Společná část - otázka č. 12

Programování v jazyce JAVA: struktura tříd a programu. Události, zdroj a posluchač události, šíření událostí, vlastní události, více zdrojů a posluchačů, rozlišení zdrojů. Výjimky a jejich zpracování, propagace výjimek, hierarchie výjimek, kontrolované a nekontrolované výjimky. (A0B36PR2)

June 2, 2012

1 JAVA - struktura jazyka

Většina této části PRG2 byla zpracována pro otázky pro PRG1 (5, 6, 7 -> struktura tříd a programu). Proto zde vynecháno, nebo stručně doplněno co chybělo.

1.1 Typy programovacích jazyků

- deklarativní - nepopisují jak se co má dělat, ale co má být výsledkem (př. HTML, Prolog: nepopisuje kroky výpočtu, ale fakta o problému a požadovaný výsledek)
- imperativní - zbytek (JAVA, C...), viz. imperativní programování otázka č. 5

1.2 Programovací styly

- Naivní
- Procedurální
- Objektově orientovaný
- Návrhové vzory
- SOA - service-oriented architecture

2 JAVA GUI

(Tato kapitola = pouze stručný přehled <= není to přímo vypsáno jako otázka). Vizuální a interaktivní komunikaci počítač-člověk podporují balíčky:

2.1 Knihovny pro GUI

Základní knihovny:

2.1.1 AWT - Abstract Window ToolKit

- první, těžké(heavyweight)
- vykreslení zajišťuje platforma – rychlejší, ne vždy vše funguje vše stejně

2.1.2 SWING

- doporučené
- nové komponenty (tree-view, list box,...),
- robustní
- Look and Feel - na platformě nezávislý a vypadá stejně na všech platformách a přitom respektuje i18n
- důsledné oddělení modelu od pohledu a řadiče

2.1.3 (SWT-Standard Widget Toolkit, Eclipse IBM)

- podobné AWT (platformově závislé vykreslení)
- mnoho rozšiřujících vlastností

2.2 Základní součásti GUI

Velmi stručně, jen pro úvod do problematiky, v zadání otázky není vypsáno.

2.2.1 Komponenty (dialogové prvky) - v knihovně javax.swing

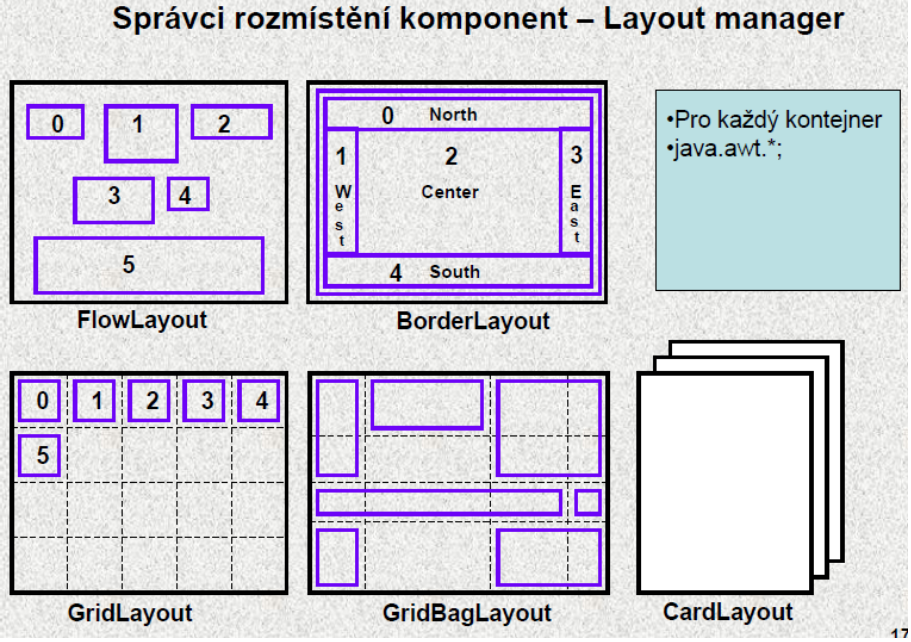
- tlačítka, seznamy, jezdci, textová pole, zatrhávací tlačítka, rádio tlačítka, ...
- společné metody pro velikost, barvu, umístění textu, ...

2.2.2 Kontejnery (v oknech) - v knihovně javax.swing

- Kontejnery se vkládají do oken
- komponenty musí být umístěny v kontejnerech
- dva základní typy kontejnerů:
  - **JPanel** – nejjednodušší, přidělí se komponenty (také JApplet),
  - **JFrame** – složitější, ale více možností

2.2.3 Správce rozmístění (Layout Manager) - v knihovně javax.swing a java.awt

- definuje pozici komponent v kontejneru
- postupně, pevná pozice, podle mřížky, sdružování, ..
- vzhled a chování celé aplikace




```
// pr. border layout
class Okno4_1 extends JFrame{

    public Okno4_1 () {

        ...
        Container kon = getContentPane();
        kon.setBackground(Color.green);
        BorderLayout srb = new BorderLayout();
```

```
        kon.setLayout(srb);
        JButton tl1 = new JButton("Test1");
        kon.add(tl1,srb.WEST);
        JButton tl2 = new JButton("Test2");
        kon.add(tl2,srb.EAST);
        JButton tl3 = new JButton("Test3");
        kon.add(tl3,srb.NORTH);
        setContentPane(kon);
    }
}
```

3 Obsluha událostí

Mechanismus, který umožní zpracovávat vstupní informace programu, predávané nejčastěji přes GUI. Mechanismus reakce na akci uživatele: stisk tlačítka, zadání textu, stisk tlačítka myši, ...

1. Zpracování vstupní informace v GUI je realizováno:
   a) vysláním „události“ na jedné straně „producentem“ (producent = třída)
   b) zachycením této „události“ (událost = objekt této třídy) „posluchačem“ (posluchač = naše třída)
2. Pro každou komponentu je třeba:
   a) deklarovat typ zachycované události, kterou je zájem zpracovat
   b) určit „posluchače“, který má událost obsloužit
3. Akcí uživatele vznikne událost
   a) událost je objektem Javy!
4. Události jsou zachyceny
   a) události jsou zpracovány (obslouženy) „posluchači“ (listener) – třídami s uživatelskými metodami pro reakci na událost
   b) „posluchači“ jsou třídy, které implementují rozhraní naslouchání – musejí mít schopnost „naslouchání“

3.1 Událost

Událost je objekt, který vznikne změnou stavu zdroje - důsledek interakce uživatele sřídícími grafickými elementy GUI

1. Událost vznikne:
   a) kliknutím na tlačítko
   b) stiskem klávesy
   c) posunem kurzoru, atd.
2. Události jsou produkovány tzv. producenty což jsou:
   a) tlačítka

b) rámy

c) ostatními grafické prvky

3.2 Zpracování události

Informace o jedné události (zdroj události, poloha kurzoru, atd.) jsou shromážděny v objektu, jehož třída určuje charakter události, napr.:

- ActionEvent ~ událost generovaná tlacítkem
- WindowEvent ~ událost generovaná oknem
- MouseEvent ~ událost generovaná myší

Všechny třídy událostí jsou **následníky třídy event** a jsou umísteny v **java.awt.event.***

Základní princip zpracování událostí:

1. Události jsou generovány zdroji událostí (jsou to objekty, které nesou informaci o události)
2. Události jsou přijímány ke zpracování posluchači událostí (to jsou opět objekty tříd s metodami schopnými událost zpracovat)
3. Zdroj události rozhoduje o tom, který posluchač má reagovat (registruje si svého posluchače)

```
class Okno extends JFrame{

    public Okno (){
        ...
        FlowLayout srb = new FlowLayout();
        kon.setLayout(srb);
        JButton aTlac = new JButton("Pred stiskem");
        kon.add(aTlac);
        // registrace posluchace
        aTlac.addActionListener(new Posluchac0());
        setContentPane(kon);
    }

}
// posluchac
class Posluchac0 implements ActionListener {

    public void actionPerformed(ActionEvent e) {
```

```
            //urcen objekt udalosti
            JButton o=(JButton)e.getSource();
            //reakce na udalost
            o.setLabel(“Po stisku");
        }
    }
```

Pozn. Jedna třída může být producentem i posluchačem (addActionListener(**this**), třída zároveň i implemetuje příslušné rozhranní a potřebnou metodu). Pro zpracování události se často používá i vnitřní třída (přehlednost, efektivita kódu, zapouzdření).

```
class Okno6 extends Okno3 {

    JLabel lab1;
    // vnitrni trida, obsluha udalosti
    class Udalost implements ActionListener {
        String vypis;
        public Udalost(String vypis) {
            this.vypis = vypis;
        }
        // oblsuha udalosti
        public void actionPerformed(ActionEvent e) {
            lab1.setText(vypis);
        }
    }
    Udalost aUD, nUD;
    // konstruktor
    Okno6() {
        lab1 = new JLabel("Nazdar6");
        kon.add(lab1);
        aUD = new Udalost("AHOJ");
        // registrace 1. posluchace
        aTlac.addActionListener(aUD);
        nUD = new Udalost("NAZDAR");
        // registrace 2. posluchace
        bbTlac.addActionListener(nUD);
    }

}
```

Anonymní třída jako posluchač:

```
aTlac.addActionListener(new ActionListener() {
```

```
        public void actionPerformed(ActionEvent e) {
            lab.setText("AHOJ");
        }
    });
```

3.3 Model šíření události

1. Události jsou předávány posluchačům, které si nejprve musí producent zaregistrovat – např. metodami:
   a) addActionListener()
   b) addWindowListener()
   c) addMouseListener()
   d) ...
2. Producent vysílá jen těm posluchačům, které si sám zaregistroval.
3. Posluchač musí implementovat některé z posluchačských rozhraní (!) (schopnost naslouchat):
   a) ActionListener
   b) WindowListener
   c) MouseListener
   d) ...

3.4 Implementace modelu události

Posluchac události musí implementovat príslušné rozhraní (interface), tj. implementovat príslušné abstraktní metody rozhraní. Pro každý druh události je definována **abstraktní metoda (handler)**, která tuto událost ošetruje:

- actionPerformed
- mouseClicked
- windowClosing, atd.

Handlery jsou deklarovány v rozhraních - posluchaci:

- ActionListener
- MouseListener
- WindowListener, atd.

Předání události posluchači ve skutečnosti znamená vyvolání činnosti handleru. Objekt události je předán jako skutečný parametr handleru. Producent registruje posluchače zavoláním registrační metody:

- addActionListener
- addMouseListener
- addWindowListener, atd.

Vazba mezi producentem a posluchacem je vztah N:M. Jeden posluchac muže být registrován u více producentů, u jednoho producenta muže být registrováno více posluchačů. Událost se předá všem posluchačům, avšak pořadí zpracování není zaručeno.

3.5 Více zdrojů události - jeden posluchač

Následuje ukázka zpracování události a rozlišení producenta:

```
// rozliseni popisem zdroje
public void actionPerformed(ActionEvent e) {

    String s = e.getActionCommand();
    String napis = (s.equals(aTlac.getLabel()))?"PRVNI":"DRUHE";
    lab.setText(napis);

}
// rozliseni objektem zdroje
public void actionPerformed(ActionEvent e) {

    Object o = e.getSource();
    String napis = (o.equals(aTlac)) ? "PRVNI!":"DRUHE!";
    lab.setText(napis);

}
```

3.6 Jeden zdroj událost - více posluchačů

Zajistí se zkrátka vícenásobným voláním metody addActionListener(new Posluchac()) u příslušného elementu. Každá třída=posluchač má svojí metodu, která se po události provede. Pořadí provedení není určeno.

3.7 Zrušení posluchače

removeActionListener(ActionListener posluchac)

4 Výjimky

Vyjímka je „nestandardní situace“:

1. Situace, které jsou nestandardní, či které my považujeme za nestandardní, měli bychom reagovat a mužeme a dokážeme reagovat (RuntimeException)
   a) Pokus o čtení z prázdného zásobníku EmptyStackException
   b) Dělení nulou, indexování mimo rozsah pole, špatný formát císel AritmeticException, NumberFormatException
2. Situace, na které musíme reagovat, Java nás přinutí (Exception, IOException)
   a) Odkaz na chybějící soubor FileNotFoundException
3. Chyba v hardware, závažné chyby, nemůžeme reagovat (Error), (OutOfMemoryError,UnknownError)
   a) Chyba v JVM
   b) HW chyba

Obecne chyba vzniká pri porušení sémantických omezení jazyka Java. Bezpe4nostní 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.

4.1 Reakce na výjimky

Chyba při provádení programu v jazyku Java nemusí znamenat ukončení programu – chybu lze ošetrit a pokracovat 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.

4.1.1 Úplné neošetření výjimky

Žádné použití throws nebo try-catch, způsobí ukončení programu - CHYBA. Java sama ohlásí při překladu, které části jsou kritické a je je třeba ošetrit. Nejsou-li, pak minimálně "vyhozením" na vyšší úroven pomocí klauzule throws, jinak nedojde k překladu.

4.1.2 Neošetření výjimky, ale předání výše

public static void main(String[] args) **throws IOException** { ... }

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

4.1.3 Ošetření výjimky a předání výš - throw

```
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; // predani vyse
    }

}
```

Volající metoda musí opět použít try-catch, nebo výjimku throws výše.

4.1.4 Kompletní Ošetření výjimky - try-catch

Kompletní ošetření výjimky se provádí konstrukcí try - catch:

try { ... } catch (Exception e) { ... }

Klauzulí catch muže být i více pro ruzné typy výjimek. Pokud výjimka vyhovuje jedné větvi catch, tato se provede a ostatní už ne. Existuje také blok finally, ten se provede vždy, ať už výjimka nastane nebo ne.

4.2 Mechanismus šíření výjimek

Jestliže vznikne výjimka, potom JVM hledá odpovídající klauzuli, která je schopná výjimku ošetrit (tj. převzít rízení):

1. pokud výjimka vznikla v bloku príkazu try, hledá se odpovídající klauzule catch v tomto príkazu, další príkazy bloku try se neprovedou a rízení se předá konstrukci ošetrující výjimku daného typu do místa ošetrení výjimky (tzv. handler = catch blok)

2. pokud výjimka vznikne mimo príkaz try, předá se rízení do místa volání metody a pokračuje se podle předchozího bodu

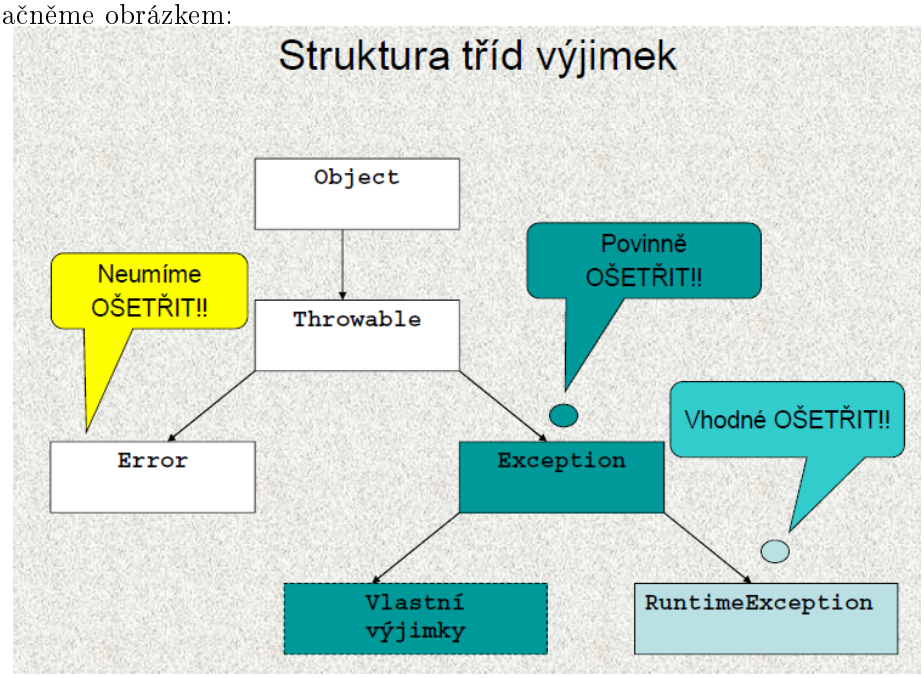
3. pokud taková konstrukce v těle funkce (metody, konstruktoru) není, skončí funkce nestandardně a výjimka se šíří na dynamicky nadřazenou úroven

4. není-li výjimka ošetrena ani ve funkci main, vypíše se (na výstup) a program skončí

Pro rozlišení různých typů výjimek je v jazyku Java zavedena řada knihovních tříd, výjimky jsou instancemi těchto tříd.

4.3 Typy výjimek

Začněme obrázkem:



4.3.1 Kontrolované

Kontrolované výjimky musí být na rozdíl od nekontrolovaných explicitne deklarovány v hlavicce metody, ze které se mohou šírit, jedná se o výjimky trídy Exception, je nutné je povinne obsloužit. Oznacují se též jako výjimky synchronní:

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

Potomek třídy Exception muže ošetrovat i "naše" výjimky, jednoznačne synchronní, vznikají na námi specifikovaném míste. Vyžadují povinné ošetrení, jinak se ohlásí!. Třída Exception je nadtřída výjimek, které prevážne vznikají vlastní chybou aplikace a má smysl je ošetrovat (typicky ošetrení chyb vstupu/výstupu (IOException)).

4.3.2 Nekontrolované

Nekontrolované výjimky jsou takové, které se mohou šírit z vetšiny metod a proto by jejich deklarování obtežovalo, tzv. asynchronní výjimky.

- bežný uživatel není schopen výjimku ošetrit – ze třídy Error. Třída Error je nadtřída všech výjimek, které převážne vznikají v důsledku softwarových či hardwarových chyb výpočetního systému a které vetšinou nelze v aplikaci smysluplně ošetrit.
  - MemoryOverflowError - přetecení paměti
  - ClassFormatError - chybný formát byte-kódu
- chyby, které ošetrujeme podle potřeby, prekladac nekontroluje, zda tyto výjimky jsou ošetreny - podtřídy třídy RuntimeException. Nemusíme na ně reagovat, ale mužeme je predat "výše", překladač nás k reakci nenutí.
  - ArithmeticException - delení 0
  - IndexOutOfBoundsException - indexace mimo meze
  - NumberFormatException - nedovolený prevod znaku na císlo, prectení nenumerické hodnoty
  - NegativeArraySizeException - vytvárení pole se zápornou délkou
  - NullPointerException - dereference odkazu null

4.4 Vlastní výjimky

Ve vlastních třídách muže nastat stav, který chceme ošetrit standardně výjimecným stavem. Vlastní výjimka je potomkem třídy Exception. Jedná se o tzv.synchronní výjimku, vzniká na presne definovaném míste. Vetšinou se jedná o výjimku, na kterou chceme reakci uživatele. Reakci na vlastní výjimky systém vyžaduje.

```
throw new Exception(); // generujeme vyjímku
```

4.4.1 Vytvoření vlastní výjimky

```
class MojeVyjimka extends Exception {

    int n;
    int d;
    MojeVyjimka(int i, int j) {
        n = i;
        d = j;
    }
    public String toString() {
```

```
        return "Hodnota " + n + "/" + d +" není integer.";
    }
}
//priklad pouziti
throw new MojeVyjimka(numer[i], denom[i]);
```