

Výjimky



A0B36PR2-Programování 2
Fakulta elektrotechnická
České vysoké učení technické

Výjimky

Obsah

- Pojem výjimky
- Princip mechanismu zpracování výjimek, **try + catch**
- Kompletní zpracování výjimek
- Vyhození výjimky, propagace výjimek
- *Generování vlastní výjimky*
- *Hierarchie výjimek*
- Blok **finally**

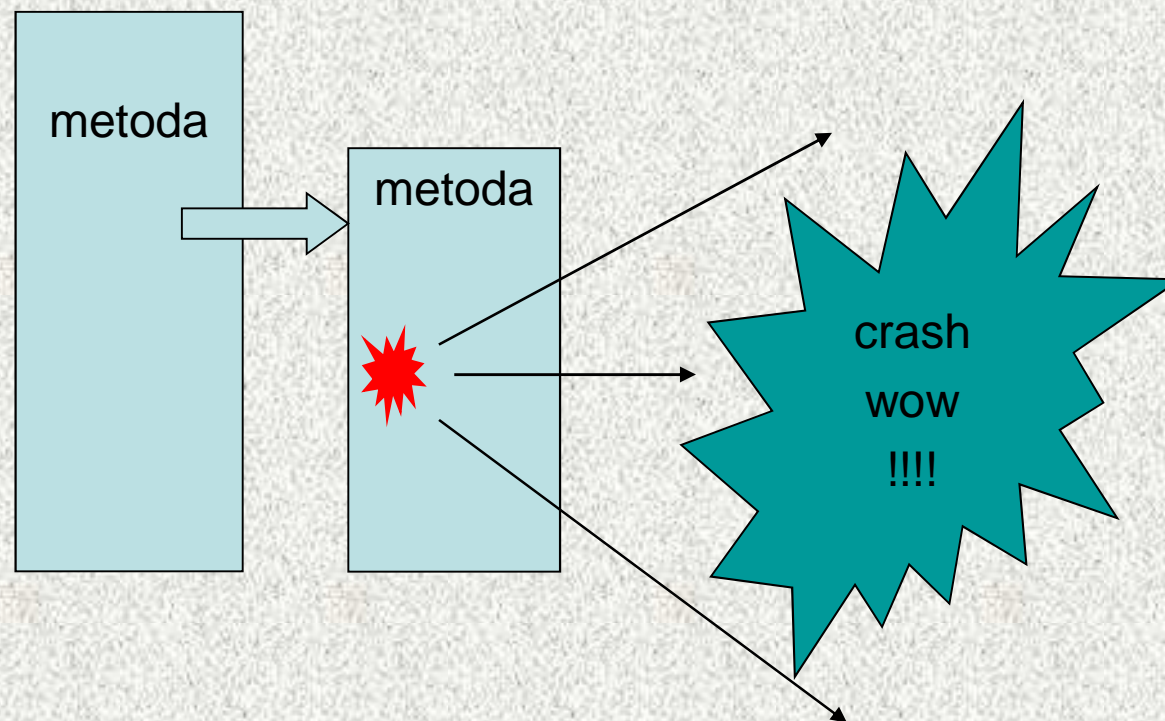
Výjimky

- Vyjímka je „nestandardní situace“ za běhu programu:
 1. Situace, které jsou nestandardní či **které my považujeme za nestandardní**, měli bychom reagovat a můžeme a dokážeme reagovat (**RuntimeException**), např. :
 - *Pokus o čtení z prázdného zásobníku*
`EmptyStackException`
 - *Dělení nulou, indexování mimo rozsah pole, špatný formát čísel*
`ArithmeticException`, `NumberFormatException`
 2. Situace, **na které musíme reagovat**, Java nás přinutí (**Exception**, `IOException`)
 - *Odkaz na chybějící soubor*
`FileNotFoundException`
 3. Chyba v hardware, závažné chyby, **nemůžeme reagovat** (**Error**)
 - *Chyba v JVM, HW chyba*
(`OutOfMemoryError`, [UnknownError](#))

Výjimky

- **Výjimka je nestandardní situace při výpočtu programu**, např.:
 - *indexace mimo meze pole, neexistující soubor, dělení nulou, málo paměti, nedovolená reakce uživatele*, apod., která znemožňuje pokračovat řádným způsobem.
- Obecně chyba vzniká při **porušení sémantických omezení** jazyka Java
- **Bezpečnostní prvek** Javy:
 - zpracování chyb a nestandardních stavů není ponecháno jen na vůli programátora!!
- **Reakce na očekávané chyby se vynucuje** na úrovni překladu, při nerespektování se překlad nepodaří
- Chyba při provádění programu v jazyku Java nemusí znamenat ukončení programu – chybu lze ošetřit a pokračovat dál
- Při vzniku výjimky je automaticky vytvořen **objekt**, který nese informace o vzniklé výjimce.
- Mechanismus výjimek umožní přenést řízení z místa, kde výjimka vznikla do místa, kde bude zpracována
 - Oddělení "výkonné" části (**try**) od části "chybové" - **catch**

Výjimky – vznik nestandardní situace



Neošetření výjimky

```
import java.util.Scanner;
import java.io.*;

public class XVyjimka {
    public static void main(String[] args)
        throws IOException {
        Scanner sc = new Scanner(new File("cisla.txt"));

        System.out.print("Precte cislo ze souboru");
        int i = sc.nextInt();
        System.out.println("Cislo je: " + i);
    }
}
```

Ošetření výjimky

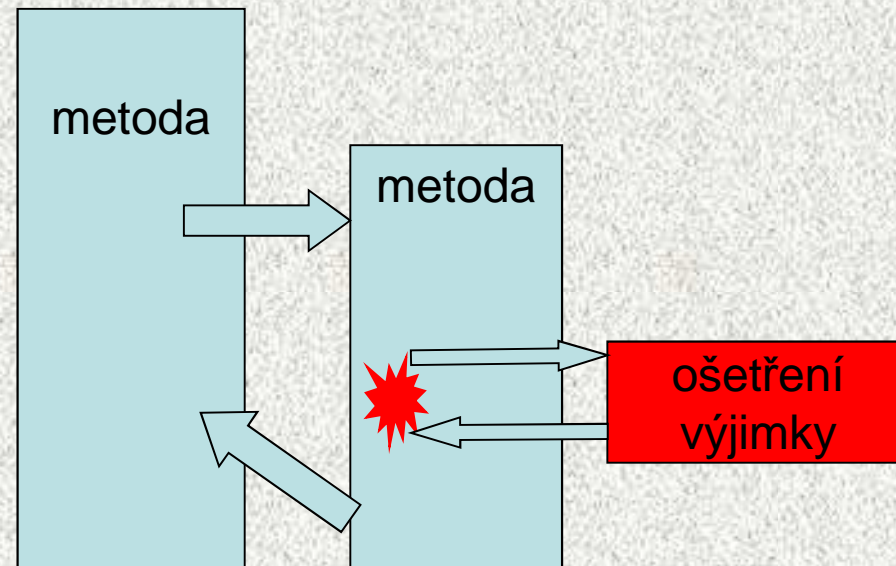
```
import java.util.Scanner;

import java.io.*;

public class XVyjimka_Osetreni {

    public static void main(String[] args) throws Exception {
        try {
            String s = "cisla1.txt";
            Scanner sc = new Scanner(new File(s));
            System.out.print("Precte cislo ze souboru " + s);
            int i = sc.nextInt();
            System.out.println(" Cislo je: " + i);
        } catch (IOException e) {
            System.out.println("Cislo je: " + 100002);
        }
    }
}
```

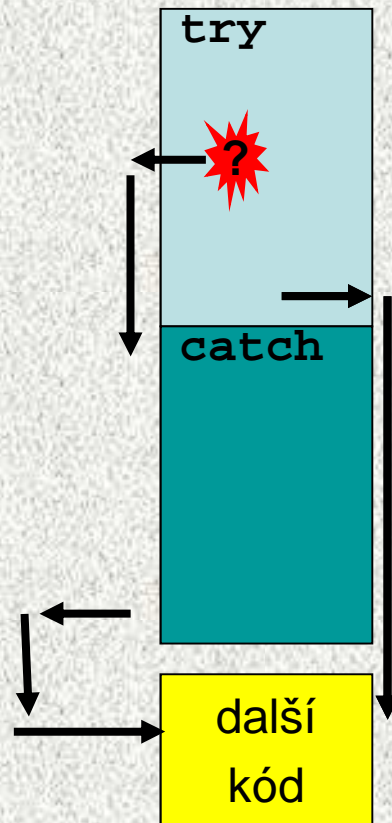
Výjimky – ideální ošetření



Mechanismus šíření výjimek v jazyku Java

- Jestliže vznikne výjimka, potom *JVM* hledá odpovídající řešení, které je schopné výjimku ošetřit (tj. převzít řízení):
 - pokud výjimka vznikla v bloku příkazu `try`, hledá se odpovídající klauzule `catch` v tomto příkazu, další příkazy bloku `try` se neprovedou a řízení se předá konstrukci ošetřující výjimku daného typu do místa ošetření výjimky (tzv. handler)
 - pokud výjimka vznikne mimo příkaz `try`, předá se řízení do místa volání metody a pokračuje se podle předchozího bodu
 - pokud taková konstrukce v těle funkce (metody, konstruktoru) není, skončí funkce nestandardně a výjimka se šíří na dynamicky nadřazenou úroveň
 - není-li výjimka ošetřena ani ve funkci `main`, vypíše se a program skončí
 - pro rozlišení různých typů výjimek je v jazyku Java zavedena řada knihovných tříd, výjimky jsou instancemi těchto tříd

Bloky try a catch

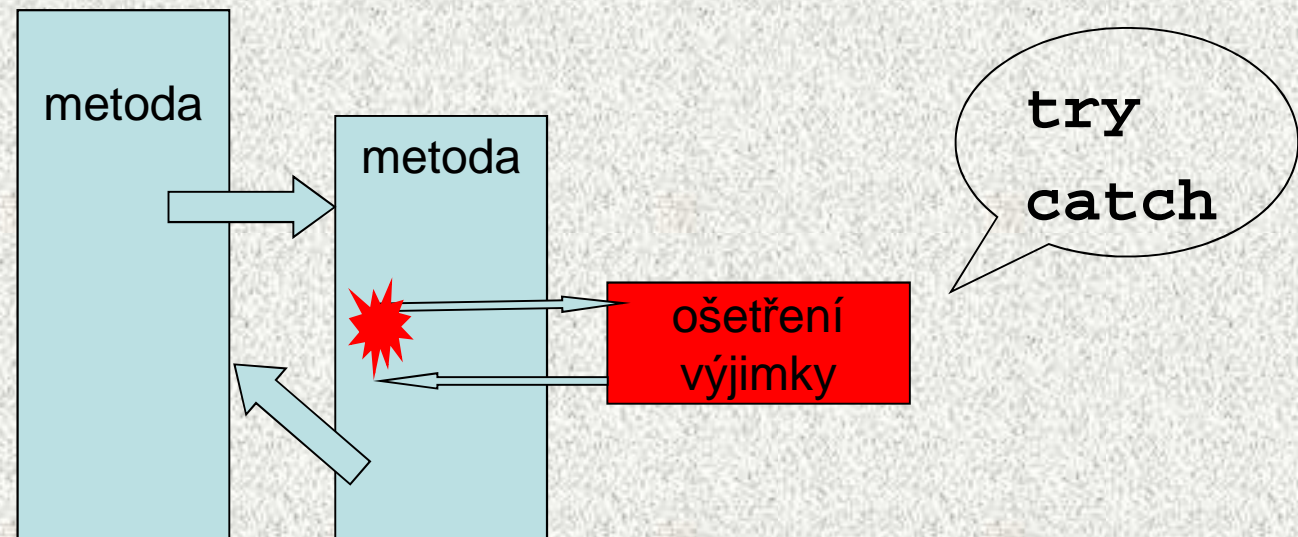


Výjimky, způsoby ošetření

1. Bez ošetření vyjímky - **CHYBA**
2. Ošetření předáme výše, výjimku nelze či nechceme ošetřit
3. Zachytíme, (částečně) ošetříme a mimo to ji předáme výše
4. Zachytíme a kompletně ošetříme

Poznámka: Nevíme-li dopředu, jak uživatel chce naložit se vzniklou výjimkou, snažíme se použít poslední možnost

Výjimka – zachytíme a kompletně ošetříme



Kompletní ošetření výjimky

- Při vzniku výjimky se řízení (provádění) programu přenesse z místa vzniku výjimky do místa ošetření výjimky. Místo ošetření výjimky (tzv. handler) je klauzule `catch` příslušná příkazu `try`:

```
class XVyjimka0 {  
    void m(int[] pole) {  
        try {  
            int i = 10;  
            pole[i] = 0;  
        } catch (ArrayIndexOutOfBoundsException ioe) {  
            System.out.println("Ignoruji prekroceni...");  
        }  
    }  
    public static void main(String[] args) {  
        int[] pole={1,2,3};  
        new XVyjimka0().m(pole);  
        System.out.print("a pokračuji\n");  
    }  
}
```

Ignoruji prekroceni...
a pokračuji

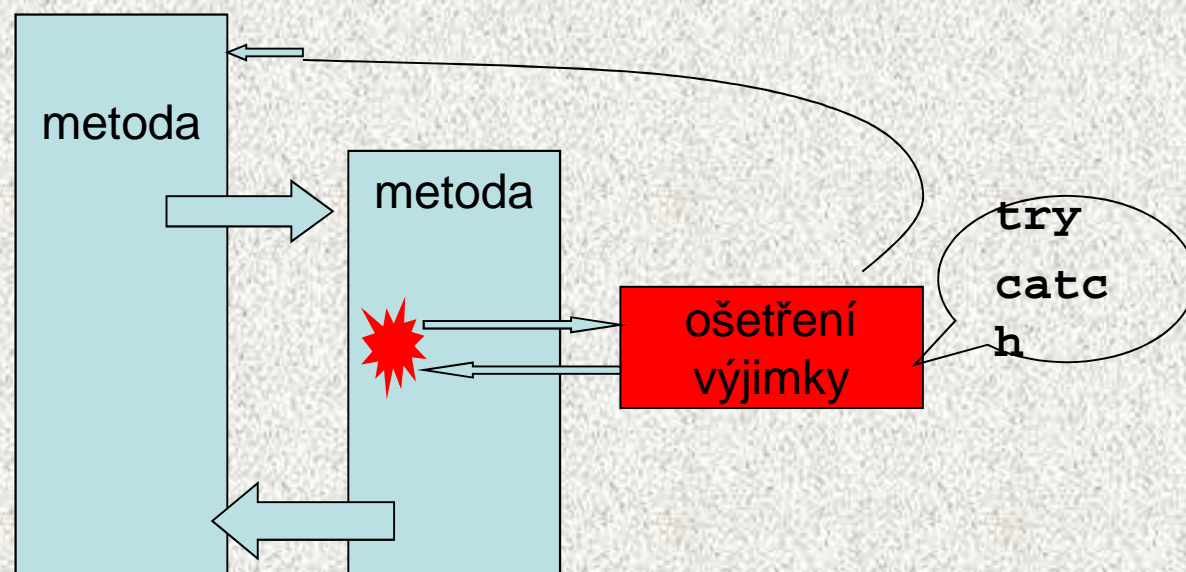
Kompletní ošetření výjimky

- Řešení jazykovou konstrukcí **try-catch**

Jedná se o systém:

- průchodu kritickou částí - tzv. chráněný úsek - **try**
- s případnou následnou reakcí na vzniklou výjimečnou událost v bloku (**catch**), označovaný jako *handler*
- klauzule **catch** odpovídá výjimce tehdy, pokud je objekt výjimky kompatibilní pro přiřazení s parametrem **catch** klauzule
- klauzulí **catch** může být i více pro různé typy výjimek

Ošetření výjimky a předání výš



Ošetření výjimky a předání výš, nevhodně!

```
public class XVyjimka1 {  
  
    public static int XctiInt() { // ošetření ve funkci  
        try {  
            Scanner sc = new Scanner(System.in);  
            int i = sc.nextInt();  
            return i;  
        } catch (Exception e) {  
            System.out.println("Chyba v udaji");  
            return -1;  
        }  
    }  
}
```


Ošetření výjimky a předání výš, nevhodně!

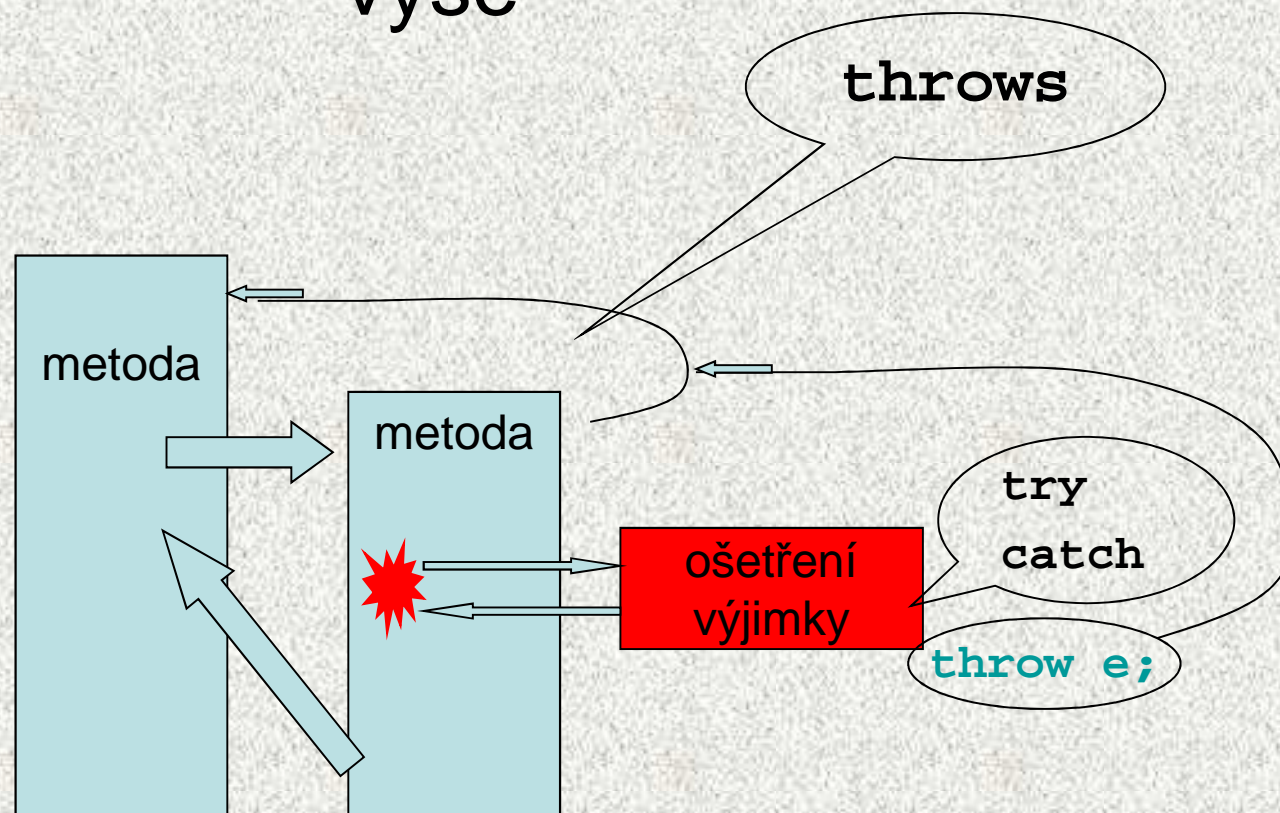
```
public static void main(String[] args) {  
    for (int j = 1; j < 3; j++) {  
        System.out.println("\nZadej pocet cihel: ");  
        int i = XctiInt();  
        if (i != -1) {  
            System.out.println("Cihel je: " + i);  
        } else {  
            System.out.println("Počet cihel neudán ");  
        }  
    }  
}
```

Zadej počet cihel: č
Chyba v údaji
Počet cihel neudán

Poznámka

- Ve volající funkci `main` se testuje "chybový" kód zaslaný zachycenou výjimkou
 - Nekonzistentní řešení, lépe pomocí příkazu `throw`
 - Způsob řešení typu „C“

Výjimka – zachytíme, ošetříme + předáme výše



Ošetření výjimky a předání výš, throw

```
public class Xvyjimka2 {  
  
    public static int XctiInt() throws Exception {  
        try {  
            Scanner sc = new Scanner(System.in);  
            int i = sc.nextInt();  
            return i;  
        } catch (Exception e) {  
            System.out.println("Chyba v udaji");  
            throw e;  
        }  
    }  
}
```

Ošetření výjimky a předání výš, throw

```
public static void main(String[] args) {  
    for (int j = 1; j < 3; j++) {  
        try {  
            System.out.println("\nZadej pocet cihel: ");  
            int i = XctiInt();  
            System.out.println("Cihel je: " + i);  
        } catch (Exception e) {  
            System.out.println("Počet cihel neudán ");  
        }  
    }  
}
```

Zadej pocet cihel:
r
Chyba v udaji
Počet cihel neudán

Ošetření výjimky a předání výš, `throw` - komentář

Zpracování výjimky:

1. Odchycení pomocí `catch (Exception e)` v `cXctiInt()`
2. Ošetření: `System.out.println("Chyba v udaji");`
3. Předání výše pomocí `throw e;` (`e` typu `Exception`)
4. Odchycení ve funkci `main` pomocí `catch(Exception e)`
5. Ošetření:

```
System.out.println(„Počet cihel neudan “);
```

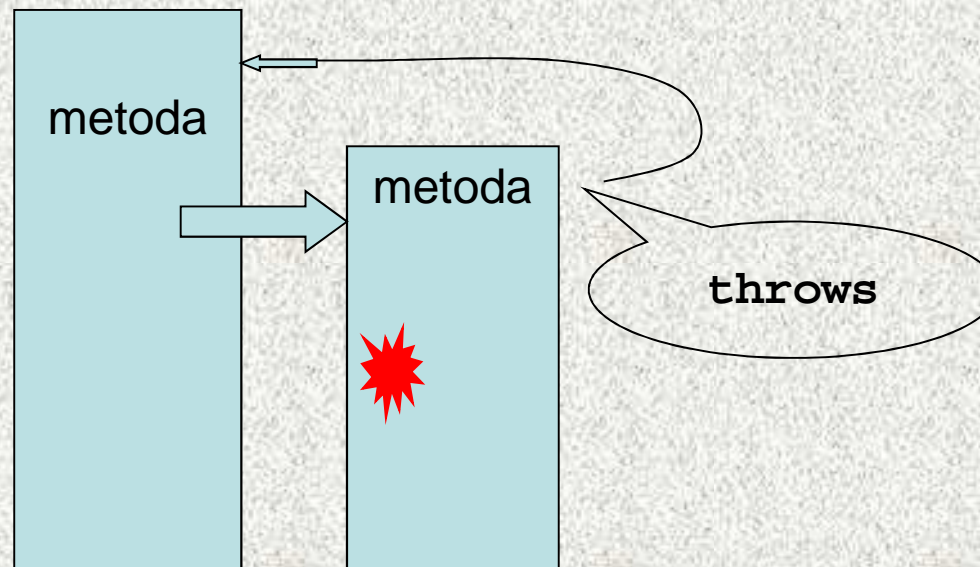
Funkce `main` by mohla výjimku předat výše:

```
public static void main(String[] args) throws Exception {}
```

Proměnná `e` funguje jako formální parametr metody:

```
catch (IOException e) {  
    System.out.println("Chyba v udaji");  
    throw e; // vyhození výjimky  
}
```

Neošetření výjimky – předáním výše



Neošetření výjimky – předáním výše

- Předání výjimky do volající metody - tzv. **propagace výjimek**, šíření výjimek v případě "neošetření" výjimky, "zřeknutí se odpovědnosti" a předání klauzulí **throws**
- Nejjednodušší řešení, žádný zásah do kódu doplněním hlavičky metody
`public static int ctiInt() throws IOException {`
- Podobně i metoda `main`, Výjimku pak řeší JVM
`public static void main(String[] args) throws IOException {`

Nevýhoda:

Nutíme uživatele metody se postarat o "vyhozenou" výjimku

Oprávnění použití:

Závisí-li chod dalšího programu na korektní funkci metody, nemá cenu ji ošetřovat a přitom by činnost programu stejně nemohla pokračovat (prostě tu hodnotu musíme mít a ne jen "nespadnutý" program!)

Neošetření výjimky – předáním výše

- Java sama ohlásí při překladu, které části jsou kritické a je je třeba ošetřit, nejsou-li, pak minimálně "vyhozením" na vyšší úroveň pomocí klauzule `throws`, jinak nedojde k překladu.

Neošetření výjimky – předáním výše

```
public class XVyjimka1NE {
public static int XctiInt() throws Exception {
    Scanner sc = new Scanner(System.in);
    int i = sc.nextInt();
    return i;
}
public static void main(String[] args) {
    for (int j = 1; j < 3; j++) {
        try {
            System.out.println("\nZadej pocet cihel: ");
            int i = XctiInt();
            System.out.println("Cihel je: " + i);
        } catch (Exception e) {
            System.out.println("Počet cihel neudán ");
        }
    }
}
}
```

```
Zadej pocet cihel:
r
Počet cihel neudán
```

Vygenerování vlastní výjimky

- Ve vlastních třídách může nastat stav, který chceme ošetřit standardně výjimečným stavem
- Vlastní výjimka je potomkem třídy **Exception**:
 - jedná se o tzv. **synchronní výjimku**, vzniká na přesně definovaném místě
 - většinou se jedná o výjimku, na kterou **chceme** reakci uživatele

Poznámka:

Není doporučené používat vyhození výjimky jako náhrady break (čas!)

Vygenerování výjimky - příklad

```
public class XVyjimka3 {
    public static int XctiInt() throws Exception {
        try {
            Scanner sc = new Scanner(System.in);
            int i = sc.nextInt();
            if (i == 0) {
                throw new Exception(); // generujeme vyjimku
            }
            return i;
        } catch (Exception e) {
            System.out.println("Chyba v udaji");
            throw e;
        }
    }
}
```

Vygenerování výjimky – příklad, dokončení

```
for (int j = 1; j < 3; j++) {  
    try {  
        System.out.println("\nZadej pocet cihel: ");  
        int i = XctiInt();  
        System.out.println("Cihel je: " + i);  
    } catch (Exception e) {  
        System.out.println("Počet cihel neudán ");  
    }  
}
```

Zadej pocet cihel:
r
Chyba v udaji
Počet cihel neudán

Zadej pocet cihel:
0
Chyba v udaji
Počet cihel neudán

Vytvoření a vygenerování vlastní výjimky

- V praktickém programování potřebujeme ošetřit mimořádné situace formou výjimek
- Třídy výjimek vzniknou děděním od třídy **Exception**
- Vznik výjimečné situace bude charakterizován generováním instance objektu třídy výjimky,
- Reakci na vlastní výjimky systém vyžaduje

Vytvoření a vygenerování vlastní výjimky

```
class MojeVyjimka extends Exception {
    int n;    // deklarace třídy vyjimky
    int d;
    MojeVyjimka(int i, int j) {
        /* super("Hodnota " + n + "/" + d + " není integer."); */
        n = i;
        d = j;
    }

    public String toString() {
        return "Hodnota " + n + "/" + d + " není integer.";
    }
}
```

Vygenerování vlastní výjimky, příklad II

```
public class CustomExceptDemol {
    public static void main(String args[]) {
        int numer[] = { 4, 8, 15, 32, 64, 127, 256, 512 };
        int denom[] = { 2, 0, 4, 4, 0, 8 };
        for(int i=0; i<numer.length; i++) {
            try {
                System.out.print((i+1) + ". ");
                if((numer[i] % denom[i]) != 0)
                    throw new MojeVyjimka(numer[i], denom[i]);
                System.out.println(numer[i] + "/" +
                                    denom[i] + " je " +
                                    numer[i]/denom[i]);
            }
        }
    }
}
```

Vygenerování vlastní výjimky, příklad III

```
catch (ArithmeticException exc) {  
    System.out.println("Nelze dělit nulou!");  
}  
catch (ArrayIndexOutOfBoundsException exc) {  
    System.out.println("Mimo rozsah");  
}  
catch (MojeVyjimka exc) { // ošetření  
    System.out.println(exc); // vlastní výjimka  
}  
}  
}  
}
```


Vygenerování vlastní výjimky, příklad, výsledek

Výsledek činnosti:

1. $4/2$ je 2
2. Nelze dělit nulou!
3. Hodnota $15/4$ není integer.
4. $32/4$ je 8
5. Nelze dělit nulou!
6. Hodnota $127/8$ není integer.
7. Mimo rozsah
8. Mimo rozsah

Ošetřování výjimek, které "nikdy nenastanou"

- **Nešvar:** nevyplnění těla `catch`:

```
catch (Exception e) {  
}
```

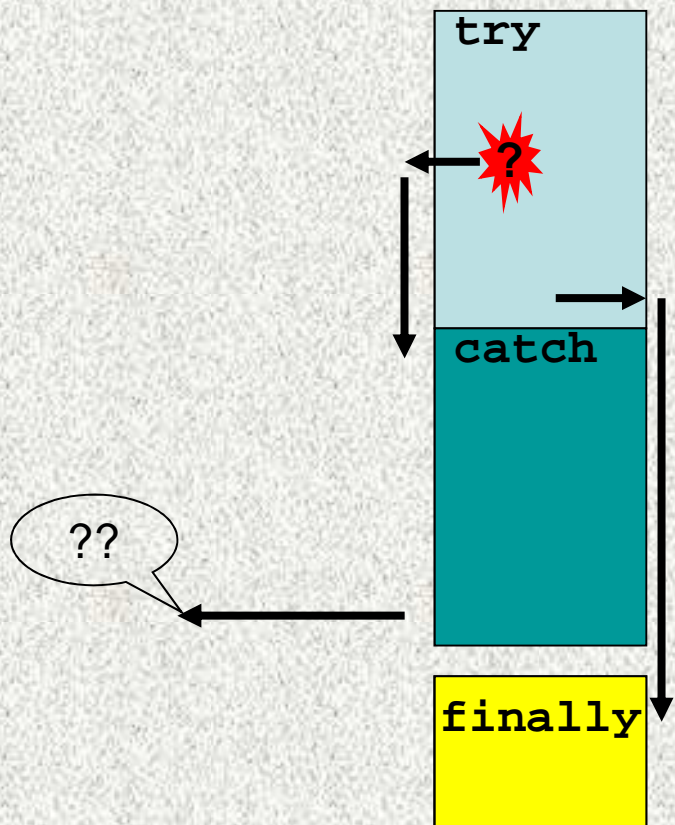
- I v případě, že máme pocit, že chyba „nemůže nastat“, ponecháme na kritickém místě otestování a výpis informací o výjimce pomocí:

```
catch (Exception e) {  
    e.printStackTrace(); // vypisuje chybové hlášení  
}
```

Pozn.: Výpis je na konzoli, pro grafické prostředí lépe do souboru:

```
System.setErr(PrintStream err)  
System.setOut(PrintStream out)
```

Bloky try a catch a jak dál po ...?



Blok `finally`

V běhu programu je často třeba vykonat nějaké akce, ať dojde k vyvolání výjimky či nikoli

- typicky ukončení práce se soubory

Tuto situaci řeší blok `finally`:

```
try {  
    // hlídaný blok  
}  
catch (TypVyjimky e) {  
    // ošetření výjimky e, proběhne pouze v případě vzniku výjimky  
}  
finally {  
    // akce, které proběhnou vždy  
}
```

Blok `finally` se vykoná i v případě (!!), že v blocích `try` či `catch` je:

- příkaz `return`
- vyvolána i nezachycená výjimka, propagovaná výše

Blok `finally`, příklad

```
class UseFinally {
    public static void genException(int what) {
        int t;
        int nums[] = new int[2];
        System.out.println("Přečteno " + what);
        try {
            switch(what) {
                case 0:
                    t = 10 / what; // dělení nulou
                    break;
                case 1:
                    nums[4] = 4; // mimo rozsah
                    break;
                case 2:
                    return; // výskok z try bloku, bez výjimky
            }
        }
    }
}
```

Blok `finally`, příklad, pokr. I

```
catch (ArithmeticException exc) {
    System.out.println("Nelze dělit nulou!");
}
catch (ArrayIndexOutOfBoundsException exc) {
    System.out.println("Mimo rozsah");
}
finally {
    System.out.println("    Společná větev");
}
}}
```

```
class FinallyDemo {
    public static void main(String args[]) {
        for(int i=0; i < 3; i++) {
            UseFinally.genException(i);
        }
    }
}
```

Blok `finally`, příklad, pokr. II

Výstup:

Přečteno 0

Nelze dělit nulou!

Společná větev

Přečteno 1

Mimo rozsah

Společná větev

Přečteno 2

Společná větev

Poznámka:

Konstrukci `try-catch-finally` lze zkrátit i na `try-finally`:

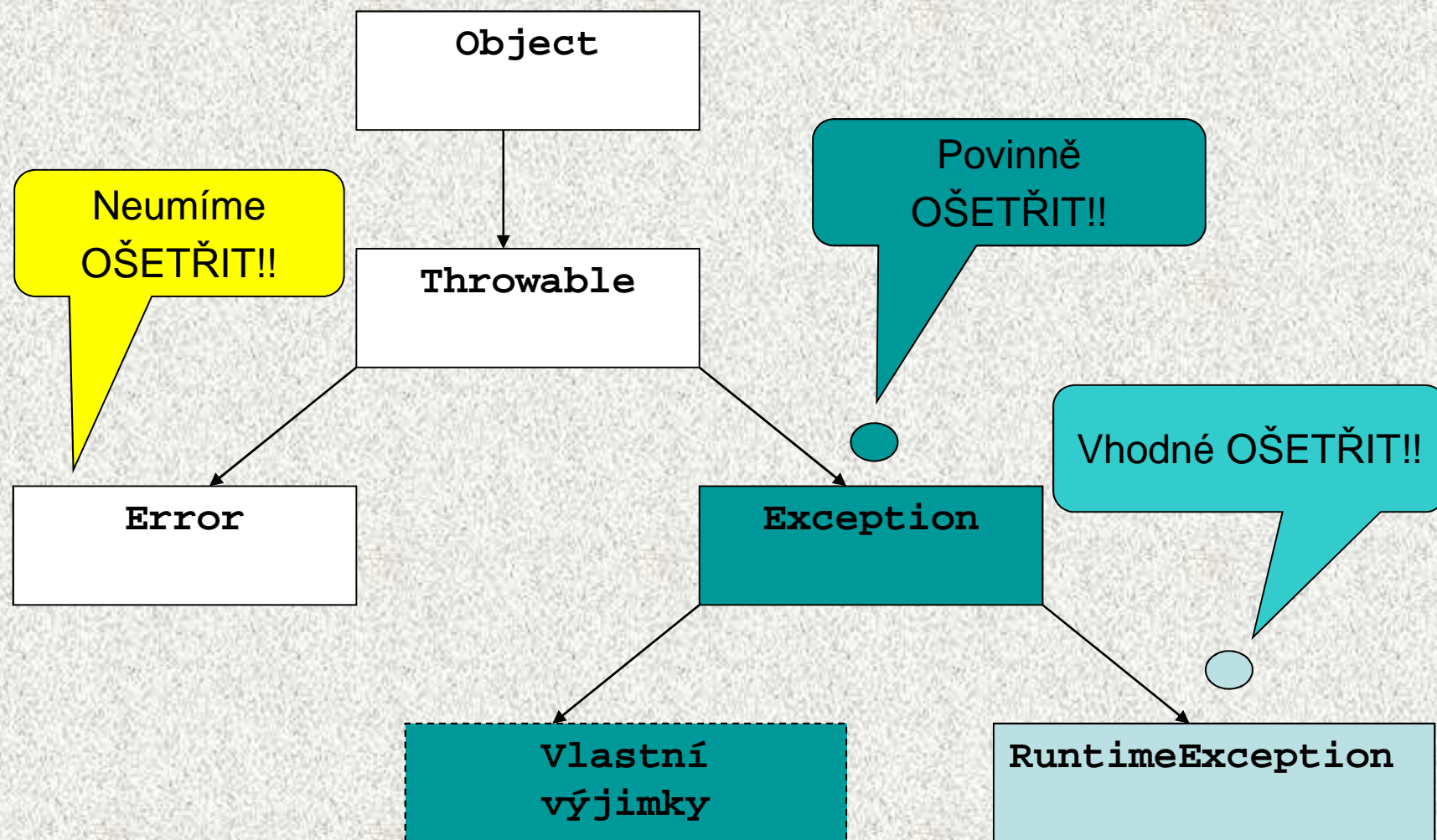
- důvodem může být požadavek bezpodmínečného vykonání části kódu při výjimce, která směřuje mimo, nemusí jít o ošetření výjimky

Seskupování výjimek

- V programu mohou nastat výjimky různých typů a úrovní, např.:
 - `IOException` (třída `Exception`) - musí být uvedena
 - `NumberFormatException` (třída `RuntimeException`) - měla by být uvedena, nenumerický vstup
- Pro každou výjimku - vlastní blok `catch`
- Bloky `catch` jsou procházeny v pořadí, jak jsou uvedeny:
- Jakmile vzniklá výjimka vyhovuje třídě uvedené v bloku `catch` (či rodičovské třídě), provede se tento blok a ostatní jsou přeskočeny ("`break`")
- Pozor, je-li v prvním `catch` zachycena výjimka `Exception`, ostatní bloky jsou zbytečné
- Lze tak rozlišovat různé úrovně výjimek:

```
catch (NumberFormatException e) {  
    System.out.println("Chyba ve formátu čísla");  
}  
catch (IOException e) { // nadtřída NumberFormatException  
    System.out.println("Jiná běhová chyba");  
}
```


Struktura tříd výjimek



Kontrolované a nekontrolované výjimky

- **Kontrolované** výjimky musí být na rozdíl od nekontrolovaných explicitně deklarovány v hlavičce metody, ze které se mohou šířit, jedná se o výjimky třídy **Exception**, je nutné je povinně obsloužit. Označují se též jako **výjimky synchronní**:

```
void m() throws Exception {  
    if (...) throw new Exception();  
}
```

- **Nekontrolované** výjimky jsou takové, které se mohou šířit z většiny metod a proto by jejich deklarování obtěžovalo, tzv. **asynchronní výjimky**:

- běžný uživatel není schopen výjimku ošetřit – třída **Error**:

MemoryOverflowError ~ přetečení paměti

- chyby, které ošetřujeme podle potřeby, překladač nekontroluje, zda tyto výjimky jsou ošetřeny - podtřídy třídy **RuntimeException**:

ArithmeticException ~ dělení 0

IndexOutOfBoundsException ~ indexace mimo meze

Třída **Error**

- Představuje závažné chyby na úrovni virtuálního stroje (JVM)
- Nejsme schopni je opravit, tedy neopravujeme
- Třída **Error** je nadtřída všech výjimek, které převážně vznikají v důsledku softwarových či hardwarových chyb výpočetního systému a které většinou nelze v aplikaci smysluplně ošetřit (např. **ClassFormatError**).

Příklad:

ClassFormatError ~ chybný formát byte-kódu

MemoryOverflowError ~ přetečení paměti

Třída `RuntimeException` (podtřída `Exception`)

- Představuje třídu chyb, která se můžeme s úspěchem ošetřit
- Je třeba je očekávat, jedná se tzv. **asynchronní výjimky**, např.:
 - dělení nulou - `ArithmeticException`
 - nedovolený převod znaků na číslo - `NumberFormatException`
- Nemusíme na ně reagovat, ale můžeme je předat "výše", překladač nás k reakci nenutí
- Reagujeme na ně podle našeho odhadu jejich výskytu
- Pokud to špatně odhadneme a nastane chyba, Java nám indikuje místo chyby, které opravíme tak, že tam vložíme ošetření výjimky
- Prakticky není možné ošetřit všechny výjimky `RuntimeException` - nepřehledný kód

`ArithmeticException`

- dělení nulou

`ClassCastException`

- nedovolené přetypování

`IndexOutOfBoundsException`

- indexace mimo meze

`NegativeArraySizeException`

- vytváření pole se zápornou délkou

`NullPointerException`

- dereference odkazu `null`

`NumberFormatException`

- přečtení nenumernické hodnoty

Třída `Exception`

- Označovány jako **kontrolované** (`checked`), nebo též **synchronní**, nemohou se vyskytovat kdekoli, jen v souvislosti s voláním "nebezpečných" metod, typicky metody vstupu/výstupu (`IOException`)
- Potomek třídy `Exception` může ošetřovat i "naše" výjimky, **jednoznačně synchronní**, vznikají na námi specifikovaném místě
- Vyžadují povinné ošetření, jinak se ohlásí překladač!
- Třída `Exception` je nadtřída výjimek, které převážně vznikají vlastní chybou aplikace a má smysl je ošetřovat
 - typicky ošetření chyb vstupu/výstupu (`IOException`)



Neošetření výjimky

- **Překladač:**

G:\Priprava VYUKA 2010\PR2\PROJEKTY\PR2-05-VYJIMKY\src\pr2_05\XVyjimka.java:18:
unreported exception java.io.FileNotFoundException; must be caught or declared to be
thrown Scanner sc = new Scanner(new File("cisla.txt")); 1 error

- **Výjimka – špatná data:**

Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:840)
at java.util.Scanner.next(Scanner.java:1461)
at java.util.Scanner.nextInt(Scanner.java:2091)

Precte cislo ze souboru at java.util.Scanner.nextInt(Scanner.java:2050)
at pr2_05.XVyjimka.main(XVyjimka.java:22)

Java Result: 1

- **Výjimka – chybí soubor:**

Exception in thread "main" java.io.FileNotFoundException: cisla.txt1 (Systém nemůže nalézt
uvedený soubor)

at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(FileInputStream.java:106)
at java.util.Scanner.<init>(Scanner.java:636)
at pr2_05.XVyjimka.main(XVyjimka.java:19)

Java Result: 1

BUILD SUCCESSFUL (total time: 1 second)