throws EmptyStackException {
        if( counter == 0 ) {
            throw new EmptyStackException();
        }
        counter --;
        return data[counter];
    }

   public void push(double newData)throws FullStackException {
        if( counter == data.length ) {
            throw new FullStackException(newData, data.length);
        }
        data[counter] = newData;
        counter ++;
    }
}
```

príklad obsluhy výnimiek (každá výnimka obslúžená zvlášť):

```
public static void main(String[] args) {
    try {
        Stack stack1 = new Stack(2);
        Stack stack2 = new Stack(-10); //výnimka
        stack1.push(1);
        stack1.push(2);
        stack1.push(3); //výnimka
        stack1.pop();
        stack1.pop();
        stack1.pop(); //výnimka
    } catch (CapacityStackException exception) {
        System.err.println(exception.getMessage());
    } catch (EmptyStackException exception) {
        System.err.println(exception.getMessage());
    } catch (FullStackException exception) {
        System.err.println(exception.getMessage());
    }
}
```

príklad obsluhy výnimiek

- blok `catch` pre typ `AbstractStackException` zachytí všetky výnimky typu `AbstractStackException` alebo výnimky podtried triedy `AbstractStackException`
- výnimky typu `FullStackException` budu ale zachytené v prvom bloku `catch`, pretože je umiestnený pred blokom zachytávajúcim typ `AbstractStackException`. Výnimky typu `FullStackException` už nebudú obslúžené v druhom bloku

```
public static void main(String[] args) {
    try {
        Stack stack1 = new Stack(2);
        Stack stack2 = new Stack(-10); //výnimka
        stack1.push(1);
        stack1.push(2);
        stack1.push(3); //výnimka
        stack1.pop();
        stack1.pop();
        stack1.pop(); //výnimka
    } catch (FullStackException exception) {
        //zachytenie výnimky typu FullStackException
        System.err.println("full: " + exception.getMessage());
    } catch (AbstractStackException exception ) {
        //zachytenie výnimiek typu
        // - CapacityStackException
        // - EmptyStackException
        System.err.println("vseobecne: " + exception.getMessage());
    }
}
```

príklad obsluhy rôznych typov výnimiek v jednom bloku `catch` (využitie operátora |)

```
public static void main(String[] args) {
    try {
        Stack stack1 = new Stack(2);
        Stack stack2 = new Stack(-10); //výnimka
        stack1.push(1);
        stack1.push(2);
        stack1.push(3); //výnimka
        stack1.pop();
        stack1.pop();
        stack1.pop(); //výnimka
    } catch (CapacityStackException exception) {
        //zachytenie CapacityStackException
        System.err.println("vytvaranie zasobnika: "
                           + exception.getMessage());
    } catch (EmptyStackException | FullStackException exception) {
        //zachytenie EmptyStackException alebo FullStackException
        System.err.println("praca so zasobnikom: "
                           + exception.getMessage());
    }
}
```

Ak `catch` blok zachytáva viac typov výnimiek, jeho parameter je implicitne `final`.

Zreťazené výnimky

Aplikácie často reagujú na výnimku tak, že spôsobia ďalšiu výnimku (iného typu)
príklad:

```
try {
    // .....
}
catch (LowLevelException exeption) {
    //pretransformuje nízkoúrovňovú chybu na vysokoúrovňovú
    throw new HighLevelException("detailna sprava", exception);
}
```

Trieda `Throwable` obsahuje konštruktory a metódy podporujúce zreťazené výnimky:

```
Throwable(String, Throwable)
Throwable(Throwable)
Throwable getCause()
Throwable initCause(Throwable)
```

Argument konštruktorov a metódy `initCause`() (typu `Throwable`) predstavuje výnimku, ktorá je príčinou aktuálnej výnimky. Metóda `getCause`() vráti výnimku, ktorá spôsobila aktuálnu výnimku.

Try-with-resource a rozhranie `AutoCloseable`

Do bloku `finally` sa umiestňuje kód, ktorý treba vykonať bez ohľadu na to, či výnimka vznikla, alebo nie. Častým príkladom je práca so systémovými zdrojmi (napr. súbormi), ktorá je potenciálnym zdrojom výnimiek. Kód pracujúci so systémovými zdrojmi zvykne byť umiestnený v bloku `try`, uzatvorenie systémových zdrojov (napr. súborov) v bloku `finally`.
Pre zjednodušenie práce existuje v jave ďalšia možnosť definovania bloku `try` (*try-with-resources*), vhodná pre prácu so systémovými zdrojmi, ktoré implementujú rozhranie `AutoCloseable`. Toto rozhranie definuje metódu `close()`, ktorá sa automaticky vykoná pri ukončení bloku `try`.

Príklad (čítanie znakov zo súboru a ich posielanie cez sieť)

CitacSuboru.java

```
public class CitacSuboru implements AutoCloseable {

    public CitacSuboru(String subor) throws FileNotFoundException{
        //otvorenie súboru
    }

    public char citajDalsi() throws IOException {
        //prečítanie a vrátenie znaku zo súboru
    }

    @Override
    public void close() throws IOException {
        //zatvorenie súboru
    }
}
```

SietovyVysielac.java

```
public class SietovyVysielac implements AutoCloseable{

    public SietovyVysielac(String sietovaAdresa) throws IOException{
        //otvorenie komunikácie
    }

    public void posli(char znak) throws IOException {
        //poslanie znaku
    }

    @Override
    public void close() throws IOException {
        //uzatvorenie komunikácie
    }
}
```

Použitie – jednoduchý `try`

```
try {
    CitacSuboru subor = null;
    SietovyVysielac vysielac = null;
    try {
        subor = new CitacSuboru("cesta_k_suboru");
        vysielac = new SietovyVysielac("adresa_prijemcu");

        //čítanie súboru a posielanie údajov cez sieť
    }
    finally {
        if (subor != null) {
            subor.close(); //metóda close zvyčaje môže vyhodiť výnimku
        }
        if (vysielac != null) {
            vysielac.close();//metóda close zvyčaje môže vyhodiť výnimku
        }
    }
}
catch .....
```

Použitie – try-with-resource

```
try (
    CitacSuboru subor = new CitacSuboru("cesta_k_suboru");
    SietovyVysielac vysielac=new SietovyVysielac("adresa_prijemcu");
)
{
    //čítanie súboru a posielanie údajov cez sieť
}
//metódy close sa vykonajú automaticky
catch .....
```

Použitie kontrolovaných a nekontrolovaných výnimiek

- Pri vytváraní vlastných typov výnimiek je väčšinou vhodné, aby tieto výnimky boli kontrolované.
- Uvádzanie výnimiek, ktoré môže metóda spôsobiť je v popise (dokumentácii) metódy rovnako dôležité ako vstupné parametre a návratová hodnota. To sa týka hlavne kontrolovaných výnimiek.
- Výnimky za behu (napr. aritmetické výnimky, prístup k členu cez `null`) môžu nastať na ľubovoľnom mieste programu a v typickom programe ich môže byť veľa. Sú následkom programátorských chýb. Kvôli prehľadnosti sa preto výnimky za behu (runtime exception) nepridávajú do deklarácii metód.

Výhoda použitia výnimiek

- prehľadnosť
- oddelenie logiky hlavného programu od spracovania chýb
- ak je spracovanie chyby v niektorej nadradenej metóde v zásobníku volaní, netreba prenášať informáciu o chybe cez návratové hodnoty, alebo cez argumenty. Informácie sa prenášajú v objekte výnimky
- možnosť informovať o chybe počas vykonávania konštruktora (konštruktor môže podobne ako metóda tiež spôsobiť výnimku)