TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektronika a informatika

Studijní obor: 1802R022 – Informatika a logistika

Implementace tříd pro efektivní práci s polynomy v R^1, R^2, R^3

Classes implementation for effective work
with multinominals in R^1, R^2, R^3

Bakalářská práce

Autor: **Jan Eder**

Vedoucí bakalářské práce: Ing. Dalibor Frydrych, Ph. D.

Konzultant: Ing. Jan Šembera, Ph. D.

V Liberci 15. 5. 2007

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum: 15. 5. 2007

Podpis: ….....

Jan Eder

Poděkování

Zde bych rád poděkoval vedoucímu mojí bakalářské práce Panu Ing. Daliboru Frydrychovi, Ph. D. za jeho profesionální vedení, ochotu řešit a konzultovat problémy spojené s danou problematikou a především za jeho lidský přístup.

Velký dík patří i Panu Ing. Janu Šemberovi, Ph. D. pod jehož patronací tato práce vznikala.

Nakonec bych rád poděkoval rodině za její trpělivost.

Abstrakt

Tato práce se zabývá implementací základních tříd popisujících polynomy v objektovém jazyce JAVA. Implementace je provedena pro polynomy jedné, dvou a tří proměnných. Realizuje efektivní způsob uložení polynomu v paměti počítace a obsahuje podstatné matematické operace, které lze s polynomy provádět.

Práce představuje ucelený a dokumentovaný nástroj pro práci s polynomy, který lze použít při vývoji rozsáhlých modelů založených na metodě konečných prvků.

Abstract

This work deals with implementation of basics classes describing multinominals in object language JAVA. Implementation is executed for multinominals one, two and three of variables. It realizes effective method of storing multinominals and it includes essentials mathematical operations which they do with multinominals.

The work presents compact and historiated instrument for work with multinominals which it can be used in development of larges – scales models based on finite element method.

Obsah

Úvod

Dnes nejrozšířenější metodou řešení technických problémů popsaných parciálními diferenciálními rovnicemi je metoda konečných prvků. Podstatou této metody je diskretizace řešené oblasti a hledání aproximace řešení úlohy pomocí jednoduchých, po částech spojitých funkcí.

Jako standardní aproximační funkce lze dnes označit lineární funkce. Výhodou lineárních aproximačních funkcí a důvod jejich velkého rozšíření je jejich snadná implementace. Při hrubé distretizaci řešené oblasti však dochází k velké chybě. Tu lze snížit zjemněním diskretizace. To ale vede ke zvýšení nároků na výpočetní výkon.

Volba kvalitnějších aproximačních funkcí přináší kvalitnější aproximaci řešení, ale na druhé straně vyžaduje náročnější implementaci. S velkou výhodou lze použít pro aproximaci řešení polynomy. Nabízí širokou škálu aproximačních funkcí, kdy zvyšování řádu polynomu přináší různou kvalitu aproximace řešení.

Tato práce se bude zabývat implementací základních tříd popisujících polynomy a matematické operace, které lze s polynomy provádět.

1 Teoretická část

Teoretická část je zaměřena na objasnění pojmu polynom a na matematické operace, které se s polynomy provádí. Kromě vysvětlení polynomu jedné proměnné zde jsou vysvětleny polynomy dvou a tří proměnných. Matematické operace prováděné s polynomy jsou blíže popsány a vysvětleny pro každý typ polynomu zvlášť.

1.1 Polynomy

Obecně se polynomem jedné proměnné n – tého řádu rozumí výraz ve tvaru

$$p(x)=\sum_{i=0}^{n} a_i \cdot x^i = a_0 \cdot x^0 + a_1 \cdot x^1 + a_2 \cdot x^2 + ... + a_n \cdot x^n, \quad kde \quad a_n \neq 0 \quad .$$

Nechť je n dané přirozené číslo ($n \in$ N), $a_0, a_1, a_2, .. , a_n$ daná reálná čísla, x je proměnná, pak součet se nazývá mnohočlen (polynom) n – tého řádu v proměnné x s koeficienty $a_0, a_1, a_2, .. , a_n$ z číselného oboru R. Sčítanci $a_i x_i$ se nazývají členy mnohočlenu. Člen a_0 se nazývá absolutní člen, člen $a_1 x$ lineární člen, člen $a_2 x^2$ kvadratický člen a člen $a_3 x^3$ se nazývá kubickým členem.

Pojem polynomu lze zobecnit na případ více proměnných. Polynomem dvou proměnných n – tého řádu se označuje polynom

$$p(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i} a_{ij} \cdot x^i \cdot y^j = a_{00} \cdot x^0 \cdot y^0 + a_{10} \cdot x^1 \cdot y^0 + a_{01} \cdot x^0 \cdot y^1 + ... + a_{ij} \cdot x^i \cdot y^j ,$$
$$kde \exists a_{ij} \neq 0 \quad pro \quad i+j=n \quad , \quad n \in N \quad .$$

Polynomem tří proměnných n – tého řádu se označuje polynom

$$p(x,y,z)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j} a_{ijk} \cdot x^i \cdot y^j \cdot z^k = a_{000} \cdot x^0 \cdot y^0 \cdot z^0 + a_{001} \cdot x^0 \cdot y^0 \cdot z^1$$
$$+ a_{010} \cdot x^0 \cdot y^1 \cdot z^0 + a_{100} \cdot x^1 \cdot y^0 \cdot z^0 + ... + a_{ijk} \cdot x^i \cdot y^j \cdot z^k ,$$
$$kde \quad \exists a_{ijk} \neq 0 \quad pro \quad i+j+k=n \quad , \quad n \in N \quad .$$

S polynomy lze provádět matematické operace součet, součin, parciální derivaci, integraci a výpočet funkční hodnoty. Tato bakalářská práce je zaměřená na implementaci těchto matematických operací.

1.2 Součet polynomů

Polynom jedné proměnné

Je dán polynom *n* – tého řádu

$$f(x)=\sum_{i=0}^{n} a_i \cdot x^i$$

a polynom *m* – tého řádu, kde *m* je dané přirozené číslo ($m \in \mathrm{N}$),

$$g(x)=\sum_{i=0}^{m} b_i \cdot x^i \quad .$$

Sečtením polynomů *f(x)* a *g(x)* je získán polynom

$$h(x)=f(x)+g(x)=\sum_{i=0}^{r} \left(a_i+b_i\right) \cdot x^i \quad ,$$

kde $r = max\ (n,m)$ je řád výsledného polynomu.

Polynom dvou proměnných

Je dán polynom *n* – tého řádu

$$f(x,y)=\sum_{i=0}^{n} \sum_{j=0}^{n-i} a_{ij} \cdot x^i \cdot y^j$$

a polynom *m* – tého řádu, kde *m* je dané přirozené číslo ($m \in \mathrm{N}$),

$$g(x,y)=\sum_{i=0}^{m} \sum_{j=0}^{m-i} b_{ij} \cdot x^i \cdot y^j \quad .$$

Sečtením polynomů *f(x, y)* a *g(x, y)* je získán polynom

$$h(x,y)=f(x,y)+g(x,y)=\sum_{i=0}^{r} \sum_{j=0}^{r-i} \left(a_{ij}+b_{ij}\right) \cdot x^i \cdot y^j \quad ,$$

kde $r = max\ (n, m)$ je řád výsledného polynomu.

Polynom tří proměnných

Je dán polynom *n* – tého řádu

$$f(x,y,z)=\sum_{i=0}^{n} \sum_{j=0}^{n-i} \sum_{k=0}^{n-i-j} a_{ijk} \cdot x^i \cdot y^j \cdot z^k$$

a polynom *m* – tého řádu, kde *m* je dané přirozené číslo ($m \in \mathrm{N}$),

$$g(x,y,z)=\sum_{i=0}^{m}\sum_{j=0}^{m-i}\sum_{k=0}^{m-i-j} b_{ijk}\cdot x^i\cdot y^j\cdot z^k \quad .$$

Sečtením polynomů *f(x, y, z)* a *g(x, y, z)* je získán polynom

$$h(x,y,z)=f(x,y,z)+g(x,y,z)=\sum_{i=0}^{r}\sum_{j=0}^{r-i}\sum_{k=0}^{r-i-j}\left(a_{ijk}+b_{ijk}\right)\cdot x^i\cdot y^j\cdot z^k \quad ,$$

kde *r = max (n, m)* je řád výsledného polynomu.

1.3 Součin polynomů

Polynom jedné proměnné

Je dán polynom *n* – tého řádu

$$f(x)=\sum_{i=0}^{n} a_i\cdot x^i \quad ,$$

a polynom *m* – tého řádu

$$g(x)=\sum_{j=1}^{m} b_j\cdot x^j \quad .$$

Součinem polynomů *f(x)* a *g(x)* je získán polynom

$$h(x)=f(x)\cdot g(x)=\sum_{i=0}^{n}\sum_{j=0}^{m} a_i\cdot b_j\cdot x^{i+j} \quad ,$$

kde *n + m* je řád výsledného polynomu.

Polynom dvou proměnných

Je dán polynom *n* – tého řádu

$$f(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i} a_{ij}\cdot x^i\cdot y^j \quad ,$$

a polynom *m* – tého řádu

$$g(x,y)=\sum_{k=0}^{m}\sum_{l=0}^{m-k} b_{kl}\cdot x^k\cdot y^l \quad .$$

Součinem polynomů *f(x, y)* a *g(x, y)* je získán polynom

$$h(x,y)=f(x,y)\cdot g(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{m}\sum_{l=0}^{m-k}a_{ij}\cdot b_{kl}\cdot x^{i+k}\cdot y^{j+l} \quad ,$$

kde $n + m$ je řád výsledného polynomu.

Polynom tří proměnných

Je dán polynom *n* – tého řádu

$$f(x,y,z)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j}a_{ijk}\cdot x^{i}\cdot y^{j}\cdot z^{k} \quad ,$$

a polynom *m* – tého řádu

$$g(x,y,z)=\sum_{q=0}^{m}\sum_{r=0}^{m-q}\sum_{s=0}^{m-q-r}b_{qrs}\cdot x^{q}\cdot y^{r}\cdot z^{s} \quad .$$

Součinem polynomů *f(x, y, z)* a *g(x, y, z)* je získán polynom

$$h(x,y,z)=f(x,y,z)\cdot g(x,y,z)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j}\sum_{q=0}^{m}\sum_{r=0}^{m-q}\sum_{s=0}^{m-q-r}a_{ijk}\cdot b_{qrs}\cdot x^{i+q}\cdot y^{j+r}\cdot z^{k+s} \quad ,$$

kde $n + m$ je řád výsledného polynomu.

1.4 Parciální derivace polynomu

Polynom jedné proměnné

Je dán polynom *n* – tého řádu

$$p(x)=\sum_{i=0}^{n}a_{i}\cdot x^{i} \quad .$$

Jeho parciální derivací podle proměnné *x* vznikne polynom *m* – tého řádu ($m = n – 1$) ve tvaru

$$\frac{\partial p}{\partial x}=\sum_{i=0}^{m}b_{i}\cdot x^{i}, \quad kde \quad b_{i}=(a_{i+1})\cdot(i+1) \quad .$$

Polynom dvou proměnných

Je dán polynom *n* – tého řádu

$$f(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}a_{ij}\cdot x^{i}\cdot y^{j} \quad .$$

Jeho parciální derivací podle proměnné x vznikne polynom m – tého řádu ($m = n - 1$) ve tvaru

$$\frac{\partial f}{\partial x} = \sum_{i=0}^{m} \sum_{j=0}^{m-1} b_{ij} \cdot x^i \cdot y^j , \quad kde \quad b_{ij} = (a_{(i+1)j}) \cdot (i+1) \ ,$$

a parciální derivací podle proměnné y vznikne polynom p – tého řádu ($p = n - 1$) ve tvaru

$$\frac{\partial f}{\partial y} = \sum_{i=0}^{p} \sum_{j=0}^{p-1} c_{ij} \cdot x^i \cdot y^j , \quad kde \quad c_{ij} = (a_{i(j+1)}) \cdot (j+1) \ .$$

Parciální derivací polynomu podle proměnné x se rozumí derivace jednotlivých sčítanců obsahujících proměnnou x. Pokud sčítanec obsahuje i proměnnou y, považuje se y za konstantu a nederivuje se. Parciální derivací polynomu podle proměnné y se rozumí derivace jednotlivých sčítanců obsahujících proměnnou y. Pokud sčítanec obsahuje i proměnnou x, považuje se x se za konstantu a nederivuje se.

Polynom tří proměnných

Je dán polynom n – tého řádu

$$g(x, y, z) = \sum_{i=0}^{n} \sum_{j=0}^{n-i} \sum_{k=0}^{n-i-j} a_{ijk} \cdot x^i \cdot y^j \cdot z^k \ .$$

Jeho parciální derivací podle proměnné x vznikne polynom m – tého řádu ($m = n - 1$) ve tvaru

$$\frac{\partial g}{\partial x} = \sum_{i=0}^{m} \sum_{j=0}^{m-i} \sum_{k=0}^{m-i-j} b_{ijk} \cdot x^i \cdot y^j \cdot z^k \ , \quad kde \quad b_{ijk} = a_{(i+1)jk} \cdot (i+1) \ .$$

Parciální derivací podle proměnné y vznikne polynom p – tého řádu ($p = n - 1$) ve tvaru

$$\frac{\partial g}{\partial y} = \sum_{i=0}^{p} \sum_{j=0}^{p-i} \sum_{k=0}^{p-i-j} c_{ijk} \cdot x^i \cdot y^j \cdot z^k \ , \quad kde \quad c_{ijk} = a_{i(j+1)k} \cdot (j+1) \ ,$$

a parciální derivací podle proměnné z vznikne polynom q – tého řádu ($q = n - 1$) ve tvaru

$$\frac{\partial g}{\partial z} = \sum_{i=0}^{q} \sum_{j=0}^{q-i} \sum_{k=0}^{q-i-j} d_{ijk} \cdot x^i \cdot y^j \cdot z^k \ , \quad kde \quad d_{ijk} = a_{ij(k+1)} \cdot (k+1) \ .$$

Parciální derivací polynomu podle proměnné x se rozumí derivace jednotlivých sčítanců obsahujících proměnnou x.

Pokud sčítanec obsahuje i proměnné *y* nebo *z*, považují se *y* a *z* za konstanty a nederivují se. Parciální derivací polynomu podle proměnné *y* se rozumí derivace jednotlivých sčítanců obsahujících proměnnou *y*. Pokud sčítanec obsahuje i proměnné *x* nebo *z*, považují se *x* a *z* se za konstanty a nederivují se. Parciální derivací polynomu podle proměnné *z* se rozumí derivace jednotlivých sčítanců obsahujících proměnnou *z*. Pokud sčítanec obsahuje i proměnné *x* nebo *y*, považují se *x* a *y* se za konstanty a nederivují se.

1.5 Integrace polynomu

Polynom jedné proměnné

Je dán polynom *n* – tého řádu

$$p(x)=\sum_{i=0}^{n} a_i \cdot x^i \quad .$$

Jeho integrací podle proměnné *x* vznikne polynom *m* – tého řádu (*m* = *n* + 1) ve tvaru

$$\int p(x)\,dx=\sum_{i=1}^{m} b_i \cdot x^i, \quad kde \quad b_i = a_{i-1}\cdot\left(\frac{1}{i}\right) \quad a \quad b_0=0 \quad .$$

Polynom dvou proměnných

Je dán polynom *n* – tého řádu

$$f(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i} a_{ij} \cdot x^i \cdot y^j \quad .$$

Jeho integrací podle proměnné *x* vznikne polynom *r* – tého řádu (*r* = *n* + 1) ve tvaru

$$\int f(x,y)\,dx=\sum_{i=1}^{r}\sum_{j=0}^{r-i} b_{ij} \cdot x^i \cdot y^j, \quad kde \quad b_{ij}=a_{(i-1)j}\cdot\left(\frac{1}{i}\right) \quad a \quad b_{00}=0 \quad ,$$

a integrací podle proměnné *y* vznikne polynom *p* – tého řádu (*p* = *n* + 1) ve tvaru

$$\int f(x,y)\,dy=\sum_{i=0}^{p}\sum_{j=0}^{p-i} c_{ij} \cdot x^i \cdot y^j, \quad kde \quad c_{ij}=a_{i(j-1)}\cdot\left(\frac{1}{j}\right) \quad a \quad c_{00}=0 \quad .$$

Polynom tří proměnných

Je dán polynom n – tého řádu

$$f(x,y,z)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j} a_{ijk}\cdot x^i\cdot y^j\cdot z^k \quad .$$

Jeho integrací podle proměnné x vznikne polynom r – tého řádu ($r = n + 1$) ve tvaru

$$\int f(x,y,z)dx=\sum_{i=1}^{r}\sum_{j=0}^{r-i}\sum_{k=0}^{r-i-j} b_{ijk}\cdot x^i\cdot y^j\cdot z^k \quad , \quad kde \quad b_{ijk}=a_{(i-1)jk}\cdot\left(\frac{1}{i}\right) \quad a \quad b_{000}=0 \quad .$$

Integrací podle proměnné y vznikne polynom p – tého řádu ($p = n + 1$) ve tvaru

$$\int f(x,y,z)dy=\sum_{i=1}^{p}\sum_{j=0}^{p-i}\sum_{k=0}^{p-i-j} c_{ijk}\cdot x^i\cdot y^j\cdot z^k \quad , \quad kde \quad c_{ijk}=a_{i(j-1)k}\cdot\left(\frac{1}{j}\right) \quad a \quad c_{000}=0 \quad ,$$

a integrací podle proměnné z vznikne polynom q – tého řádu ($q = n + 1$) ve tvaru

$$\int f(x,y,z)dz=\sum_{i=1}^{q}\sum_{j=0}^{q-i}\sum_{k=0}^{q-i-j} d_{ijk}\cdot x^i\cdot y^j\cdot z^k \quad , \quad kde \quad d_{ijk}=a_{ij(k-1)}\cdot\left(\frac{1}{k}\right) \quad a \quad d_{000}=0 \quad .$$

1.6 Funkční hodnota polynomu

Polynom jedné proměnné

Při výpočtu funkční hodnoty polynomu v konkrétním bodě A se za proměnnou x dosadí souřadnice bodu A a provede se suma. Tento postup lze zefektivnit použitím tzv. **Hornerova schematu,** které značně zrychluje výpočet tím, že se při výpočtu nepracuje s mocninami proměnné x, ale pouze se provádí součet součinů.

Polynom n – tého řádu

$$p(x)=\sum_{i=0}^{n} a_i\cdot x^i$$

lze zapsat ve tvaru

$$p(x)=(...((a_n\cdot x+a_{n-1})\cdot x+a_{n-2})\cdot x+...+a_1)\cdot x+a_0$$

zapíšeme – li

$$\begin{aligned}
&suma=0 \quad ,\\
&suma_n=a_n \quad ,\\
&suma_{n-1}=suma_n\cdot x+a_{n-1} \quad ,\\
&suma_{n-2}=suma_{n-1}\cdot x+a_{n-2} \quad ,\\
&...\\
&suma_0=suma_1\cdot x+a_0 \quad ,
\end{aligned}$$

pak poslední číslo $suma_o$ představuje právě hodnotu polynomu $p(x)$ v bodě x.

Polynom dvou proměnných

Zde nelze použít **Hornerovo schema**, ale určitou jeho modifikaci. Při výpočtu se nepočítají přímo mocniny proměnných *x* a *y*, ale tyto mocniny se předpočítávají tak, že jsou vytvořeny 2 jednorozměrné pole, kde následující pozice v poli je výsledkem součinu předchozí pozice v poli s konkrétní hodnotou proměnné *x* nebo *y*. To vede ke zrychlení celého výpočtu oproti výpočtu používajícího mocniny.

Polynom *n* – tého řádu

$$f(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i} a_{ij}\cdot x^i\cdot y^j$$

lze přepsat do tvaru

$$f(x,y)=\sum_{i=0}^{n}\sum_{j=0}^{n-i} a_{ij}\cdot x^i\cdot y^j = a_{00}\cdot x[0]\cdot y[0]+a_{10}\cdot x[1]\cdot y[0]+..+a_{ij}\cdot x[i]\cdot y[j]\ ,$$

kde

$$\begin{array}{ll} x[0]=1.0\ , & y[0]=1.0\ , \\ x[1]=x[0]\cdot x\ , & y[1]=y[0]\cdot y\ , \\ x[2]=x[1]\cdot x\ , & y[2]=y[1]\cdot y\ , \\ \ldots & \ldots \\ x[i]=x[i-1]\cdot x\ , & y[j]=y[j-1]\cdot y\ . \end{array}$$

Po zvolení proměnné, do které se budou ukládat dílčí součty, je výsledkem funkční hodnota polynomu dvou proměnných *n* – tého řádu při konkrétních hodnotách proměnných *x* a *y*.

Polynom tří proměnných

Výpočet funkční hodnoty polynomu tří proměnných je podobný výpočtu funkční hodnoty polynomu dvou proměnných. Opět zde nelze aplikovat **Hornerovo schema**, ale pouze jeho modifikaci. Ke zrychlení celého výpočtu se použijí tři jednorozměrná pole, ve kterých se předpočítávají mocniny proměnných *x*, *y* a *z* tak, že hodnota následující pozice v poli je výsledkem součinu předchozí pozice v poli s konkrétní hodnotou proměnné *x* nebo *y* nebo *z*.

Polynom n – tého řádu

$$p(x,y,z)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j} a_{ijk}\cdot x^i\cdot y^j\cdot z^k$$

lze přepsat do tvaru

$$p(x,y,z)=\sum_{i=0}^{n}\sum_{j=0}^{n-i}\sum_{k=0}^{n-i-j} a_{ijk}\cdot x^i\cdot y^j\cdot z^k = a_{000}\cdot x[0]\cdot y[0]\cdot z[0]+a_{100}\cdot x[1]\cdot y[0]\cdot z[0] + a_{ijk}\cdot x[i]\cdot y[j]\cdot z[k] \ ,$$

kde

$$\begin{array}{lll} x[0]=1.0 \ , & y[0]=1.0 \ , & z[0]=1.0 \ , \\ x[1]=x[0]\cdot x \ , & y[1]=y[0]\cdot y \ , & z[1]=z[0]\cdot z \ , \\ x[2]=x[1]\cdot x \ , & y[2]=y[1]\cdot y \ , & z[2]=z[1]\cdot z \ , \\ \ldots & \ldots & \ldots \\ x[i]=x[i-1]\cdot x \ , & y[j]=y[j-1]\cdot y \ . & z[j]=z[j-1]\cdot z \ . \end{array}$$

Po zvolení proměnné, do které se budou ukládat dílčí součty, je výsledkem funkční hodnota polynomu tří proměnných n – tého řádu při konkrétních hodnotách proměnných x a y a z.

2 OOP – Základy

Tato kapitola má za cíl přiblížit a objasnit čtenáři problematiku objektově orientovaného programování.

Základní myšlenkou OOP je rozložení celku na jednotlivé objekty komunikující mezi sebou. Poskládáním jednotlivých objektů vznikne funkční aplikace. Výhodou objektově orientovaného přístupu je oddělení autorské části programu od uživatelské části.

Pro jednotnou terminologii celé bakalářské práce se bude pod pojmem objekt vždy uvažovat polynom dané proměnné z důvodu větší přehlednosti a vysvětlení dané problematiky.

2.1 Třídy a objekty

Třídy jsou základními kameny OOP. Třída definuje data a metody pracující nad těmito daty a slouží jako předloha pro vytváření objektů. Obsahuje vlastnosti společné pro všechny objekty vytvořené podle této třídy. Vytvořených objektů podle konkrétní třídy může být libovolné množství, ale také žádný.

Uvažujme třídu Polynom reprezentující obecný polynom. Po každém polynomu vytvořeném podle této třídy chceme, aby měl svoje jméno, řád, koeficienty a abychom s ním mohli provádět matematické operace součet, součin, derivaci, integraci a výpočet funkční hodnoty. Řád a jméno polynomu jsou data. Matematické operace jsou metody definované v této třídě. Při vytváření objektu, v našem případě polynomu, se zadají parametry řád nebo jméno a pole koeficientů. Výsledkem je polynom využívající matematické operace definované v třídě PolynomD.

2.2 Datové typy

Data mohou být proměnné primitivního datového typu (int, double, ...) nebo proměnné typu třída. Každá proměnná má pevně určený datový typ a může v ní být uložena pouze hodnota odpovídající tomuto datovému typu.

Datový typ proměnné určuje obor hodnot, které proměnná může nabývat.

Proměnná primitivního datového typu nese pouze jednu elementární, atomickou a dále nestrukturovatelnou hodnotu. Proměnná typu třída obsahuje pouze odkaz na příslušný objekt nebo pole.

2.3 Metody

Metodami se v OOP rozumí procedury nebo funkce. Slouží nejen k provádění matematických operací s polynomy, ale také k jejich správě (tisk, nastavení koeficientů).

Metody mohou být rozhraním mezi uživatelem a jejich vnitřní implementací. Pokud uživatel chce např. zderivovat polynom, použije příslušnou metodu. Vnitřní strukturu a algoritmus výpočtu nevidí, nemá k němu přístup. Ne vždy je dobré, aby měl uživatel přístup ke všem datům. To, k jakým datům může mít uživatel přístup je vysvětleno v kapitole 2.4.

2.4 Řízení přístupu

Přístup k datům lze řídit v několika úrovních. Vše, co je označeno **public** (veřejný), je přístupné z libovolného místa. Vše, co je označeno **private** (soukromý), je přístupné pouze v rámci třídy. Vše, co je označeno **protected** (chráněný), je přístupné jen ze tříd poděděných od této třídy. Pokud není označení uvedeno, nazývá se tento přístup přátelským a vše je přístupné jen ze tříd stejného balíku.

Tato slova označující úroveň přístupu uživatele k datům se nazývají modifikátory a používají se u proměnných, metod a tříd.

2.5 Dědičnost

Dědičnost je velmi silným principem využívaným v OOP. Pokud se v jedné nebo více třídách objevuje stejný zdrojový kód, je vhodné tento kód napsat do jedné třídy, od které tyto třídy vzniknou děděním. Výhodou je, že nově vzniklá třída (potomek) má k dispozici stejná data a metody pracující nad danými daty jako třída, od které byla poděděná (rodič). Při modifikaci zdrojového kódu se tento kód nemusí upravovat ve všech třídach, které ho obsahují, ale upraví se pouze v rodičovské třídě, tzn. zdrojový kód se modifikuje na jednom místě, což vede k větší přehlednosti, usnadnění práce a vyšší produktivitě.

V tomto projektu se od obecné třídy Polynom se odvodí třídy Polynom1D, Polynom2D a Polynom3D. Tyto třídy zapouzdřují data a metody odpovídající konkrétním typům polynomů.

2.6 Abstraktní třída

Abstraktní třída je rodičovskou třídou pro více tříd odpovídajících konkrétním objektům, sama však žádné objekty nemůže vytvářet. Může obsahovat data a poskytovat metody pracující nad těmito daty, které jsou společné těmto odvozeným třídám.

Polynom je abstraktním pojmem, pro který není možné přesně definovat matematické operace, protože nevíme kolika proměnných tento polynom je. Je tedy abstraktní třídou. Obsahuje však konkrétní data společná pro všechny polynomy, např. jméno polynomu, řád a rozhraní metod, které pracují s těmito daty.

Mezi ně patří např. getName() a setName(). Tyto metody jsou definovány jako abstraktní, obsahují pouze definice daných metod. Nemají žádné tělo a v poděděných třídách je nutné je implementovat – toto si vynutí překladač.

3 Implementace úlohy

K řešení této bakalářské práce zabývající se prací s polynomy bylo použito jazyka Javy. Java je objektově orientovaným programovacím jazykem a důvodů pro použití tohoto programovacího jazyka bylo, že Java je právě jazykem objektovým, nezávislým na platformě, bezpečným a jednoduchým.

Pro práci s polynomy v R^1, R^2 a R^3 byla vytvořena abstraktní třída s názvem Polynom zastřešující třídy s názvy Polynom1D, Polynom2D a Polynom3D odpovídající třídám polynomům jedné, dvou a třech proměnných. Abstraktní třída je rodičem těchto tříd, obsahuje data a poskytuje metody společné těmto třídám.

Názvy tříd byly takto zkráceny z důvodu lepší orientace ve zdrojovém kódu a z programátorského hlediska, kdy takto zkrácený název plně vystihuje o jaký polynom se jedná, resp. o jaký polynom daného prostoru se jedná.

3.1 Výběr vhodných datových struktur

Výběr vhodných datových struktur je důležitým aspektem již v počátku celé analýzy. Chybně zvolená datová struktura značně komplikuje celé řešení a naopak vhodně zvolená datová struktora vede k efektivnímu řešení celé úlohy. Uložit polynom do paměti lze buď pomocí kolekce nebo pomocí pole.

Uložení polynomu do kolekce pro tuto úlohu není tak efektivní jako uložení do pole, protože při změně konkrétního prvku polynomu musíme prohledávat celou kolekci, a to vede k časové náročnosti.

Naopak uložení polynomu do pole má obrovskou výhodu oproti uložení do kolekce, a tou je přímý a rychlý přístup k jednotlivým prvkům pole. Další výhodou je typová kontrola.

Právě rychlost spolu s přímým přístupem k datům jsou nejdůležitějšími kritérii výběru datové struktury, a proto bylo použito pro práci s polynomy ukládání do polí, ale je potřeba speciální přístup k prvkům, speciální aritmetika.

3.2 Polynom1D

Polynomem1D se rozumí polynom jedné proměnné. Než se s daným polynomem bude pracovat, musí být vytvořen. K vytvoření jsou použity 3 konstruktory lišící se vstupními parametry.

Prvním typem konstruktoru je konstruktor vytvářející polynom s názvem uloženým ve vstupní proměnné *name* a s koeficienty uloženými v jednorozměrném poli *coef*.

```
public Polynom1D( String name, double[] coef ) {
    this.name = ( name );
    this.coef = new double[ coef.length ];
    System.arraycopy( coef, 0, this.coef, 0, coef.length );
}
```

Text 1: Vytvoření polynomu pomocí konstruktoru I. - zdrojový kód

Kromě tohoto konstruktoru se používá i konstruktor vytvařející polynomy totožné s polynomem, který je parametrem tohoto kontruktoru. Nový polynom se liší pouze v odlišném odkazu na sebe. Zdrojový kód je níže.

```
public Polynom1D( Polynom1D a ) {
    this.name = a.name;
    this.coef = new double[ a.getOrder() + 1 ];
    System.arraycopy( a.coef, 0, this.coef, 0, coef.length );
}
```

Text 2: Vytvoření polynomu pomocí konstruktoru II. - zdrojový kód

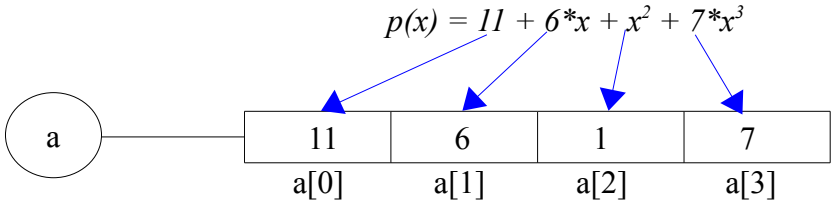
A posledním typem konstruktoru je konstruktor vytvářející polynomy řádu takového, jakou má hodnotu vstupní proměnná *rad*. Pole koeficientů je o 1 segment větší než je hodnota řádu polynomu. Polynom se naplní hodnotami 0. Jméno polynomu se implicitně nastaví na hodnotu *Druhý polynom*.

```
public Polynom1D( int rad ) {
    name = " Druhý polynom ";
    koef = new double[ rad + 1 ];
}
```

Text 3: Vytvoření polynomu pomocí konstruktoru II. - zdrojový kód

Způsob uložení do paměti

Polynom jedné proměnné je vhodné uložit do jednorozměrného pole. Uložení polynomu do jednorozměrného pole se zdá jako nejefektivnější řešení jak z hlediska kompaktnosti uložení, tak z hlediska modifikace (změny) jednotlivých členů mnohočlenu, protože každý člen je uložen na pozici v poli odpovídající jeho hodnotě exponentu. K poli se přistupuje stejně jako k objeku, pomocí odkazu. Konkrétní uložení polynomu do jednorozměrného pole ilustruje příklad níže.



3.3 Polynom2D

Polynomem2D se rozumí polynom dvou proměnných. Stejně jako u Polynomu1D, tak i v tomto případě se musí daný polynom nejdříve vytvořit a následně používat.

K vytváření polynomů dvou proměnných jsou použity konstruktory lišící se vstupními parametry.

```
public Polynom2D ( String name, double[] coef2 ) {
    int delkaPolynomu = coef2.length;
    int delka2D = 0;
    int a = 0;
    while ( delkaPolynomu > 0 ) {
      ++delka2D;
      delkaPolynomu -= delka2D;  }
    this.name = name;
    this.coef = new double[ delka2D ][];
     for ( int i = 0; i < delka2D; ++i ) {
```

Text 5: Vytvoření polynomu 2 proměnných konstruktorem I. - zdrojový kód 1. část

```
            coef[ i ] = new double [ i + 1 ]; }
        for ( int i = 0; i < getOrder() + 1; ++i ) {
            for ( int j = 0; j < coef[ i ].length; ++j ) {
                if ( a <= coef2.length - 1 ) {
                    coef[ i ][ j ] = coef2[ a ];
                    ++a; }
                else {
                    coef[ i ][ j ] = 0; } } } }
```

Text 6: Vytvoření polynomu 2 proměnných konstruktorem I. - zdrojový kód 2. část

```
public Polynom2D ( Polynom2D a ) {
    this.name = a.name;
    this.coef = new double[ a.getOrder() + 1 ][];
    for ( int i = 0; i < a.getOrder() + 1; ++i ) {
      coef[ i ] = new double [ i + 1 ]; }
    int i;
    for ( int ir = 0; ir < getOrder() + 1; ++ir ) {
      for ( int j = 0; j < coef[ ir ].length; ++j ) {
        i = ir - j;
        coef[ ir ][ j ] = a.getCoef( i, j );
      } } }
```

Text 7: Vytvoření polynomu 2 proměnných konstruktorem II. - zdrojový kód

```
public Polynom2D( int rad ) {
    name = " Druhý polynom ";
    coef = new double[ rad + 1 ][];
    for ( int i = 0; i < rad + 1; ++i ) {
      coef[ i ] = new double [ i + 1 ]; } }
```

Text 8: Vytvoření polynomu dvou proměnných konstruktorem III. - zdrojový kód

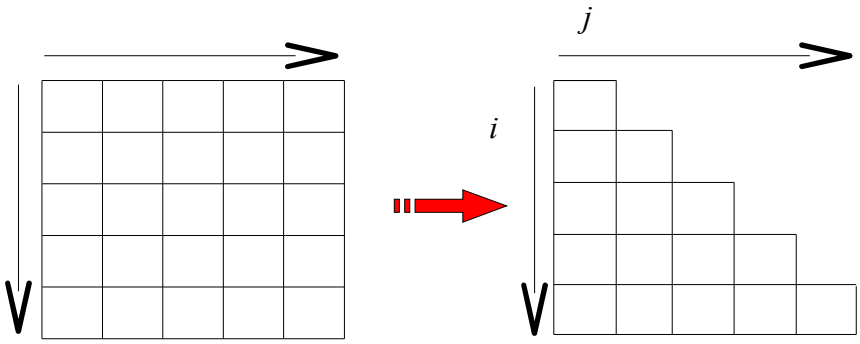
Způsob uložení do paměti

Při ukládání polynomů dvou proměnných do paměti se vychází ze stejné myšlenky jako v případě polynomů jedné proměnné, a to uložit daný polynom do pole. Protože se pracuje s polynomy dvou proměnných, musí být dané pole dvourozměrné.

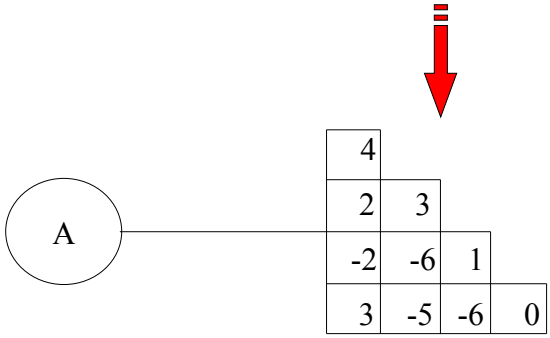
Dvourozměrné pole lze chápat jako matici. Každá pozice v poli je určena svými souřadnicemi, x – ovou a y – ovou. Jelikož se x používá i pro označení proměnných polynomu, budou se souřadnice v poli označovat písmeny i a j. Písmeno i udává řád polynomu, rozdíl $i - j$ udává mocninu proměnné x a j udává mocninu proměnné y.

Pro uložení a práci s polynomy bude dostačující využívat pouze část z celého dvourozměrného pole, a to pozice pod hlavní diagonálou a pozice na hlavní diagonále, protože velikost největší mocniny proměnné y nebude nikdy větší než řád polynomu, maximálně se budou sobě rovnat. Pokud polynom nebude obsahovat všechny proměnné daného řádu, budou na pozice chybějících proměnných uloženy nuly.

Pozicování začíná od nuly a směrem odshora dolů.



$$p(x) = 4 + 2*x + 3*y - 2*x^2 - 6*x*y + y^2 + 3*x^3 - 5*x^2*y - 6*x*y^2$$



K tomuto dvojrozměrnému poli se opět přistupuje pomocí odkazu. Přístup k jednotlivým indexům se provádí následovně: např. A[3][2], kde A je odkazem na toto pole a v hranatých závorkách jsou uvedeny souřadnice indexu pole. Na této konkrétní pozici je uloženo číslo – 6.

3.4 Polynom3D

Polynomem3D se rozumí polynom tří proměnných. I zde jsou použity tři typy konstruktorů jako u Polynomu1D a Polynomu2D.

Zdrojový kód konstruktoru vytvářející polynom s názvem uloženým ve vstupní proměnné *name* a s koeficienty uloženými v jednorozměrném poli *coef2* je níže.

```
public Polynom3D ( String name, double[] coef2) {
    int delkaPolynomu = coef2.length; int celkem = 1; int xx; int a = 0; int posun = 0;
    int pocitadlo = 0; int first = 0;
    while ( delkaPolynomu > 0 ) { ++ pocitadlo; celkem += posun; delkaPolynomu -= celkem;
        if ( first == 0 ) { posun = 2; first = 1; }
        else { ++posun; } }
    this.name = name; this.coef = new double[ pocitadlo ][ ][ ];
    for ( int i = 0; i <= getOrder(); ++i ) { coef[ i ] = new double [ i + 1 ][ ]; xx = 1;
        for ( int j = 0; j <= i; ++j ) { coef [ i ][ j ] = new double [ xx ];  ++xx; } }
    for ( int i = 0; i <= getOrder(); ++i ) {
        for ( int j = 0; j < coef[ i ].length; ++j ) {
            for ( int k = 0; k < coef[ i ][ j ].length ; ++k ) { if ( a <= coef2.length - 1 ) {
                coef[ i ][ j ][ k ] = coef2[ a ]; ++a; }
            else { coef[ i ][ j ][ k ] = 0; }}}}}
```

Text 9: Vytvoření polynomu 3 proměnných konstruktorem I.

Vytvoření polynomu jako kopii existujícího polynomu s rozdílným odkazem na sebe se provádí konstruktorem uvedeným na další stránce.

```
public Polynom3D ( Polynom3D a ) {
    int xx = 0;
    this.name = a.name;
    this.coef = new double [ a.getOrder() + 1][][];
    for ( int i = 0; i <= getOrder(); ++i ) {
       coef[ i ] = new double [ i + 1 ][ ]; xx = 1;
       for ( int j = 0; j <= i; ++j ) {
          coef [ i ][ j ] = new double [ xx ]; ++xx;}} int i; int j;
    for ( int ir = 0; ir <= getOrder(); ++ir ) {
       for ( int jr = 0; jr < coef[ ir ].length ; ++jr ) {
          for ( int k = 0; k < coef[ ir ][ jr ].length; ++k ) { i = ir – jr; j = jr - k;
             coef[ ir ][ jr ][ k ] = a.getCoef( i, j, k ); } } } }
```

Text 10: Vytvoření polynomu 3 proměnných konstruktorem II.

A posledním typem konstruktoru je konstruktor vytvářející polynom daného řádu zadaného uživatelem. Takto vytvořený polynom má všechny koeficienty nastavené na hodnotu 0 a jeho jméno je implicitně nastaveno na hodnotu *Druhý polynom*.

```
public Polynom3D ( int rad ) {
    int xx = 0;
    name = "Druhý polynom";
    coef = new double[ rad + 1][][];
    for ( int i = 0; i <= getOrder(); ++i ) {
    coef[ i ] = new double [ i + 1 ][ ];
    xx = 1;
    for ( int j = 0; j <= i; ++j ) {
       coef [ i ][ j ] = new double [ xx ];
       ++xx; } } }
```

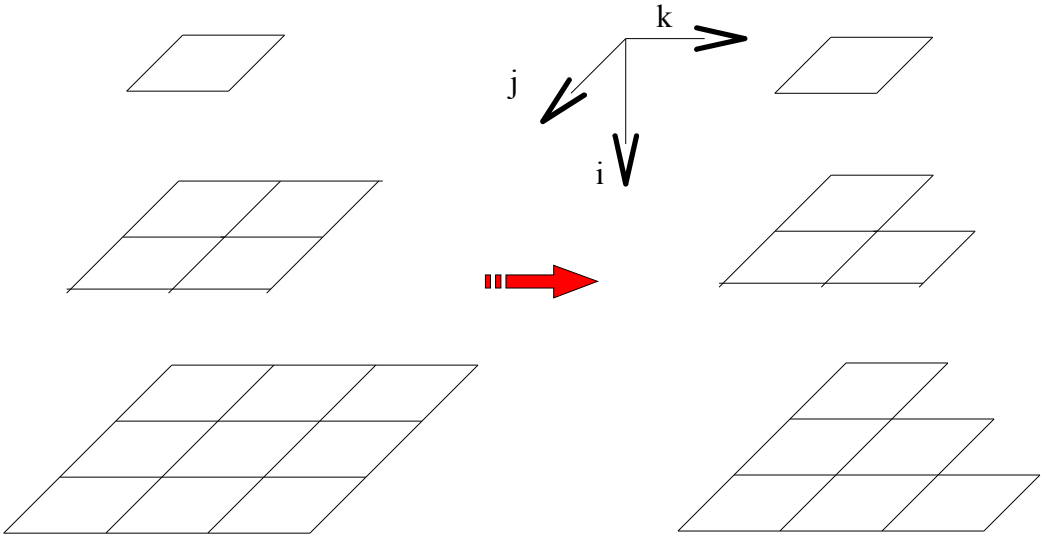
Text 11: Vytvoření polynomu 3 proměnných konstruktorem III.

Způsob uložení do paměti

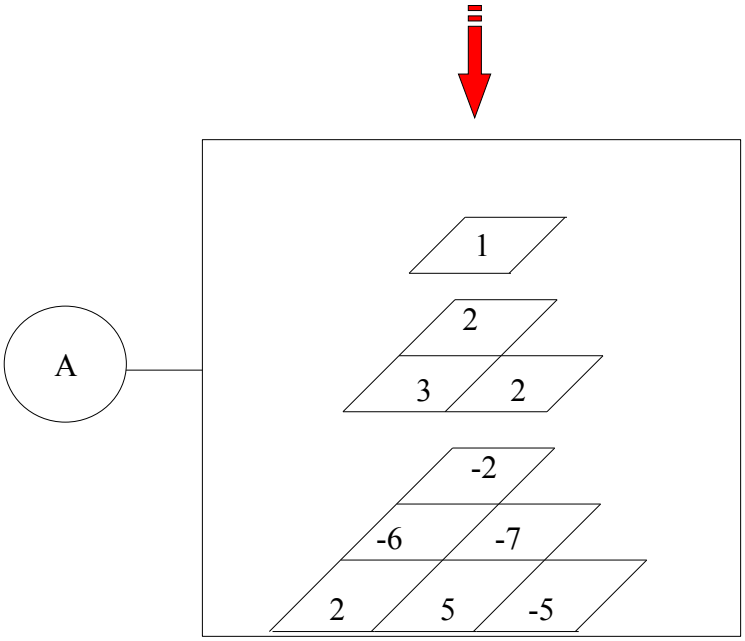
Polynom tří proměnných bude uložen v třírozměrném poli. Toto pole lze chápat jako krychli, kde je každá pozice určena svými souřadnicemi, x – ovou, y – ovou a z - ovou. Protože se tyto koeficienty používají pro označení proměnných polynomu, budou se souřadnice v poli označovat písmeny i, j a k. Písmeno i udává řád polynomu, rozdíl $i - j$ udává mocninu proměnné x, rozdíl $j - k$ udává mocninu proměnné y a k udává mocninu proměnné z.

Pro uložení a práci s polynomy bude dostačující využívat pouze část z celého třírozměrného pole, a to pozice splňující podmínku, že velikost největších mocnin proměnných x, y a z nebude nikdy větší než řád polynomu, maximálně se budou sobě rovnat. Pokud polynom nebude obsahovat všechny proměnné daného řádu, budou na pozice chybějících proměnných uloženy nuly.

Pozicování začíná od nuly a směrem odshora dolů. Pro větší přehlednost a pochopení daného způsobu uložení je třírozměrné pole rozkresleno do jednotlivých úrovní představující řád polynomu tří proměnných.



$$p(x) = 1 + 2*x + 3*y + 2*z - 2*x^2 - 6*x*y - 7*x*z + 2*y^2 + 5*y*z - 5*z^2$$



3.5 Součet polynomů

Polynom jedné proměnné

Součet dvou polynomů v programu lze provést dvěma způsoby. Rozdíl je pouze v tom, kde se daný výsledek součtu bude nacházet. Pokud bude výsledek součtu uložen v polynomu, který tento součet volal, tak se součet volá metodou polynomu **add(Polynom1D)**, kde parametrem je odkaz na druhý polynom vstupující do součtu. Zdrojový kód vypadá následovně:

p1.add(p2);

Výsledek součtu bude vypadat takto: *p1 = p1 + p2*, tzn. hodnota polynomu se nezachová, ale překryje se výsledkem součtu obou polynomů.

Bude – li potřeba zachovat oba polynomy a výsledek uložit do jiného polynomu, nezbývá nic jiného, než zavolat metodu třídy **add(Polynom1D, Polynom1D)**, kde parametry jsou odkazy na oba polynomy vstupující do součtu.

Polynom1D p3 = Polynom1D.add(p1, p2);

Výsledkem tohoto součtu je odkaz na nový polynom délky dané součtem těchto dvou dílčích polynomů obsažených v objektech, konkrétně: *p3 = p1 + p2* .

Příklad součtu dvou polynomů

$p1(x) = 2 + 3*x + 5*x^2$ (*name* = první)

$p2(x) = 7 + 2,6*x + 2*x^2 + 15*x^3 + 0,4*x^4$ (*name* = druhý)

Řešení:

a) *p1.add(p2);*

výsledek: $p1(x) = 9 + 5,6*x + 7*x^2 + 15*x^3 + 0,4*x^4$

b) *Polynom1D p3 = Polynom1D.add(p1, p2);*

výsledek: $p3(x) = 9 + 5,6*x + 7*x^2 + 15*x^3 + 0,4*x^4$

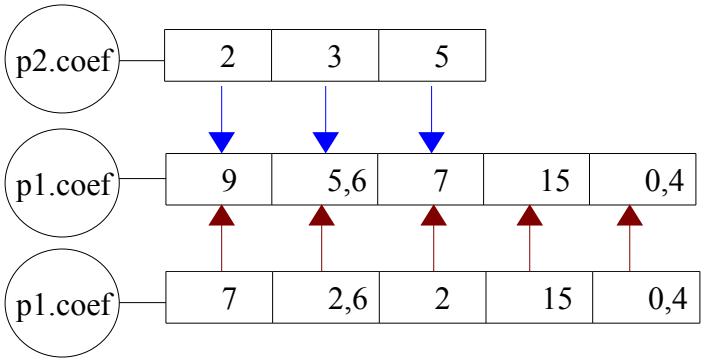
add a) V prvním kroku se pomocí metody **getOrder()** zjistí, který ze dvou polynomů je vyššího řádu. V případě, že polynom volající metodu **add(Polynom1D)** je vyššího řádu, dojde ke změně členské proměnné *name* (uloží se do ní řetězec Suma = názvy obou polynomů) a přes cyklus for se volá metoda **addCoef(int, Polynom1D)** provádějící přičtení polynomu nižšího řádu k polynomu vyššího řádu. Přičtení se provádí postupně od koeficientu a_0, do koeficientu udávajícího řád polynomu, včetně. Z důvodu vysvětlení tohoto postupu je přejmenován polynom *p1* na polynom *p2* a obráceně, protože jinak by neplatilo ,že polynom volající metodu **add(Polynom1D)** je vyššího řádu.

```
public int getOrder() {

    return( coef.length -1 );

}
```

Text *12: Metoda vracející řád polynomu - zdrojový kód*

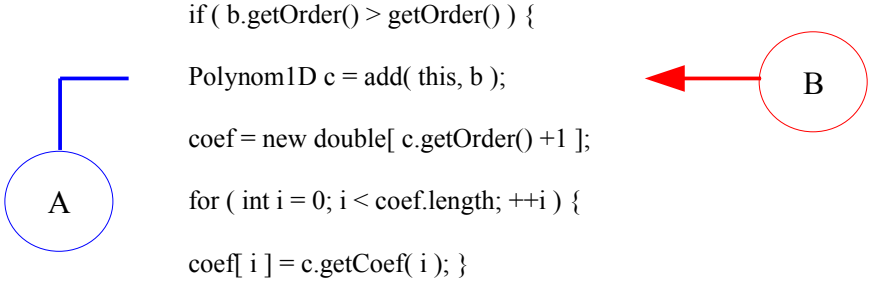


Pokud ale polynom volající metodu **add(Polynom1D)** je nižšího řádu než druhý polynom vstupující do součtu, volá se metoda třídy **add(Polynom1D, Polynom1D).** V ní se zjišťuje opět řád polynomu, protože se tato metoda může volat samostatně bez nutnosti volání metody polynomu, a proto v ní musí být obsažena kontrola zjištění vyššího řádu polynomu.

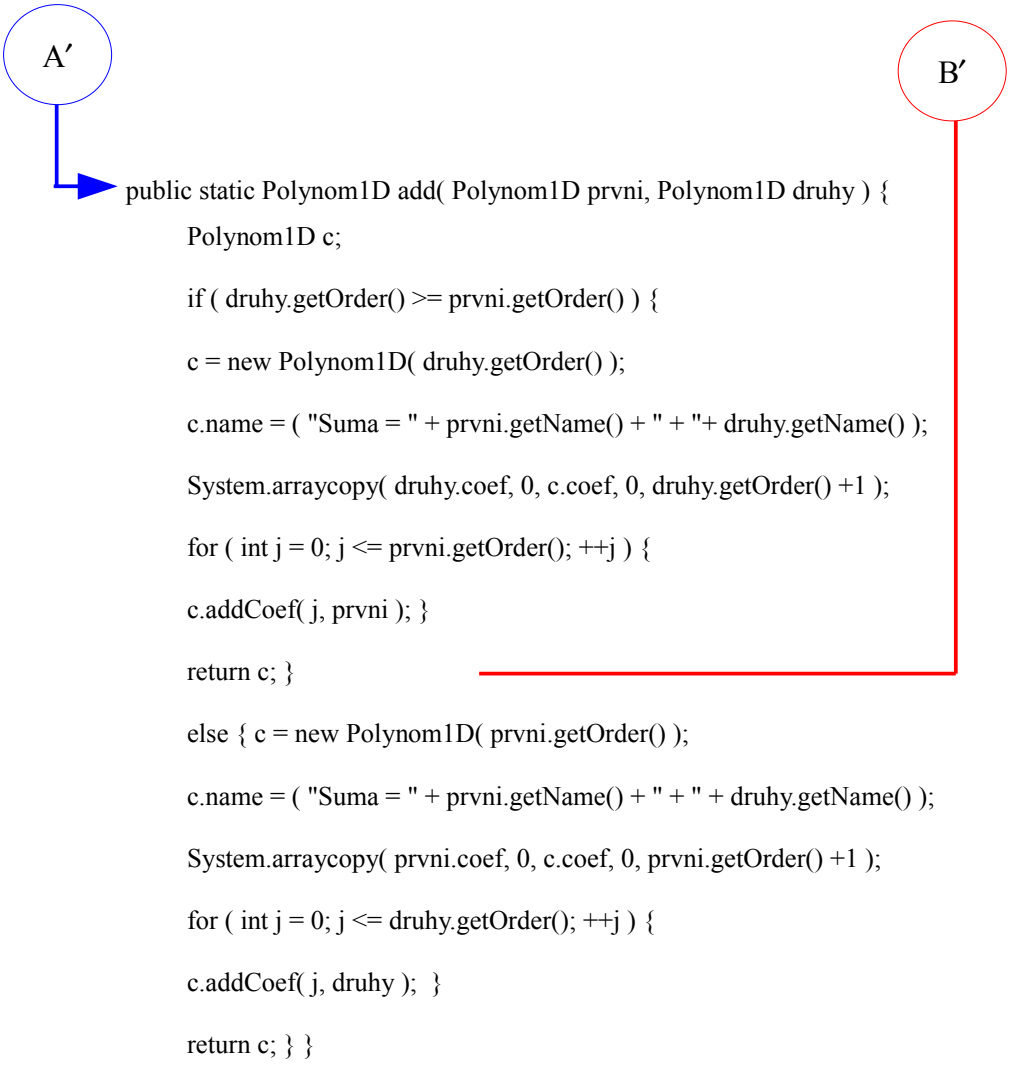
Odkaz na metodu **add(Polynom1D, Polynom1D)** je uložen v metodě polynomu a přes něj se bude přistupovat k polynomu obsahující součet polynomů a členskou proměnou *name*. V metodě třídy se vytvoří pomocí konstruktoru nový polynom obsahující polynom stejného řádu jakého je polynom vyššího řádu. K tomu se použije konstruktor s parametrem typu integer. Metodou **arraycopy()** se překopírují hodnoty do nově vzniklého jednorozměrného pole.

Metodou **addCoef(int,Polynom1D)** se přičte polynom nižšího řádu k tomuto polynomu (algoritmus je stejný jako na předchozí straně). Členská proměnná *name* se také změní. Tímto je součet hotový a klíčovým slovem **return** se vrací odkaz na součet polynomů do metody polynomu. Protože je polynom *p1* nižšího řádu než součet, musí se vytvořit v polynomu *p1* nové pole koeficienů, do kterého se uloží výsledek. Voláním metody **getCoef(int)** se nastaví koeficienty do tohoto pole.

```
public void add( Polynom1D b ) {
        if ( b.getOrder() > getOrder() ) {
        Polynom1D c = add( this, b );
        coef = new double[ c.getOrder() +1 ];
        for ( int i = 0; i < coef.length; ++i ) {
        coef[ i ] = c.getCoef( i ); }
        name = c.getName(); }
        else {
        name = ( "Suma = " + getName() + " " + b.getName() );
        for ( int i = 0; i == b.getOrder(); ++i ) {
        addCoef( i, b ); } } }
```

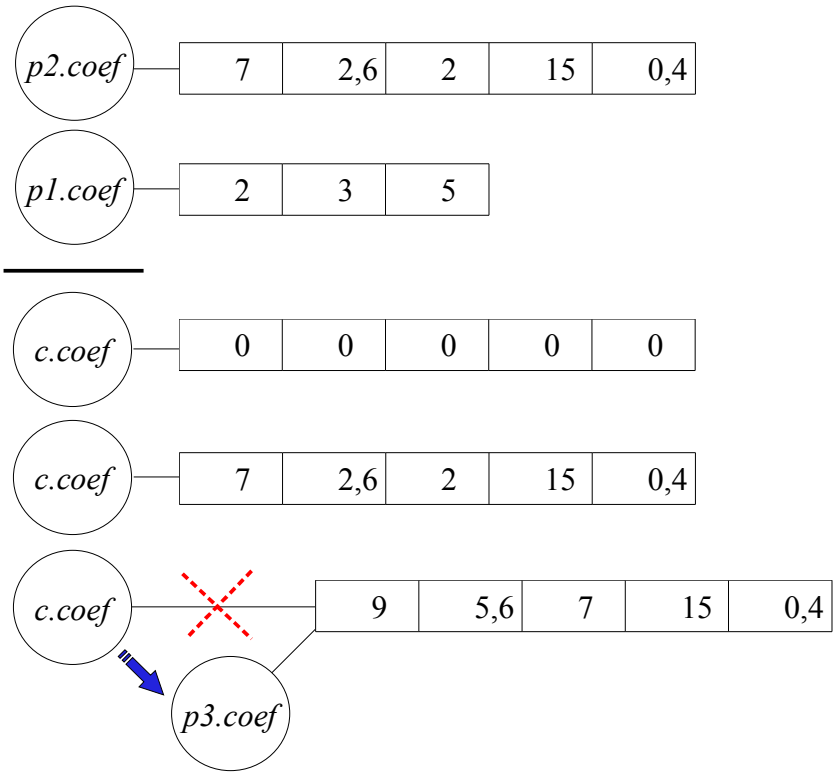


Text 13: Součet polynomů pomocí metody polynomu - zdrojový kód



Text 14: Součet polynomů pomocí metody třídy - zdrojový kód

add b) Odkaz *p3* je odkazem na polynom vzniklý v metodě třídy **add(Polynom1D, Polynom1D).** V této metodě se opět porovnává, který z polynomů je vyššího řádu. Zavolá se konstruktor vytvářející nový polynom vyššího řádu. Hodnoty koeficientů vyššího řádu se překopírují do nově vzniklého polynomu a metodou **addCoef(int, Polynom1D)** se přičte i druhý polynom do tohoto polynomu. Klíčovým slovem **return** se předá odkaz na polynom součtu polynomů do *p3*.




```
double[] k = { 2, 3, 5 };
double[] b = { 7, 2.6, 2, 15, 0.4 };
Polynom1D p1 = new Polynom1D( "první", k );
Polynom1D p2 = new Polynom1D( "druhý", b );
Polynom1D p3 = Polynom1D.add(p1,p2);
p1.print();
p2.print();
p3.print();
```

Text 15: Výpočet součtu polynomu pomocí metody třídy + tisk - zdrojový kód

Polynom dvou proměnných

Součet dvou polynomů dvou proměnných lze provést voláním metody polynomu **add(Polynom2D)** nebo metodou třídy **add(Polynom2D, Polynom2D)**. Rozdíl mezi těmito metodami, resp. v jejich použití je vysvětlen v kapitole 3.5.

Příklad součtu dvou polynomů

$p1(x, y) = 2 + 4,7*x - 5*y - 11*x^2 + 6,8*x*y - 1,4*y^2 + 3*x^3 - x^2*y - 1,2*x*y^2$

$p2(x, y) = 22 + 3,1*x - 5,7*x^2 + 4*x^3$

Řešení:

a) *p1.add(p2);*

výsledek: $p1(x, y) = 24 + 7,8*x - 10,7*y - 7*x^2 + 6,8*x*y - 1,4*y^2 + 3*x^3 - x^2*y - 1,2*x*y^2$

b) *Polynom2D p3 = Polynom2D.add(p1, p2);*

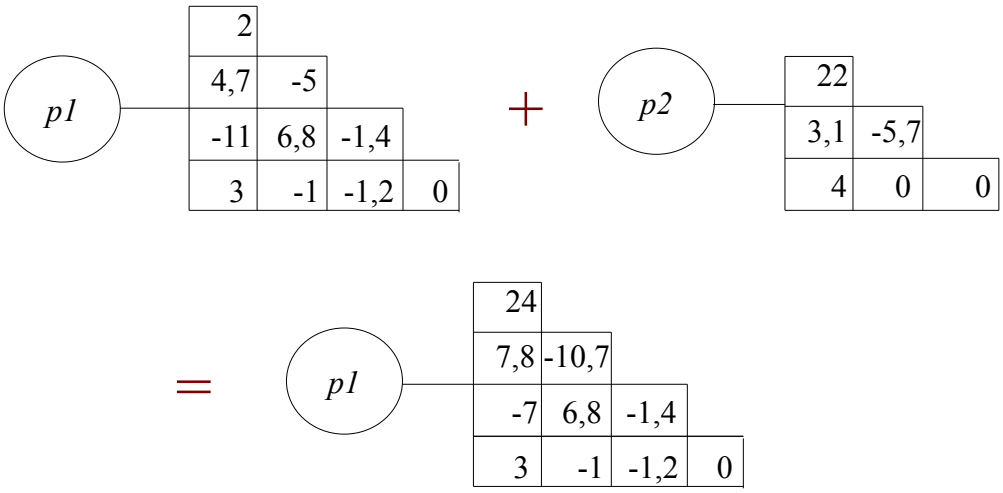
výsledek: $p3(x, y) = 24 + 7,8*x - 10,7*y - 7*x^2 + 6,8*x*y - 1,4*y^2 + 3*x^3 - x^2*y - 1,2*x*y^2$

add a) V prvním kroku se podmínkou if zjišťuje, zda – li polynom volající metodu **add(Polynom2D)** je vyššího řádu.

Tato podmínka se provádí z důvodu nutnosti volat metodu třídy jen tehdy, pokud není splněna, protože se musí vytvořit nový polynom vyššího řádu, a právě v metodě třídy je konstruktor zajišťující vytvoření tohoto polynomu.

Ze zadání je patrné, že polynom *p1* je vyššího řádu, konkrétně třetího. Pomocí dvou cyklů procházejících polynom nižšího řádu (*p2*) se volá metoda **addCoef(int, int, Polynom2D)** provádějící přičtení koeficientů tohoto polynomu k polynomu *p1*. Metodou **setName()** se uloží do proměnné *name* polynomu *p1* řetězec s hodnotou: Suma = název polynomu *p1* + název polynomu *p2*.

V případě, že by polynom *p1* byl nižšího řádu, součet polynomů by se provedl v metodě třídy, která by byla volána po vyhodnocení podmínky if. Provedení součtu metodou třídy bude vysvětleno v dalším textu.




```
public void add( Polynom2D b ) { int i, ir, j;
    if ( b.getOrder() > getOrder() ) {
        Polynom2D c = add ( this, b );
        name = c.getName();
        coef = new double[ c.getOrder() +1 ][];
        for ( i = 0; i <= c.getOrder(); ++i ) {
                coef[ i ] = new double [ i + 1 ]; }
        for ( ir = 0; ir <= getOrder(); ++ir ) {
                for ( j = 0; j < coef[ ir ].length; ++j ) {
                        i = ir - j;
                        coef[ ir ][ j ] = c.getCoef( i, j ); }}}
    else {
        for ( ir = 0; ir <= b.getOrder() ; ++ir ) {
                for ( j = 0; j < b.coef[ ir ].length; ++j ) {
                        i = ir – j;
                        addCoef( i, j, b );} }
                        setName( "Suma = " + getName() + " " + b.getName() ); } }
```

Text 16: Součet polynomů pomocí metody polynomu - zdrojový kód

add b) V této metodě se nejdříve zjišťuje, který polynom je vyššího řádu. Konstruktorem se vytvoří nový polynom řádu stejného jako má polynom vyššího řádu. Do něho se pomocí dvou for cyklů zkopírují hodnoty koeficientů polynomu vyššího řádu a dalšími dvěma for cykly, kdy se volá metoda **addCoef(int, int, Polynom2D)**, se přičtou k těmto koeficientům koeficienty nižšího polynomu, nastaví se proměnná *name* a klíčovým slovem **return** se vrací odkaz na součet polynomů *p1* a *p2*.

Protože zdrojový kód obsahující součet polynomů pomocí metody třídy je dlouhý, bude zde uvedena pouze část s lichým vyhodnocením podmínky if, což odpovídá i realitě, tzn. polynom *p1* je vyššího řádu než polynom *p2*.

```
private static Polynom2D add( Polynom2D prvni , Polynom2D druhy ) {
    Polynom2D c;
    int i, ir;
zde je stejná konstrukce zdrojového kódu jako níže s rozdílem, že místo polynomu první je druhý
else {
    c = new Polynom2D( prvni.getOrder() );
            c.name = ( "Suma = " + prvni.getName() + " + " + druhy.getName() );
            for ( ir = 0; ir <= prvni.getOrder() ; ++ir ) {
                    for ( int j = 0; j < prvni.coef[ ir ].length; ++j ) {
                    i = ir - j;
                    c.setCoef( i, j, prvni.coef[ ir ][ j ] ); } }
    for ( ir = 0; ir <= druhy.getOrder() ; ++ir ) {
        for ( int j = 0; j < druhy.coef[ ir ].length; ++j ) {
                    i = ir - j;
            c.addCoef( i, j, druhy ); } } return c; } }
```

Text 17: Součet polynomů pomocí metody třídy - zdrojový kód

Polynom tří proměnných

Součet dvou polynomů tří proměnných lze opět provést stejně jako v předchozích případech voláním metody polynomu nebo voláním metody třídy. Názvy metod provádějících součet polynomů tří proměnných jsou **add(Polynom3D)** a **add (Polynom3D, Polynom3D)**. Rozdíl mezi nimi je popsán v kapitole 3.5.

Příklad součtu dvou polynomů

$p1(x, y, z) = -2,4 + 5*x - 6,2*y - 3,1*z + 12*x^2 - 4*x*y + 2,4*x*z + 14*y^2$

$p2(x, y, z) = -5 + 4,2*x - 4,4*y - 2,7*z + 0,1*x^2$

Řešení:

a) *p1.add(p2);*

výsledek: $p1(x, y, z) = -7,4 + 9,2*x - 10,6*y - 5,8*z + 12,1*x^2 - 4*x*y + 2,4*x*z + 14*y^2$

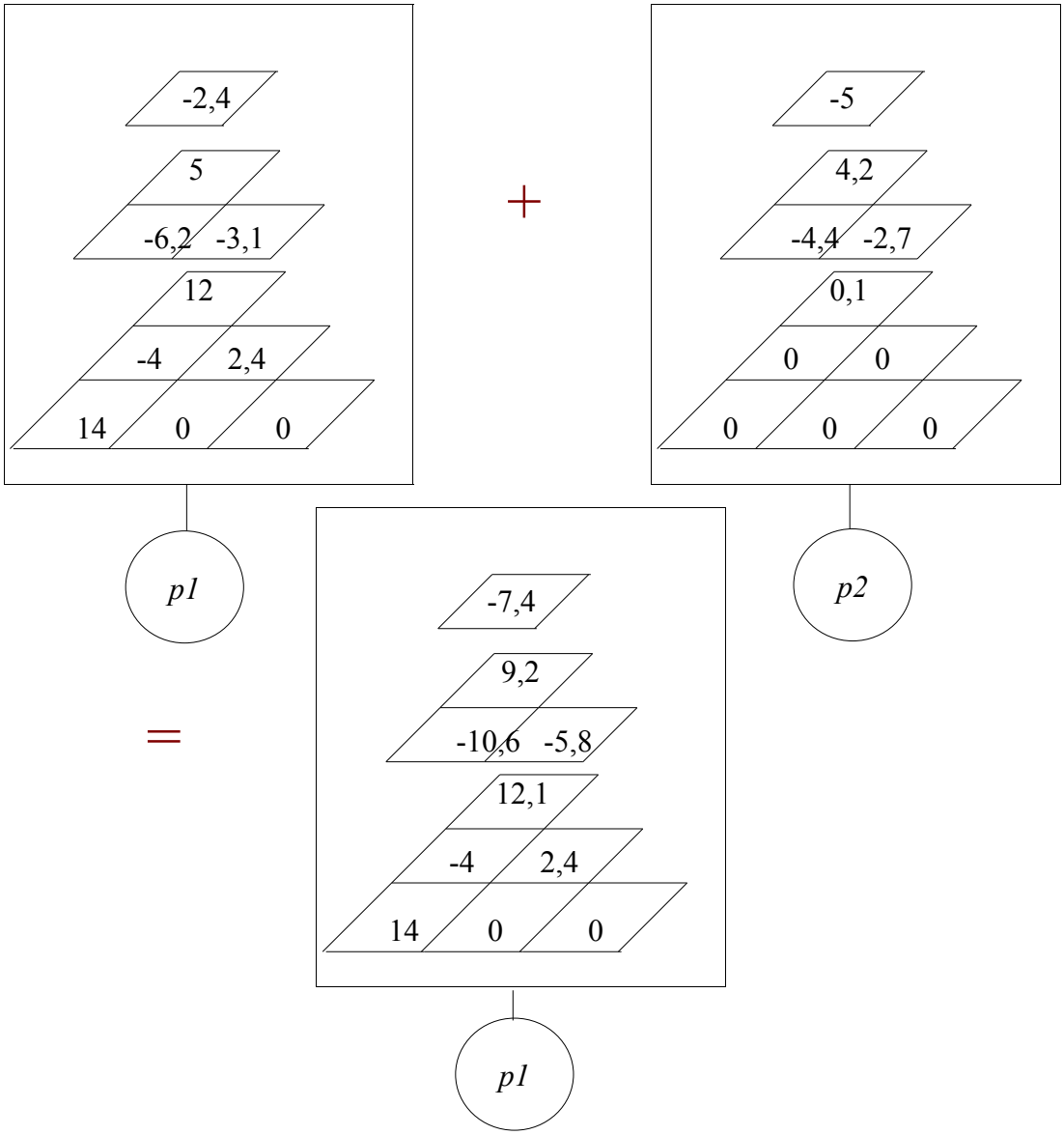
b) *Polynom3D p3 = Polynom3D.add(p1, p2);*

výsledek: $p3(x, y, z) = -7,4 + 9,2*x - 10,6*y - 5,8*z + 12,1*x^2 - 4*x*y + 2,4*x*z + 14*y^2$

add a) V prvním kroku se opět zjišťuje, zda – li polynom volající metodu **add(Polynom3D)** je vyššího řádu stejně jako tomu bylo u polynomu dvou proměnných.

Důvodem je volání statické metody pouze tehdy, když je druhý polynom vstupující do součtu (vstupní parametr metody) vyššího řádu. Je – li polynom volající metodu součtu vyššího řádu, metodou **addCoef(int, int, int, Polynom3D)** se postupně přičtou koeficienty polynomu nižšího řádu ke koeficientům polynomu vyššího řádu.

Ze zadání je zřejmé, že polynom volající metodu součtu je vyššího řádu, tudíž se k provedení součtu nepoužije statická metoda, ale pouze metoda **addCoef(int, int, int, Polynom3D)** umístěná ve třech for cyklech sloužících k procházení polynomu. Po přičtení všech koeficientů se metodou **setName()** nastaví jméno polynomu *p1* na hodnotu: Suma = název polynomu *p1* + název polynomu *p2*.




```
public void add( Polynom3D b ) { int i, ir, j, jr, k, xx;
    if ( b.getOrder() > getOrder() ) { Polynom3D c = add( this, b);
      name = c.getName(); coef = new double [ c.getOrder() + 1 ][][];
      for ( i = 0; i <= getOrder(); ++i ) { coef[ i ] = new double [ i + 1 ][ ]; xx = 1;
        for ( j = 0; j <= i; ++j ) {
          coef [ i ][ j ] = new double [ xx ];   ++xx; } }
      for ( ir = 0; ir <= getOrder(); ++ir ) { for ( jr = 0; jr < coef[ ir ].length; ++jr ) {
```

Text 18: Součet polynomů pomocí metody polynomu - zdrojový kód 1. část

```
for ( k = 0; k < coef[ ir ][ jr ].length; ++k ) { i = ir – jr; j = jr – k;
    coef[ ir ][ jr ][ k ] = c.getCoef( i, j, k ); } }}} else {
    for( ir = 0; ir <= b.getOrder(); ++ir ) { for( jr = 0; jr < b.coef[ ir ].length; ++jr ) {
      for( k = 0; k < b.coef[ ir ][ jr ].length; ++k ) { i = ir - jr – k; j = jr – k; addCoef( i, j, k, b );} }}
      setName( "Suma = " + getName() + " " + b.getName() ); }}
```

Text 20: Součet polynomů pomocí metody polynomu - zdrojový kód 2. část

add b) V metodě třídy se zjišťuje, který z polynomů je vyššího řádu. Po tomto zjištění, se vytvoří pomocí konstruktoru nový polynom jako kopie polynomu vyššího řádu s jiným odkazem na sebe a metodou **addCoef(int, int, int, Polynom3D)** se přičtou stejné koeficienty polynomu nižšího řádu ke koeficientům polynomu vyššího řádu. Metodou **setName()** se nastaví jméno nově vzniklého polynomu na hodnotu: Suma = název polynomu *p1* + název polynomu *p2* a klíčovým slovem **return** se vrátí odkaz na tento polynom do *p3*.

```
private static Polynom3D add ( Polynom3D prvni , Polynom3D druhy ) {
    Polynom3D c; int i, ir, j, jr, k;
    if ( druhy.getOrder() >= prvni.getOrder() ) { c = new Polynom3D( druhy );
      c.name = ( "Suma = " + prvni.getName() + " " + druhy.getName() );
      for( ir = 0; ir <= prvni.getOrder(); ++ir ) {
        for( jr = 0; jr < prvni.coef[ ir ].length; ++jr ) {
        for( k = 0; k < prvni.coef[ ir ][ jr ].length; ++k ) { i = ir – jr; j = jr - k;
     c.addCoef( i, j, k, prvni ); } } }return c; }else{ c = new Polynom3D( prvni );
      c.name = ( "Suma = " + prvni.getName() + " " + druhy.getName() );
      for( ir = 0; ir <= druhy.getOrder(); ++ir ) {
        for( jr = 0; jr < druhy.coef[ ir ].length; ++jr ) {
          for( k = 0; k < druhy.coef[ ir ][ jr ].length; ++k ) { i = ir - jr ; j = jr - k;
            c.addCoef( i, j, k, druhy ); } } } return c; }}
```

Text 19: Součet polynomů pomocí metody třídy - zdrojový kód

3.6 Součin polynomů

Polynom jedné proměnné

Součin polynomů lze provést pomocí metody třídy **product(Polynom1D, Polynom1D)** nebo pomocí metody polynomu **product(Polynom1D)**. Samotný součin se provádí pouze v metodě třídy a pokud je volán metodou polynomu, tak se v této metodě konkrétního polynomu volá vždy metoda **product(Polynom1D, Polynom1D)**.

Výsledek součinu je uložen buď v polynomu volající metodu **product(Polynom1D)** nebo je uložen v novém polynomu, na který ukazuje odkaz součinu dvou polynomů.

Příklad součinu dvou polynomů

$p1(x) = 2 + 3*x + 5*x^2$ (*name* = první)

$p2(x) = 7 + 2,6*x + 2*x^2 + 15*x^3 + 0,4*x^4$ (*name* = druhý)

Řešení:

a) *p1.product(p2);*

výsledek: $p1(x) = 14 + 26,2*x + 46,8*x^2 + 49*x^3 + 55,8*x^4 + 76,2*x^5 + 2*x^6$

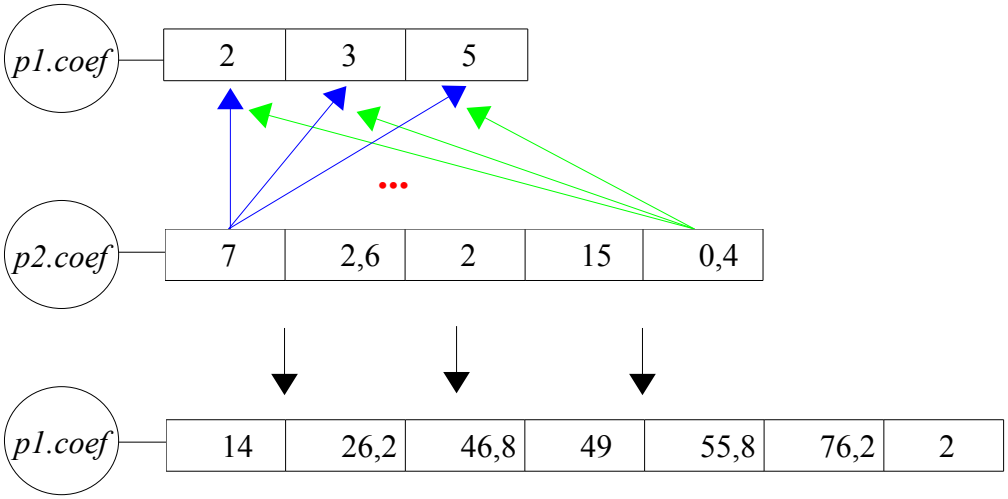
b) *Polynom1D p3 = Polynom1D.product(p1, p2);*

výsledek: $p3(x) = 14 + 26,2*x + 46,8*x^2 + 49*x^3 + 55,8*x^4 + 76,2*x^5 + 2*x^6$

add a) V prvním kroku se volá metoda třídy **product(Polynom1D, Polynom1D)** s parametry tvořící odkazy polynomů, kde prvním parametrem je odkaz na polynom, ze kterého se volá metoda polynomu (*p1*) a druhým je odkaz na polynom *p2*. V metodě třídy se vytvoří nový polynom řádu stejného jako je součet řádů obou vstupujících polynomů do této metody. Pomocí dvou cyklů for se prochází oba polynomy a provádí se součin, který se následně ukládá do nového polynomu.

Metodou polynomu **setName()** se nastaví proměnná *name* polynomu vzniklého součinem polynomů a klíčovým slovem **return** se vrací odkaz na výsledek součinu polynomů do metody polynomu *p1*.

V ní se poté nastaví metodou **getName()** proměnná *name* polynomu *p1* na hodnotu proměnné *name* polynomu vytvořeného v metodě třídy. Protože součinem polynomů vznikl polynom vyššího řádu, musí se vytvořit nové pole koeficientů polynomu *p1*, a to se provede cyklem for, ve kterém se volá metoda **getCoef(int)**.




```
public void product( Polynom1D b ) {
    Polynom1D c = product ( this, b );
    name = c.getName();
    coef = new double[c.getOrder() + 1];
    for ( int i = 0; i <= getOrder(); i++ ) {
    this.coef[ i ] = c.getCoef( i ); } }
```

A

B′

Text 21: Součin polynomů pomocí metody polynomu - zdrojový kód

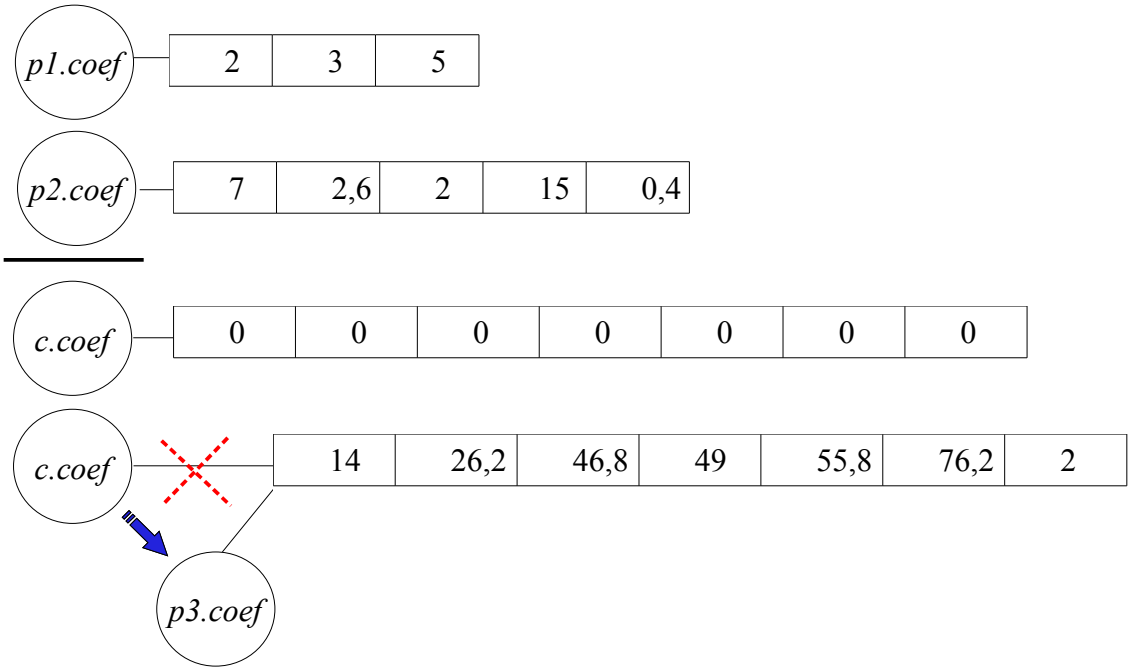
A′

```
public static Polynom1D product ( Polynom1D prvni, Polynom1D druhy ) {
        double a = 0;
        Polynom1D c = new Polynom1D ( prvni.getOrder() + druhy.getOrder() );
        for ( int i = 0; i <= druhy.getOrder(); i++ ) {
                for ( int j = 0; j <= prvni.getOrder(); j++ ) {
                a = prvni.coef[ j ] * druhy.coef[ i ];
                c.coef[ i + j] = c.coef[ i + j] + a; }}
    c.setName("Součin polynomů " + prvni.getName() + " a " + druhy.getName() );
    return c;}
```

B

Text 22: Součin polynomů pomocí metody třídy - zdrojový kód

add b) Odkaz *p3* je odkazem na polynom vzniklý součinem polynomů *p1* a *p2*, přičemž oba polynomy jsou zachovány. Postup je stejný jako v předchozím případě součinu polynomů s tím rozdílem, že se na začátku nevolá metoda třídy.



Polynom dvou proměnných

Pro součin polynomů dvou proměnných je k dispozici metoda polynomu **product(Polynom2D)** a metoda třídy **product(Polynom2D, Polynom2D)**. Samotný součin se provádí pouze v metodě třídy a pokud je volán metodou polynomu, tak se v této metodě konkrétního polynomu volá vždy metoda **product(Polynom2D, Polynom2D)**.

Součin, obecně, se vypočítá vynásobením všech sčítanců jednoho polynomu se všemi sčítanci druhého polynomu.

Příklad součinu dvou polynomů

$p1(x, y) = 1 + 2{,}4*x + 5*y + 7{,}4*x^2 + 2{,}8*x*y + 5*y^2$

$p2(x, y) = 11 + 7{,}9*x + 8{,}1*x^2$

Řešení:

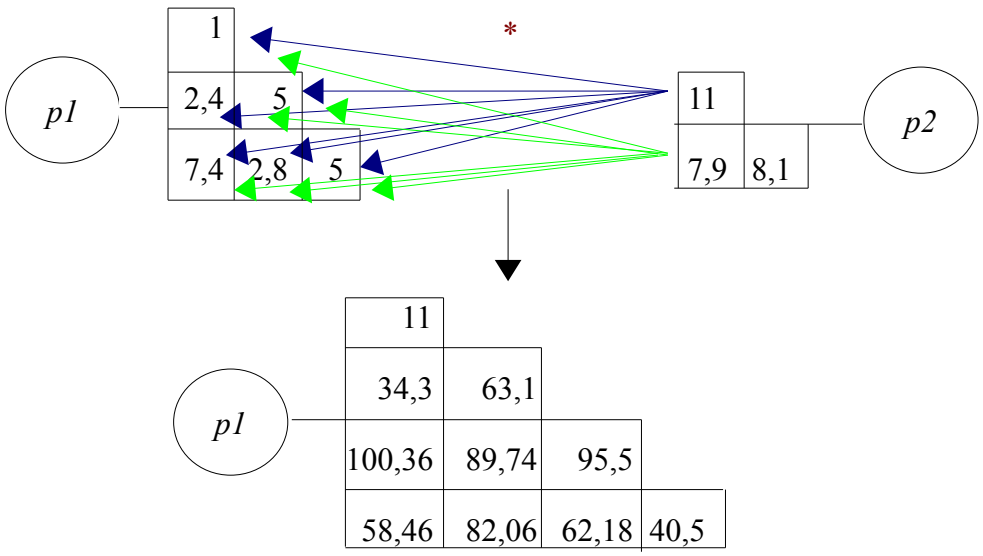
a) *p1.product(p2);*

výsledek: *p1(x, y) =11 + 34,3*x + 63,1*y + 100,36*x^2 + 89,74*x*y + 95,5*y^2 + 58,46*x^3 + 82,06*x^2*y + 62,18*x*y^2 + 40,5* y^3*

b) *Polynom2D p3 = Polynom2D.product(p1, p2);*

výsledek: *p3(x, y) =11 + 34,3*x + 63,1*y + 100,36*x^2 + 89,74*x*y + 95,5*y^2 + 58,46*x^3 + 82,06*x^2*y + 62,18*x*y^2 + 40,5* y^3*

add a) Při výpočtu součinu metodou polynomu se v této metodě volá metoda třídy **product(Polynom2D, Polynom2D)** s parametry **this** a odkazem na druhý polynom vstupující do součinu. V metodě třídy se pomocí konstruktoru vytvoří nový polynom řádu daného součtem řádů obou polynomů. Čtyřmi for cykly se procházejí všechny pozice obou polynomů, provede se součin a výsledek se uloží na odpovídající pozice nově vzniklého polynomu. *Name* se nastaví na hodnotu: Součin polynomů + název prvního polynomu + název druhého polynomu a klíčovým slovem **return** se vrátí odkaz na polynom obsahující součin polynomů do metody polynomu, ze které byla metoda třídy volána. V metodě polynomu se vytvoří nový polynom stejného řádu jako je polynom vytvořený v metodě třídy a zkopírují se do něho hodnoty koeficientů.



```
public void product( Polynom2D b ) {
        int i, ir, j;
        Polynom2D c = product ( this, b );
        name = c.getName();
        coef = new double[c.getOrder() + 1][];
        for ( i = 0; i <= c.getOrder(); ++i ) {
                coef[ i ] = new double [ i + 1 ];}
        for ( ir = 0; ir <= getOrder(); ++ir ) {
                for ( j = 0; j < coef[ ir ].length; ++j ) {
                        i = ir - j;
                        coef[ ir ][ j ] = c.getCoef( i, j ); } }}
```

A

B′

Text 23: Součin polynomů pomocí metody polynomu - zdrojový kód

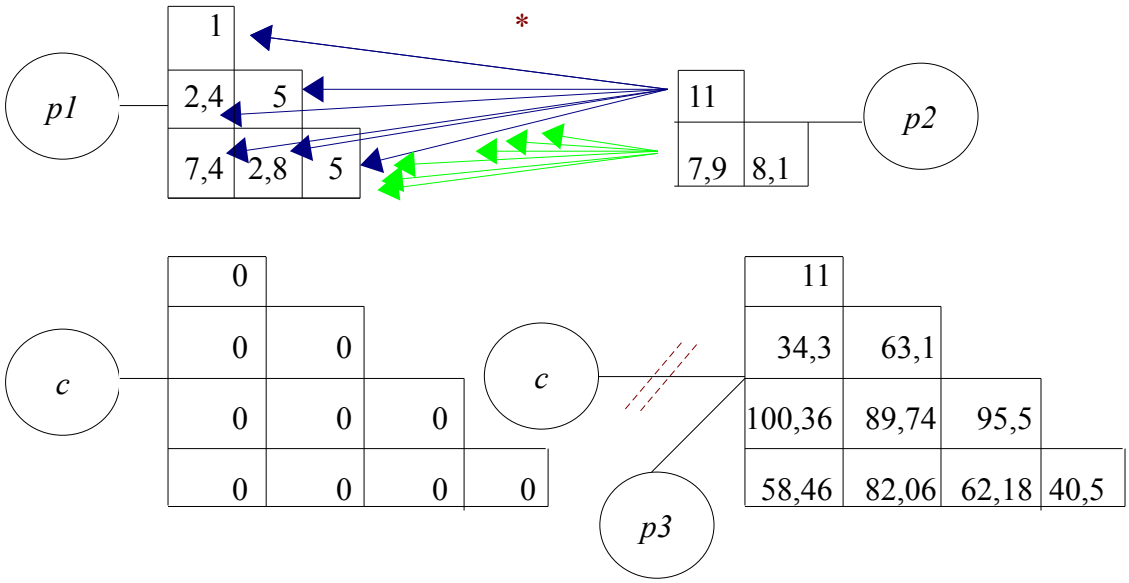
A′

```
public static Polynom2D product( Polynom2D prvni, Polynom2D druhy ) {
int i, j, i2, j2; double a;
Polynom2D c = new Polynom2D( prvni.getOrder() + druhy.getOrder() );
   for( i = 0; i <= druhy.getOrder(); ++i ) {
        for ( j = 0; j < druhy.coef[ i ].length; ++j ) {
                for ( i2 = 0; i2 <= prvni.getOrder(); ++i2 ) {
                        for ( j2 = 0; j2 < prvni.coef[ i2 ].length; ++j2 ) {
                                a = prvni.coef[ i2 ][ j2 ] * druhy.coef[ i][ j ];
                                c.coef[ i + i2 ][ j + j2 ] = c.coef[ i + i2 ][  j + j2 ] + a; } } } }
        c.setName("Součin polynomů " + prvni.getName() + " a " + druhy.getName() );
                                return c; }
```

B

Text 24: Součin polynomů pomocí metody třídy - zdrojový kód

add b) Počítání součinu statickou metodou je už popsáno v části **add a)** s tím rozdílem, že se tato metoda nemusí volat z metody polynomu a odkaz na výsledek součinu se vrací do p3. Oba polynomy jsou zachovány beze změny a výsledkem je nový polynom s odkazem p3.



Polynom tří proměnných

Součin polynomů tří proměnných se provádí pomocí dvou metod. Metodou polynomu **product(Polynom3D)** nebo metodou třídy **product(Polynom3D, Polynom3D)**. Samotný součin se vždy provádí v metodě třídy a pokud je volán metodou polynomu, tak se v této metodě konkrétního polynomu volá vždy statická metoda **product(Polynom2D, Polynom2D)**, ve které se součin polynomů vypočítá.

Příklad součinu dvou polynomů

$p1(x, y, z) = 2 - 1,4*x + 6,7*y - 4*z$

$p2(x, y, z) = 7,5 + 5,5*x - 7,9*y + 4,2*z$

Řešení:

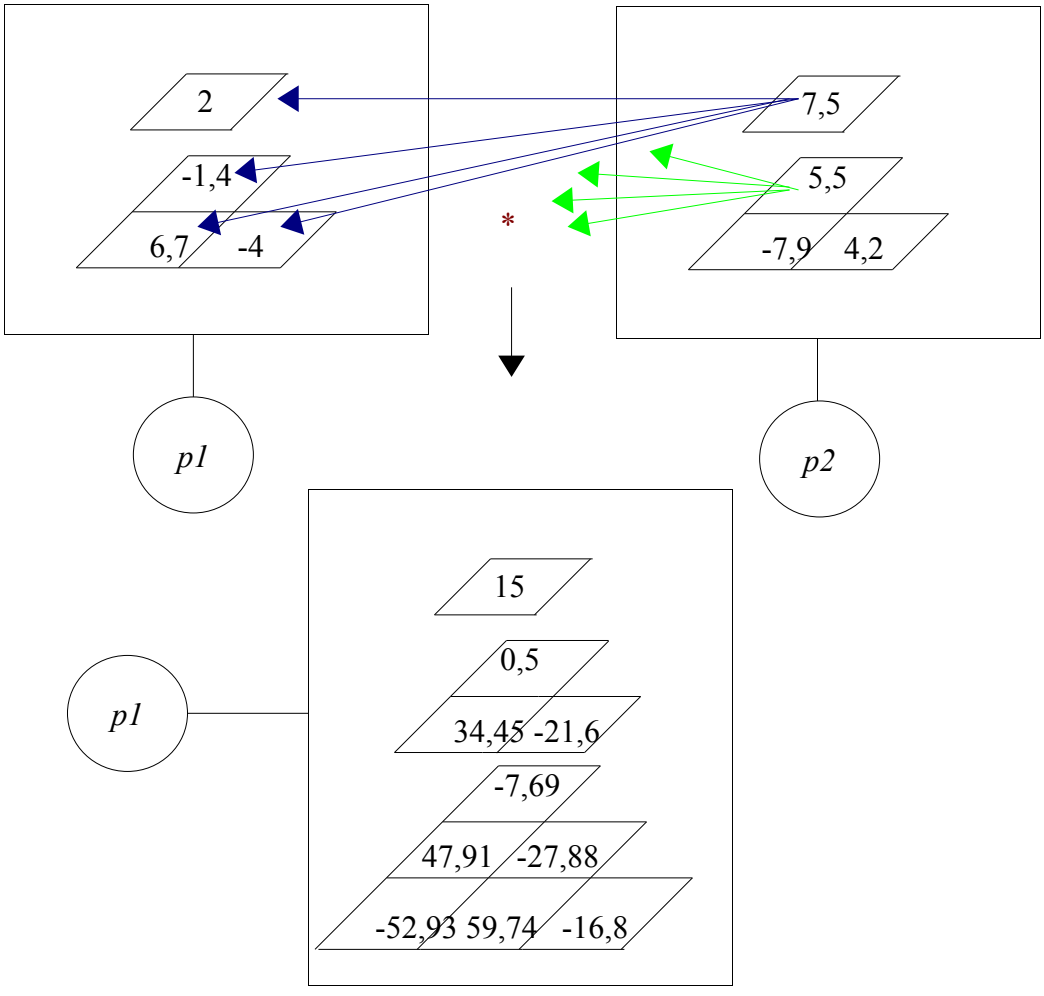
a) *p1.product(p2);*

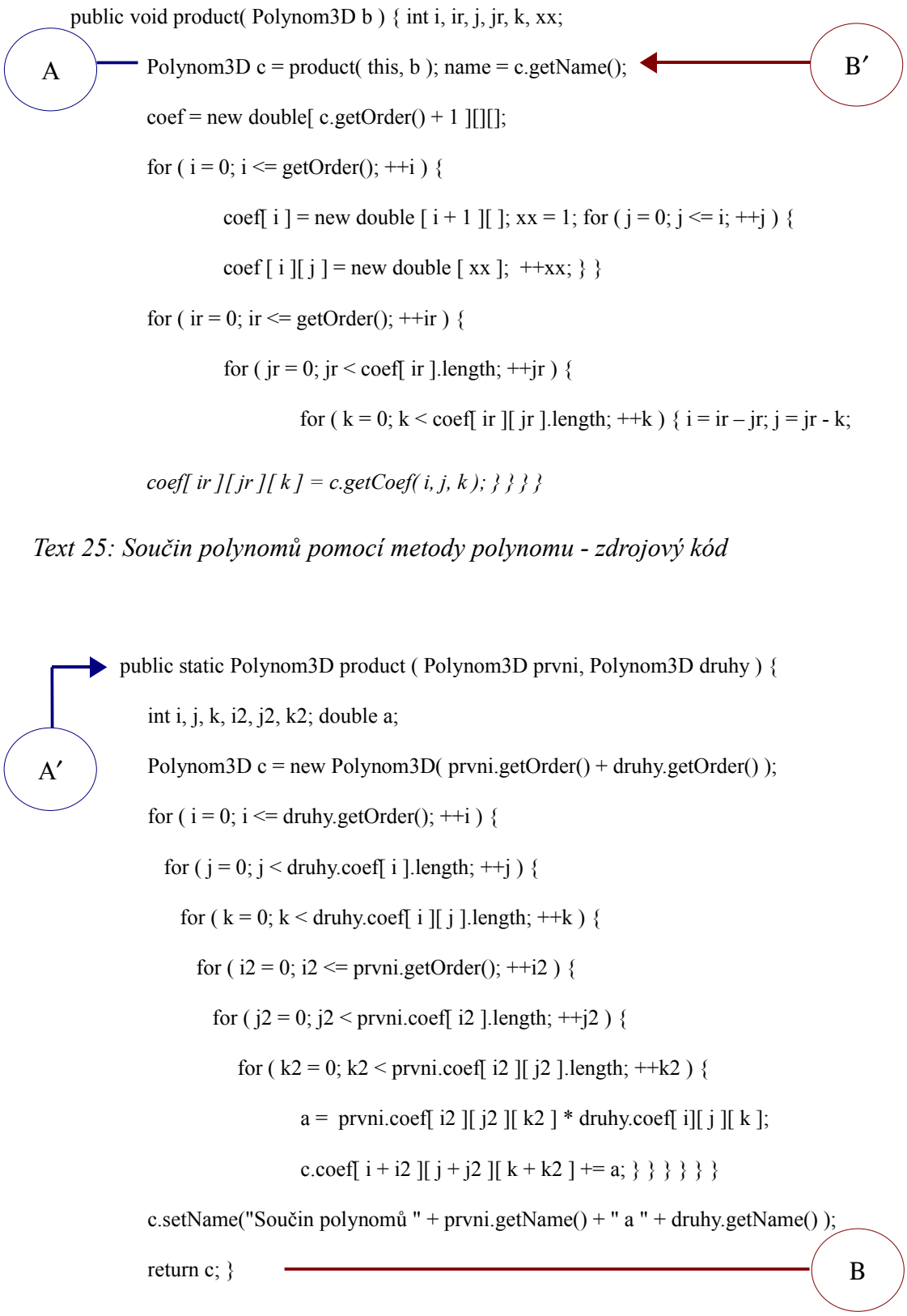
výsledek: $p1(x, y, z) = 15 + 0,5*x + 34,45*y - 21,6*z - 7,69*x^2 + 47,91*x*y - 27,88*x*z - 52,93*y^2 + 59,74*y*z - 16,8*z^2$

b) *Polynom3D p3 = Polynom3D.product(p1, p2);*

výsledek: *p3(x, y, z) =15 + 0,5*x + 34,45*y – 21,6*z – 7,69*x² + 47,91*x*y – 27,88*x*z – 52,93*y² + 59,74*y*z – 16,8*z²*

add a) V metodě polynomu se volá statická metoda **product(Polynom3D, Polynom3D)** s parametry **this** a odkazem na druhý polynom vstupující do součinu. Ve statické metodě se konstruktorem vytvoří nový polynom řádu daného součtem řádů obou vstupujících polynomů. Pomocí šesti for cyklů se procházejí postupně všechny pozice obou polynomů a provádí se součin, který se následně ukládá na nové pozice nově vzniklého polynomu. Proměnná n*ame* se nastaví na hodnotu: Součin polynomů + název prvního polynomu + název druhého polynomu a klíčovým slovem **return** se vrátí odkaz na polynom obsahující součin polynomů do metody polynomu, ze které byla metoda třídy volána. V metodě polynomu se vytvoří nový polynom stejného řádu jako je polynom vytvořený v metodě třídy a zkopírují se do něho hodnoty koeficientů.

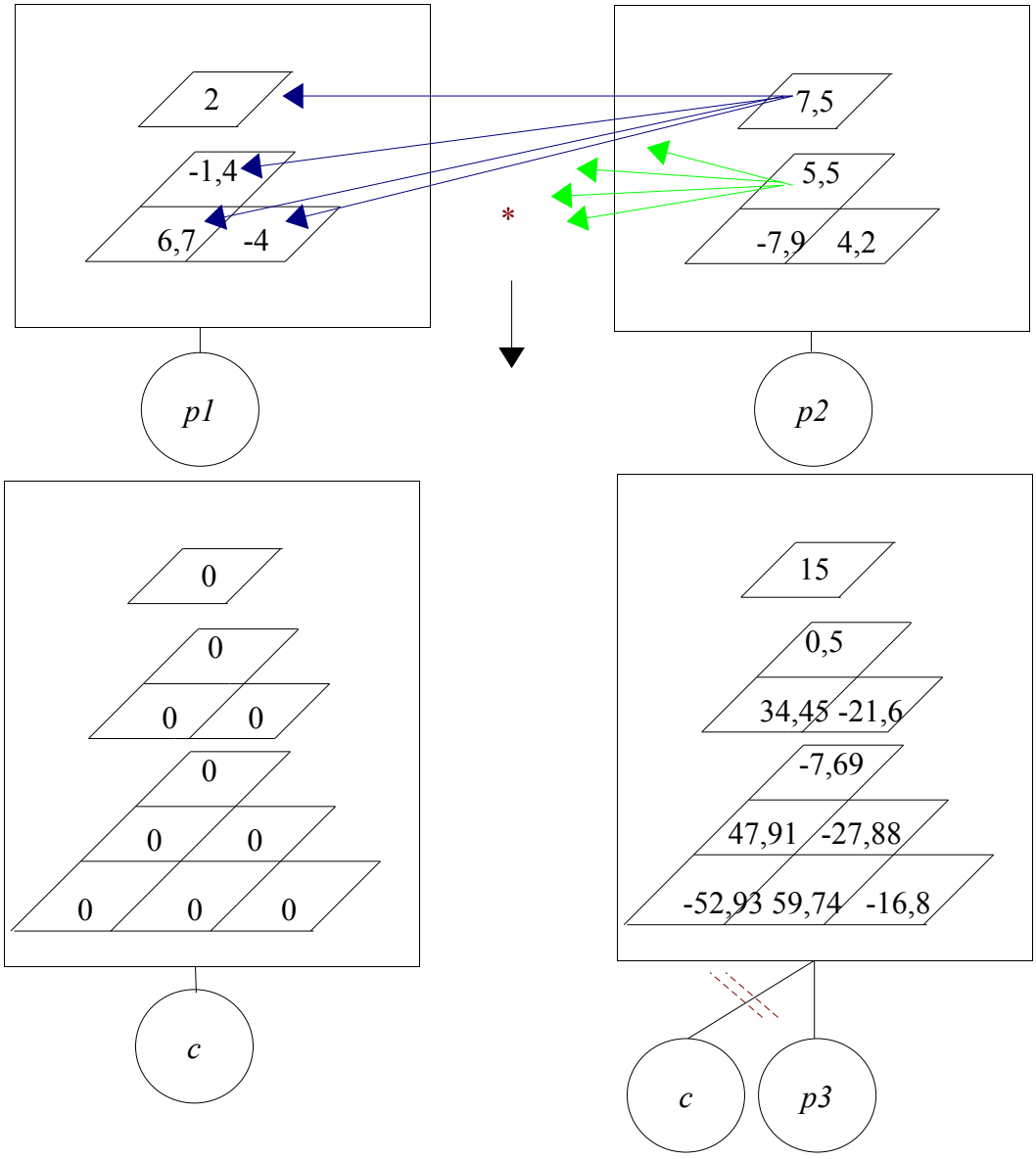




Text 25: Součin polynomů pomocí metody polynomu - zdrojový kód

Text 26: Součin polynomů pomocí metody polynomu - zdrojový kód

add b) Počítání součinu statickou metodou je popsáno v části **add a)** s tím rozdílem, že se tato metoda nemusí volat z metody polynomu a odkaz na výsledek součinu se vrací do *p3*. Oba polynomy jsou zachovány beze změny a výsledkem je nový polynom s odkazem *p3*.



3.7 Parciální derivace polynomu

Polynom jedné proměnné

Parciální derivace je určitým zobecněním derivace pro funkce více proměnných. Je zde použita z důvodu pozdější návaznosti na derivace polynomů dvou a tří proměnných.

Příklad parciální derivace polynomu

$p1(x) = 11 + 8,4*x + 9*x^2 + 1,3*x^3 + 4*x^4 + 6,7*x^5 + 14*x^6 + 3.8*x^7$

Řešení:

výsledek: $p1(x) = 8,4 + 18*x + 3,9*x^2 + 16*x^3 + 33,5*x^4 + 84*x^5 + 26,6*x^6$

Při výpočtu parciální derivace polynomu se postupuje tak, že se tento polynom nejdříve uloží do jednorozměrného pole na pozice odpovídající jeho koeficietům.

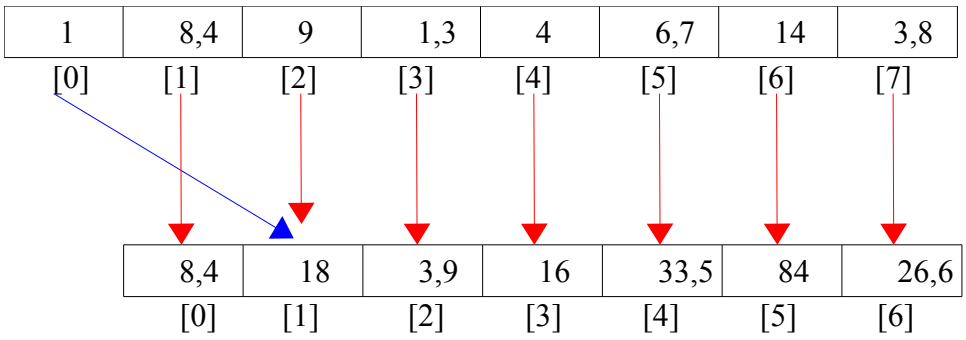
Protože parciální derivací polynomu *n* – tého řádu je polynom (*n* – 1). řádu, vytvoří se nové jednorozměrné pole délky *n* odpovídající polynomu (*n* – 1). tého řádu (viz. str. 10).

V zde uvedeném příkladě se derivuje polynom 7. řádu uložený v jednorozměrném poli délky 8. Zderivovaný polynom bude 6. řádu a bude uložen v poli o délce 7.

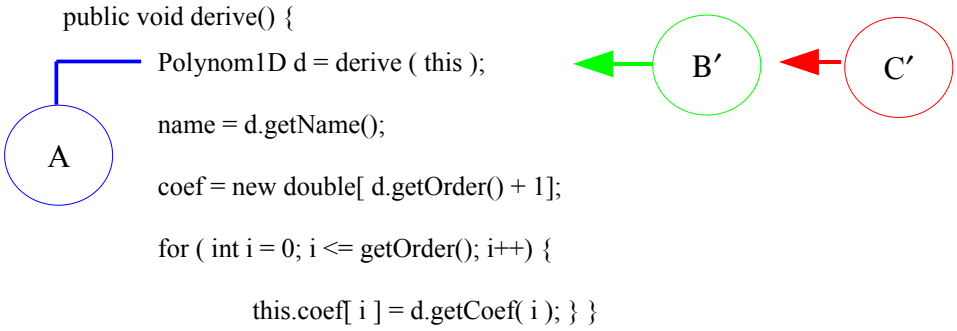
Obecný algoritmus výpočtu je následující: koeficient na *n* - té pozici je vynásoben s indexem této pozice a výsledek se uloží na pozici *n* – 1, např. derivací členu $3,8*x^7$ bude $26,6*x^6$, protože číslo 3,8 uložené na pozici 7 se vynásobí s tímto indexem pozice a uloží se na pozici 6 udávající řád polynomu.

Komplikace je při derivaci 0. členu. Při aplikování algoritmu výpočtu by se musel uložit tento člen na pozici -1, která je mimo rozsah pole a program by se musel ošetřit vyvoláním vyjímky nebo by havaroval. Řešením je, že se 0. člen uloží na pozici 1. Uložením na 1. pozici se nic nestane, protože tato hodnota bude překryta výsledkem derivace třetího členu.

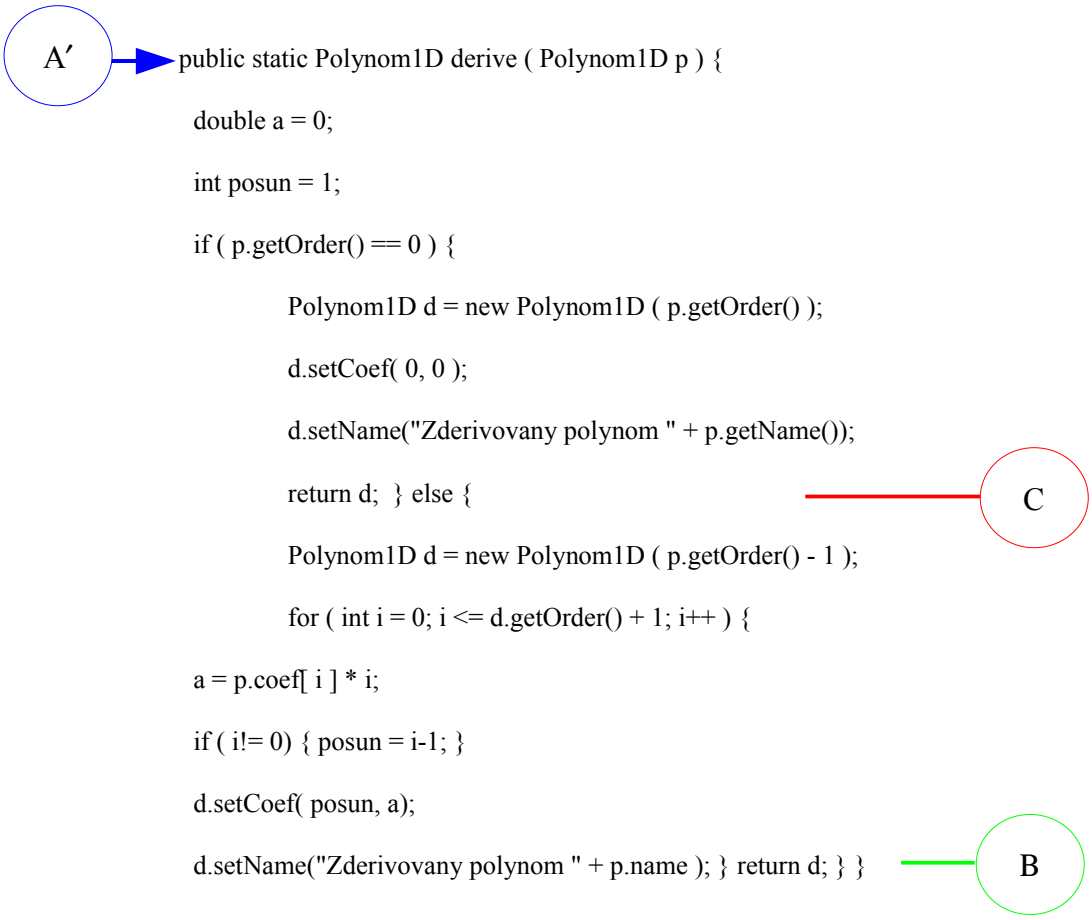
Může se ale stát, že se bude derivovat polynom 0. řádu a uložení na pozici 1 není řešením, protože by se nejednalo o parciální derivaci polynomu. Z toho důvodu se ve statické metodě pokaždé zjišťuje jakého je polynom řádu a podle zjištění řádu se program větví. Je – li polynom 0. řádu, tak se nově vytvořené pole nezmenšuje o 1, ale má stejnou velikost, tzn. je délky 1 a vloží se do indexu 0 číslo 0.



Parciální derivaci je možné zpočítat opět dvěma způsoby, buďto voláním metody polynomu nebo voláním metody třídy. Rozdíl oproti výpočtu součtu polynomů je ten, že pokud se počítá derivace polynomu pomocí metody polynomu **derive()**, vždy je následně volána metoda třídy **derive (Polynom1D)** s parametrem **this,** v ní se vypočítá derivace a pomocí **return** se vrací pouze odkaz do metody polynomu.



Text 27: Parciální derivace polynomu metodou polynomu - zdrojový kód



Text 28: Parciální derivace polynomu metodou třídy - zdrojový kód

Polynom dvou proměnných

Parciální derivací polynomu podle proměnné *x* se rozumí derivace jednotlivých sčítanců obsahujících proměnnou *x*. Pokud sčítanec obsahuje i proměnnou *y*, považuje se *y* za konstantu a nederivuje se.

Parciální derivací polynomu podle proměnné *y* se rozumí derivace jednotlivých sčítanců obsahujících proměnnou *y*. Pokud sčítanec obsahuje i proměnnou *x*, považuje se *x* se za konstantu a nederivuje se.

Příklad parciální derivace polynomu

$p(x,y) = 4 + 5{,}2*x + 8*y + 11*x^2 + 9*x*y + 3{,}5*y^2$

Řešení:

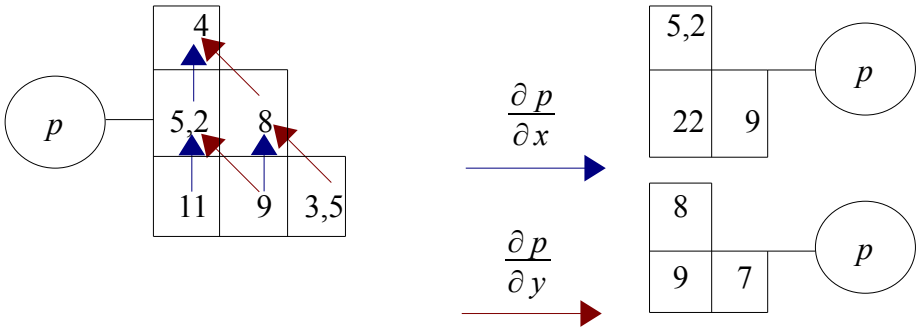
výsledek: $\frac{\partial p}{\partial x} = \underline{5{,}2 + 22*x + 9*y}$

$\frac{\partial p}{\partial y} = \underline{8 + 9*x + 7*y}$

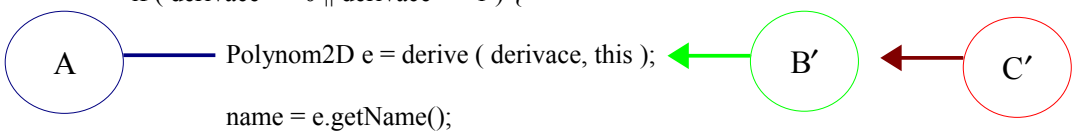
Při výpočtu parciální derivace polynomu dvou proměnných se postupuje tak, že se nejdříve zjistí podle hodnoty vstupního parametru metody polynomu **derive(int)** nebo metody třídy **derive(int, Polynom2D)** , podle které proměnné se derivuje daný polynom. Má – li vstupní parametr hodnotu 0, derivuje se polynom podle proměnné *x*, pokud má hodnotu 1, derivuje se polynom podle proměnné *y*. Je – li hodnota vstupního parametru jiná než zde uvedené, vyvolá se automaticky vyjímka a uživateli se zobrazí na obrazovku chybové hlášení.

Při parciální derivaci polynomu podle proměnné *x* se vytvoří nový polynom řádu o 1 nižšího než je daný polynom. Provede se samotný výpočet derivace, metodou **setCoef(int, int, double)** se nastaví koeficienty v novém polynomu a do proměnné *name* se vloží hodnota: Zderivovaný polynom + název polynomu. Klíčovým slovem **return** se vrátí odkaz buďto do metody polynomu nebo na místo, ze kterého byla volána metoda třídy. V parciální derivaci polynomu podle proměnné *y* je postup obdobný, rozdíl je pouze v jiném algoritmu výpočtu derivace.

Pokud je polynom 0 – tého řádu, parciální derivace podle obou proměnných je rovna 0. Výpočet parciální derivace se provádí pouze v metodě třídy.



```
public void derive( int derivace ) throws Exception {
        throw new Exception ("chybné zadání čísla uživatelem!");int i, ir, j;
        try {
        if ( derivace == 0 || derivace == 1 ) {
                Polynom2D e = derive ( derivace, this );
                name = e.getName();
                coef = new double[ e.getOrder() + 1 ][];
                for ( i = 0; i <= e.getOrder(); ++i ) {
                        coef[ i ] = new double [ i + 1 ];  }
                for ( ir = 0; ir <= e.getOrder(); ++ir ) {
                        for ( j = 0; j < coef[ ir ].length; ++j ) {
                                i = ir - j;
                                coef[ ir ][ j ] = e.getCoef( i, j ); }}} }
else { catch ( Exception e ) } }
        e.printStackTrace();
        { catch ( "chybné zadání čísla uživatelem!" ) }
```



Text 29: Parciální derivace polynomu metodou polynomu - zdrojový kód

```
public static Polynom2D derive ( int derivace, Polynom2D p ) { int i, ir, j, k;
if ( derivace == 0 )
{if ( p.getOrder() == 0 ) { Polynom2D d = new Polynom2D ( p.getOrder() );
    d.setCoef( 0, 0, 0);
    d.setName("Zderivovaný polynom " + p.getName() + "(dx)" );return d;}
    else {double a; Polynom2D d = new Polynom2D ( p.getOrder() - 1 );
    for ( ir = 0; ir <= p.getOrder() ; ++ir ) {
       for ( j = 0; j < p.coef[ ir ].length; ++j ) {i = ir - j;
          a = p.coef[ ir ][ j ] * i; if ( ir == 0) { d.setCoef( i, j, 0 ); }
          else {i -= 1; if ( i >= 0) {d.setCoef( i, j, a ); }
          else { if( j <= d.getOrder() ) {d.setCoef( 0, j, a );}}}}}
    d.setName("Zderivovaný polynom " + p.getName() + "(dx)" );
    return d;   }} else { if ( derivace == 1 ) { if ( p.getOrder() == 0 ) {
    Polynom2D e = new Polynom2D ( p.getOrder() );
    e.setCoef( 0, 0, 0);
    e.setName("Zderivovaný polynom " + p.getName() + " " + "(dy)"  );return e;}
    else { double a; Polynom2D e = new Polynom2D ( p.getOrder() - 1 );
    for ( ir = 0; ir <= p.getOrder() ; ++ir ) { for ( j = 0; j < p.coef[ ir ].length; ++j ) {
          i = ir – j; a = p.coef[ ir ][ j ] * j;  k = j - 1;
          if ( ir == 0 ) { e.setCoef( i, j, 0 ); }
          else {if ( ir == j ) { e.setCoef(i, k, a); }
             else { if ( j == 0 ) {i -= 1; e.setCoef( i, 0, 0);}
                   else {e.setCoef( i, k, a );}}}}}
    e.setName("Zderivovaný polynom " + p.getName() + "(dy)" );
    return e; }}else { System.out.printl(" Chyba zadání, tato volba není možná!!! ");
    Polynom2D e = new Polynom2D ( 10 );return e;  }}}
```

A′ → public static Polynom2D derive (int derivace, Polynom2D p) { int i, ir, j, k;

C → return d;   }} else { if (derivace == 1) { if (p.getOrder() == 0) {

B → Polynom2D e = new Polynom2D (10);return e; }}}

Parciální derivace polynomu prvního a vyšších řádů podle x

Parciální derivace polynomu prvního a vyšších řádů podle y

Text 30: Parciální derivace polynomu metodou třídy - zdrojový kód

Polynom tří proměnných

Polynom tří proměnných lze parciálně derivovat podle proměnných *x*, *y* a *z*. Výpočet lze provádět opět pomocí 2 metod. Pokud je výpočet prováděn metodou polynomu **derive(int)**, tak se vždy v této metodě volá metoda třídy **derive(int, Polynom3D)**, ve které se provede samotný výpočet a do metody polynomu se vrátí odkaz na zderivovaný polynom.

To, podle jaké proměnné se bude polynom parciálně derivovat, je závislé na hodnotě proměnné *derivace*, která je vstupním parametrem u obou typů metod. Má – li *derivace* hodnotu 0, derivuje se polynom podle proměnné *x*. Má – li hodnotu 1, derivuje se polynom podle proměnné *y* a s hodnotou 2 proměnné *derivace* se polynom derivuje podle proměnné *z*.

Je – li hodnota vstupního parametru jiná než zde uvedené, vyvolá se automaticky vyjímka a uživateli se zobrazí na obrazovku chybové hlášení.

Příklad parciální derivace polynomu

$p(x,y,z) = 4 + 5{,}2*x + 8*y + 11*z + 5{,}2*x^2 + 8*x*y + 11*x*z$

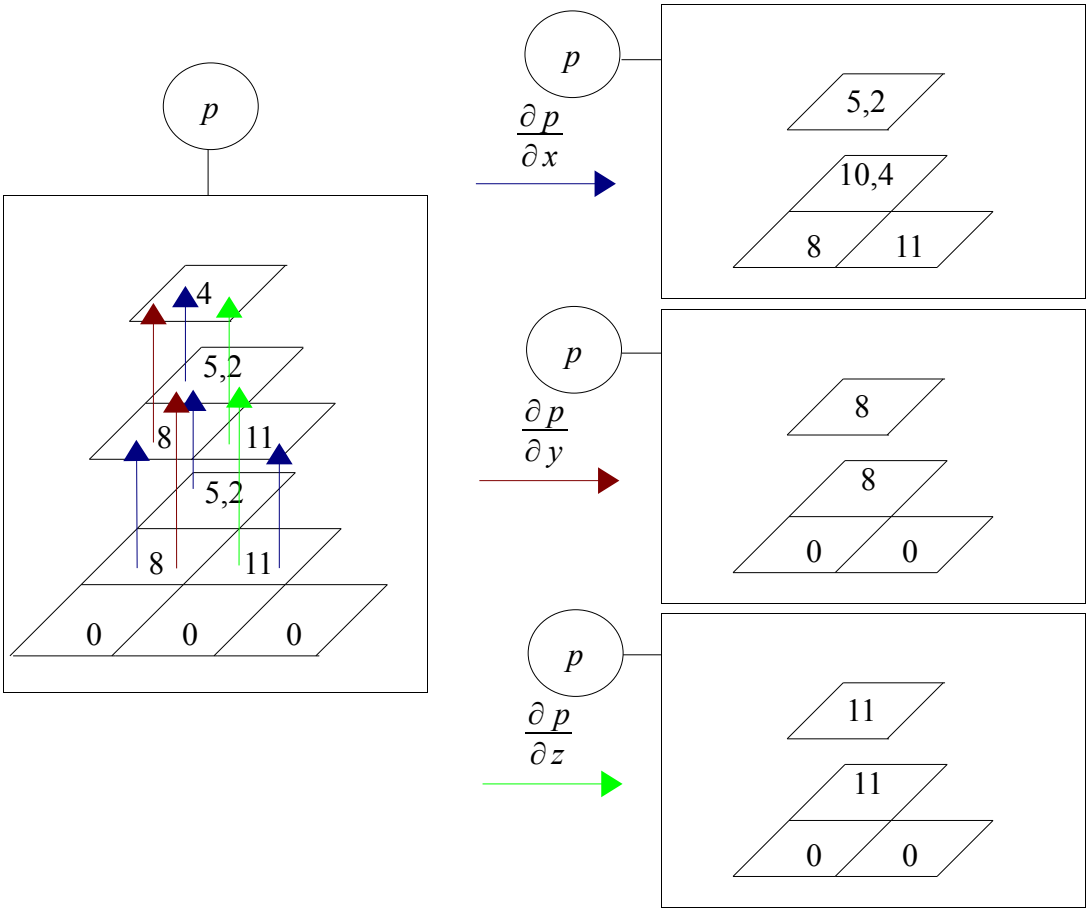
Řešení:

výsledek: $\frac{\partial p}{\partial x} = \underline{5{,}2 + 10{,}4*x + 8*y + 11*z}$

$\frac{\partial p}{\partial y} = \underline{\underline{8 + 8*x}}$

$\frac{\partial p}{\partial z} = \underline{\underline{11 + 11*x}}$

Při výpočtu parciální derivace podle výše uvedených proměnných se vždy vytvoří nový polynom řádu o 1 nižšího než byl původní polynom. Provede se výpočet a metodou **setCoef(int, int, int, double)** se nastaví koeficienty v novém polynomu. Do proměnné *name* se uloží stejná hodnota jako u polynomu dvou proměnných a klíčovým slovem **return** se vrátí odkaz na zderivovaný polynom tam, odkud byla metoda výpočtu volána.




```
public void derive( int derivace ) { int i, ir, j, jr, k, xx;
  if ( derivace == 0 || derivace == 1 || derivace == 2 ) {
    Polynom3D e = derive ( derivace, this ); name = e.getName();
    coef = new double[ e.getOrder() + 1 ][][];
     for ( i = 0; i <= getOrder(); ++i ) { coef[ i ] = new double [ i + 1 ][ ]; xx = 1;
       for ( j = 0; j <= i; ++j ) { coef [ i ][ j ] = new double [ xx ];  ++xx; } }
        for ( ir = 0; ir <= getOrder(); ++ir ) { for ( jr = 0; jr < coef[ ir ].length; ++jr ) {
          for ( k = 0; k < coef[ ir ][ jr ].length; ++k ) { i = ir – jr; j = jr - k;
            coef[ ir ][ jr ][ k ] = e.getCoef( i, j, k ); } } } } else{
            System.out.println( "Chyba zadání,tato volba neni možná!!!"); } }
```

A

B′

Text 31: Parciální derivace polynomu metodou polynomu - zdrojový kód

Z důvodu rozsáhlosti zdrojového kódu výpočtu parciální derivace pro každou proměnnou je zde uvedena pouze část, ve které se parciálně derivuje daný polynom podle proměnné *x*.

```
public static Polynom3D derive ( int derivace, Polynom3D p ) { int i, ir, j, jr, k, l; double a;
if ( derivace == 0) { if ( p.getOrder() == 0 ) {
    Polynom3D d = new Polynom3D ( p.getOrder() ); d.setCoef( 0, 0, 0, 0);
    d.setName("Zderivovaný polynom " + p.getName() + "(dx)" ); return d; }
  else { Polynom3D d = new Polynom3D ( p.getOrder() - 1 );
  for( ir = 0; ir <= p.getOrder(); ++ir ) {
    for( jr = 0; jr < p.coef[ ir ].length; ++jr ) {
      for( k = 0; k < p.coef[ ir ][ jr ].length; ++k ) { i = ir - jr ; j = jr - k;
        a = p.coef[ ir ][ jr ][ k ] * i; if ( ir == 0 ) { d.setCoef( ir, jr, k, a ); }
        else { i -= 1; if ( i >= 0) { d.setCoef( i, j, k, a ); }
          else { if ( jr <= d.getOrder() ) { d.setCoef( 0, j, k , a ); } } } } } }
d.setName("Zderivovaný polynom " + p.getName() + "(dx)" );  return d; } }
```

A′ — B

Text 32: Parciální derivace polynomu metodou třídy - zdrojový kód

3.8 Integrace polynomu

Polynomu jedné proměnné

Příklad integrace polynomu

$p1(x) = 9,2 + 4*x + 4,6*x^2 + 17,4*x^3 + 6*x^4 + 0,4*x^5 + 2,8*x^6 + 5,3*x^7$

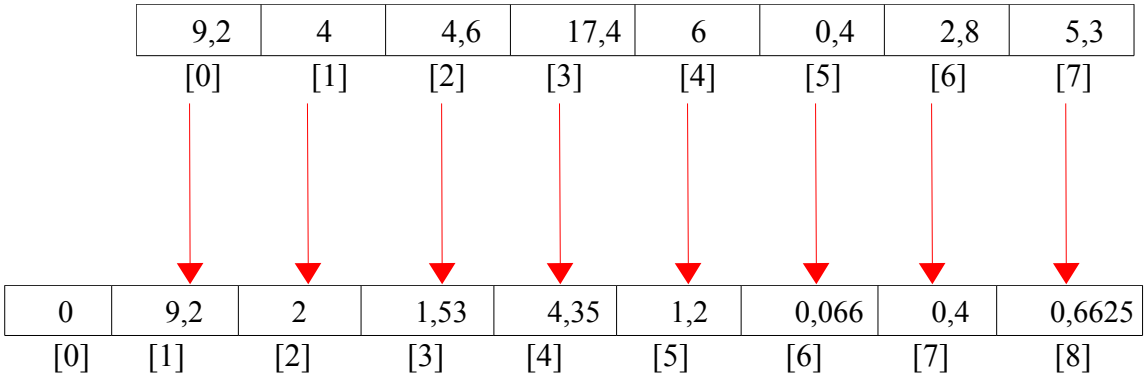
Řešení:

$$\int p1(x)dx = 0 + 9,2*x + 2*x^2 + 1,53*x^3 + 4,35*x^4 + 1,2*x^5 + 0,066*x^6 + 0,4*x^7 + 0,6625*x^8$$

Integraci polynomu lze vypočítat pomocí metody polynomu **integrate()** nebo metodou třídy **integrate(Polynom1D)**. Vždy se samotný výpočet provede v metodě třídy a pokud je integrace počítána metodou polynomu, tak se v této metodě pokaždé volá metoda třídy s klíčovým slovem **this**.

V ní se vypočítá integrace polynomu a odkaz na ni, uloženou v novém polynomu, se vrací klíčovým slovem **return** do metody polynomu. Protože integrací polynomu *n* – tého řádu je polynom (*n* + 1). řádu, vytvoří se nové jednorozměrné pole délky *n* + *2* odpovídající polynomu (*n* + 1). tého řádu.

V zde uvedeném příkladě se integruje polynom 7. řádu uložený v jednorozměrném poli délky 8. Zintegrovaný polynom bude 8. řádu a bude uložen v jednorozměrném poli délky 9. Postup výpočtu integrace polynomu je následující: koeficient na n – té pozici se vydělí indexem (n + 1). pozice a výsledek se uloží na (n + 1). pozici jednorozměrného pole, např. integrací členu $4,6*x^2$ bude $1,53*x^3$, protože číslo 4,6 uložené na pozici 2 se vydělí s indexem 3. pozice a uloží se na pozici 3 udávající řád polynomu.




```
public void integrate() {
    Polynom1D d = integrate( this ); name = d.getName();
    coef = new double[ d.getOrder() + 1 ];
    for ( int i = 0; i < coef.length; i++) { this.coef[ i ] = d.getCoef( i ); } }
public static Polynom1D integrate ( Polynom1D p ) {
    double a = 0; int posun = 0;
    Polynom1D d = new Polynom1D ( p.getOrder() + 1 );
    for ( int i = 0; i <= p.getOrder(); i++ ) { posun = i + 1;
    a = p.coef[ i ] / posun; d.setCoef( posun, a); }
    d.setName("Zintegrovaný polynom " + p.getName() ); return d; }
```

Text 33: Integrace polynomu metodou polynomu - zdrojový kód

Polynomu dvou proměnných

Příklad integrace polynomu

$p(x,y) = 4 + 2*x + 3*y + 5*x^2 + 8*x*y + 2*y^2$

Řešení:

$$\int p(x,y)\,dx = 4*x + x^2 + 3*x*y + 1{,}667*x^3 + 4*x^2*y + 2*x*y^2$$

$$\int p(x,y)\,dy = 4*y + 2*x*y + 1{,}5*y^2 + 5*x^2*y + 4*x*y^2 + 0{,}667*y^3$$

Integraci polynomu dvou proměnných lze vypočítat pomocí metody polynomu **integrate(int)** nebo metodou třídy **integrate(int, Polynom2D)**. Vždy se ale samotný výpočet provádí v metodě třídy.

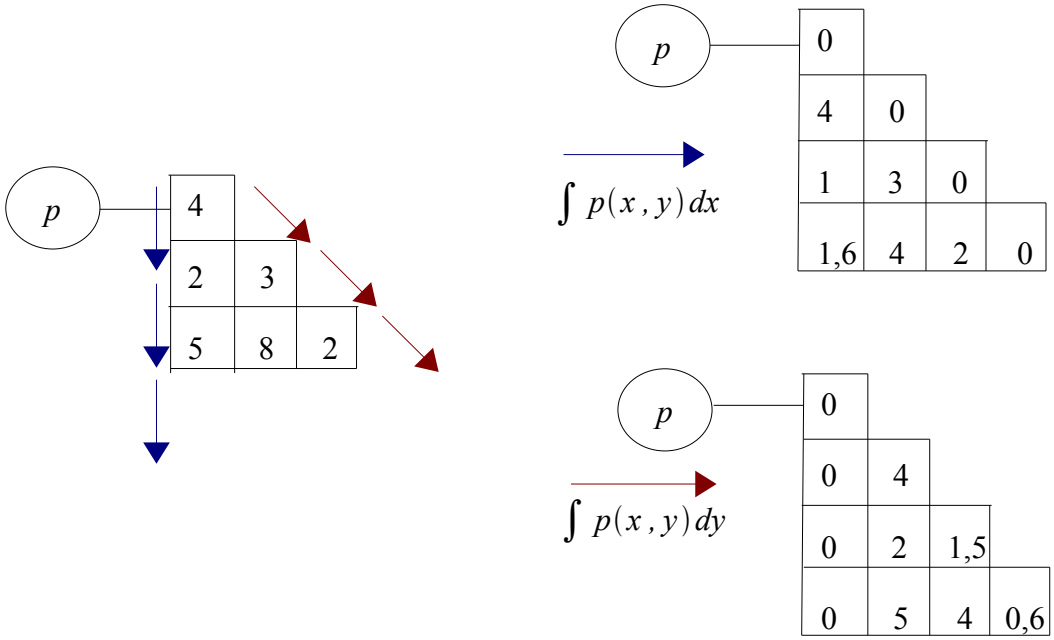
V prvním kroku se zjišťuje hodnota proměnné *integrace*, která je vstupní proměnnou metody polynomu nebo metody třídy. Její hodnota určuje, podle které proměnné se bude polynom integrovat. Má – li proměnná hodnotu 0, integruje se polynom podle proměnné *x*, má – li hodnotu 1, integruje se polynom podle proměnné *y*.

Pokud je hodnota proměnné jiná, vyvolá se výjimka a na obrazovce se objeví chybové hlášení upozorňující uživatele na chybné zadání proměnné *integrace*.

V dalším kroku se vytvoří pomocí konstruktoru nový polynom dvou proměnných řádu o 1 vyššího než byl původní polynom. Dvěma for cykly se prochází původní polynom a provádí se integrace tohoto polynomu podle proměnné *x* nebo *y* v závislosti na hodnotě vstupní proměnné *integrace*. Jednotlivé výsledky jsou ihned metodou **setCoef(int, int, double)** ukládány na nové pozice ve vytvořeném polynomu.

Klíčovým slovem **return** se vrací výsledek integrace tam odkud byla metoda volána. Byla – li volána integrace polynomu metodou polynomu, musí se po vrácení odkazu výsledku integrace do polynomu vytvořit polynom stejného řádu jakého je zintegrovaný polynom a jeho koeficienty se musí zkopírovat.

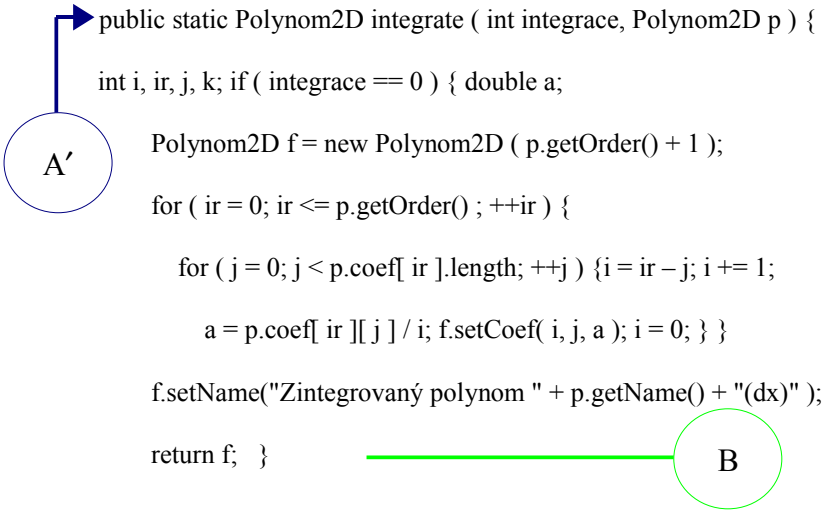
Směr červených šipek ukazuje novou pozici koeficientů po integraci podle proměnné *y*. Modré šipky ukazují směr integrace koeficientů podle proměnné *x*.




```
public void integrate( int integrace ) { int i, ir, j;
        if ( integrace == 0 || integrace == 1 ) {
A ——    Polynom2D f = integrate ( integrace, this ); name = f.getName();  <—— B′
        coef = new double[ f.getOrder() + 1 ][];
        for ( i = 0; i <= f.getOrder(); ++i ) { coef[ i ] = new double [ i + 1 ]; }
                for ( ir = 0; ir <= f.getOrder(); ++ir ) {
                for ( j = 0; j < coef[ ir ].length; ++j ) { i = ir - j;
                    coef[ ir ][ j ] = f.getCoef( i, j );} } }
    else { System.out.println( "Chyba zadání,tato volba neni možná!!!");} }
```

Text 34: Integrace polynomu dvou proměnných metodou polynomu - zdrojový kód

Z důvodu rozsáhlosti zdrojového kódu výpočtu integrace pro každou proměnnou je zde uvedena pouze část, ve které se integruje daný polynom podle proměnné *x*.



Text 35: Integrace polynomu dvou proměnných metodou třídy - zdrojový kód

Polynomu tří proměnných

Příklad integrace polynomu

$p(x, y, z) = 5 + 6*x + 2{,}8*y + 5*z$

Řešení:

$$\int p(x, y, z)dx = 5*x + 3*x^2 + 2{,}8*x*y + 5*x*z$$

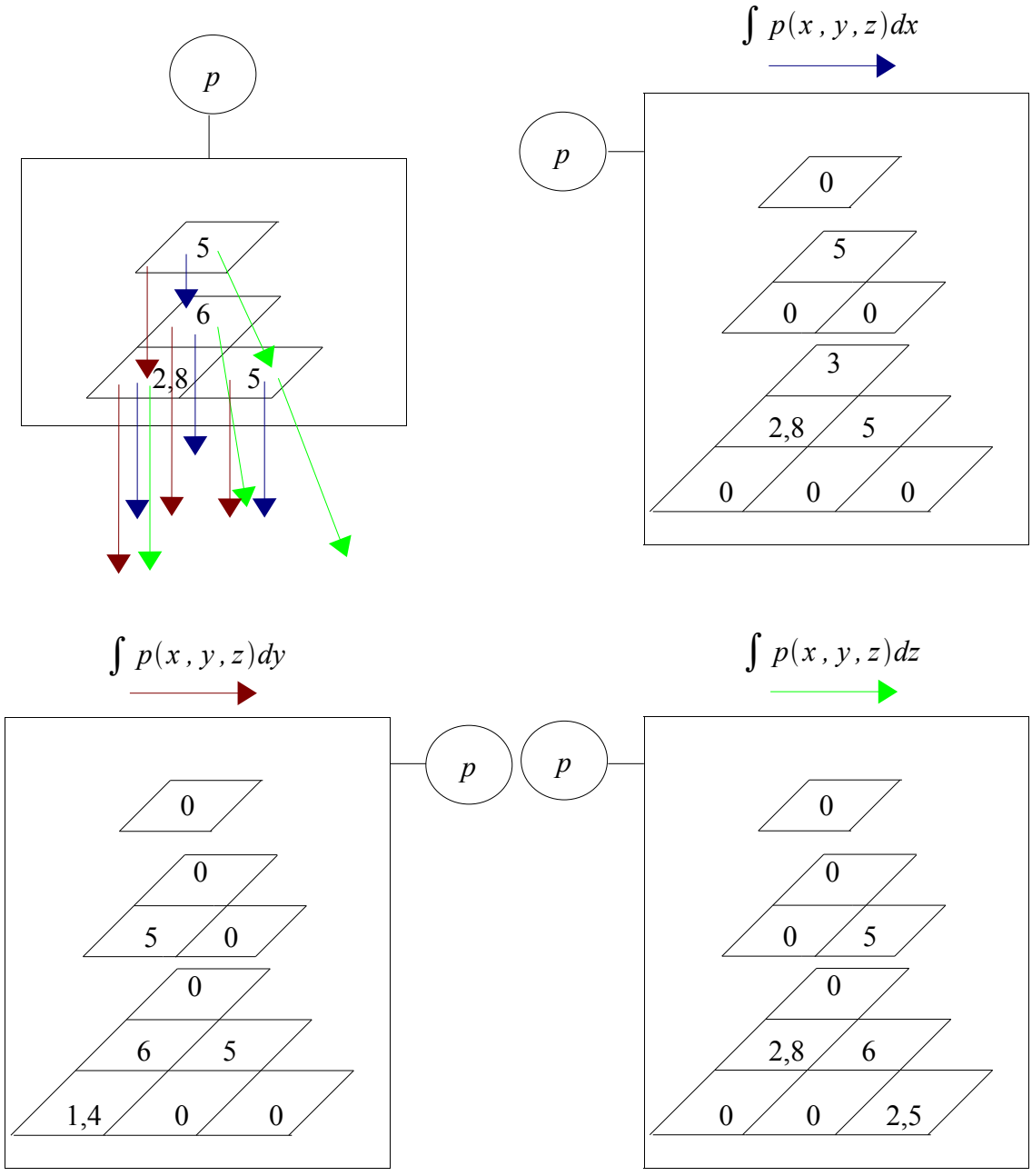
$$\int p(x, y, z)dy = 5*y + 6*x*y + 1{,}4*y^2 + 5*y*z$$

$$\int p(x, y, z)dz = 5*z + 6*x*z + 2{,}8*y*z + 2{,}5*z^2$$

Pro výpočet integrace polynomu tří proměnných jsou k dispozici 2 metody. První z nich je metoda polynomu **integrate(int)** a druhou je metoda třídy **integrate(int, Polynom3D)**. Vždy se výpočet provádí v metodě třídy. Postup výpočtu je stejný jako při výpočtu integrace polynomu dvou proměnných s tím rozdílem, že zde lze počítat i integraci polynomu podle proměnné *z*.

Při výpočtu integrace podle výše uvedených proměnných se vždy vytvoří nový polynom řádu o 1 vyššího než byl původní polynom. Provede se výpočet a metodou **setCoef(int, int, int, double)** se nastaví koeficienty v novém polynomu.

To, podle jaké proměnné se bude polynom integrovat, je závislé na hodnotě proměnné *integrace*, která je vstupním parametrem u obou typů metod. Má – li *integrace* hodnotu 0, integruje se polynom podle proměnné *x*. Má – li hodnotu 1, integruje se polynom podle proměnné *y* a s hodnotou 2 proměnné *integrace* se polynom integruje podle proměnné *z*. Je – li hodnota vstupního parametru jiná než zde uvedené, vyvolá se automaticky vyjímka a uživateli se zobrazí na obrazovku chybové hlášení.



3.9 Funkční hodnota polynomu

Polynom jedné proměnné

Příklad výpočtu funkční hodnoty polynomu Hornerovým schematem

$p(x) = x^5 + 2*x^4 + 2*x^3 + 3*x^2 + 8*x + 2$, $x = 2$

Řešení:

$(x^4 + 2*x^3 + 2*x^2 + 3*x + 8).x + 2$

$((x^3 + 2*x^2 + 2*x + 3).x + 8).x + 2$

$(((x^2 + 2*x + 2).x + 3).x + 8).x + 2$

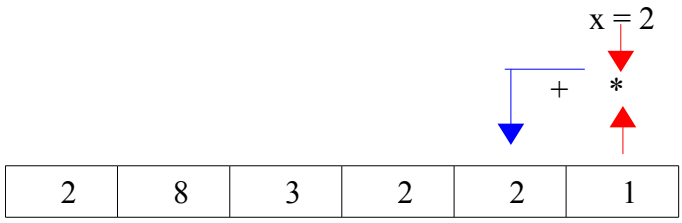
$((((x + 2).x + 2).x + 3).x + 8).x + 2$

$c_5 = 1,$ $c_4 = 1.2 + 2 = 4,$ $c_3 = 4.2 + 2 = 10,$

$c_2 = 10.2 + 3 = 23,$ $c_1 = 23.2 + 8 = 54,$ $c_0 = 54.2 + 2 = \underline{110}$

Hornerovo schema je použito i v tomto programu. Pro výpočet je vytvořena metoda polynomu s názvem **value(double)**, kde parametrem je libovolné reálné číslo. Metoda třídy nebyla vytvořena, protože výsledkem není polynom, ale pouze jeho funkční hodnota.

Po zavolání metody **value(double)** se nejvyšší koeficient vynásobí s číslem *x*, pro které je funkční hodnota polynomu počítána, a následně se sečte s druhým nejvyšším koeficientem. Tento algoritmus se aplikuje na celý polynom a výsledkem je funkční hodnota polynomu. Klíčovým slovem **return** se vrátí vypočtená hodnota např. do metody **System.out.println(p.value(2))** zajišťující tisk na obrazovku.



```
public double value( double x ) {
    double y = 0.0;
    y = coef[ coef.length -1];
    for ( int i = coef.length-2; i >=0; --i) {
       y = y*x + coef[ i ]; }
    return y; }
```

Text 36: Výpočet funkční hodnoty polynomu metodou polynomu - zdrojový kód

Polynom dvou proměnných

Funkční hodnota polynomu dvou proměnných se nepočítá pomocí **Hornerova schematu** jak tomu bylo u polynomu jedné proměnné, ale počítá se pomocí dvou jednorozměrných polí, do kterých se postupně ukládají mocniny proměnných *x* a *y*, které se následně používají v součinu s jednotlivými prvky dvourozměrného pole. Tento postup výpočtu je velice pohodlný a jednoduchý na naprogramování.

Výpočet se provádí pouze v metodě polynomu **value(double, double)**, protože výsledkem je pouze reálné číslo, nikoliv polynom, proto odpadá výpočet metodou třídy.

V metodě polynomu se nadeklaruje proměnná *soucet*, do které se postupně přičítají vypočítané funkční hodnoty polynomu a klíčovým slovem **return** se vrátí vypočtená hodnota např. do metody zajišťující tisk na obrazovku.

Příklad výpočtu funkční hodnoty polynomu

$p(x,y) = 4 + 2*x + 3*y + 5*x^2 + 8*x*y + 2*y^2, \quad x = 4,3; y = 2,7$

Řešení:

$p(4,3; 2,7) = 4 + 2*4,3 + 3*2,7 + 5*4,3^2 + 8*4,3*2,7 + 2*2,7^2$

$p(4,3; 2,7) = 220,61$

```
public double value( double x, double y ) {
    double soucet = 0; int i, j;
    double [] xp = new double[ coef.length ];
    double [] yp = new double[ coef.length ]; xp[ 0 ] = 1; yp[ 0 ] = 1;
    for( i = 0; i < coef.length; ++i ) {
      for( j = 0; j < coef[ i ].length; ++j ) {
        soucet += coef[ i ][ j ] * xp[ i - j ] * yp[ j ];  }
      if ( i == coef.length - 1 )  break;
xp[ i + 1 ] = xp[ i ] * x; yp[ i + 1 ] = yp[ i ] * y; } return soucet; }
```

Text 37: Výpočet funkční hodnoty polynomu metodou polynomu - zdrojový kód

Polynom tří proměnných

Při výpočtu funkční hodnoty polynomu tří proměnných nelze stejně jako u polynomu dvou proměnných použít **Hornerovo schema**, ale jeho určitou modifikaci.

Zde je použito tří jednorozměrných polí pro postupné ukládání mocnin proměnných *x*, *y* a *z*. Algoritmus je stejný jako u výpočtu funkční hodnoty polynomu dvou proměnných. Výpočet se provádí v metodě polynomu **value(double, double, double)**.

Příklad výpočtu funkční hodnoty polynomu

$p(x,y,z) = 9 + 12*x + 3{,}3*y + 5*z + 8*x^2 - 22*x*y + 15*x*z$,

$x = 2{,}2;\ y = 2{,}83;\ z = 2{,}21$

Řešení:

$p(2{,}2;\ 2{,}83;\ 2{,}21) = 9 + 12*2{,}2 + 3{,}3*2{,}83 + 5*2{,}21 + 8*2{,}2^2 - 22*2{,}2*2{,}83 + 15*2{,}2*2{,}21$

$p(2{,}2;\ 2{,}83;\ 2{,}21) = \underline{30{,}467}$

```
public double value ( double x, double y, double z ) { int i, ir, jr, j, k; double soucet = 0;
    double [] xp = new double[ getOrder() + 1 ]; double [] yp = new double[ getOrder() + 1 ];
    double [] zp = new double[ getOrder() + 1 ]; xp[ 0 ] = 1; yp[ 0 ] = 1; zp[ 0 ] = 1;
    for ( ir = 0; ir <= getOrder(); ++ir ) { for ( jr = 0; jr < coef[ ir ].length; ++jr ) {
        for ( k = 0; k < coef[ ir ][ jr ].length; ++k ) { i = ir – jr; j = jr - k;
          soucet += coef[ ir ][ jr ][ k ] * xp[ i ] * yp[ j ] * zp[ k ]; }} if ( ir == coef.length - 1 )
            break; xp[ ir + 1 ] = xp[ ir ] * x; yp[ ir + 1 ] = yp[ ir ] * y; zp[ ir + 1 ] = zp[ ir ] * z; }
             return soucet; }
```

Text 38: Výpočet funkční hodnoty polynomu metodou polynomu - zdrojový kód

4 Závěr

Tato práce se zabývala implementací základních tříd popisujících polynomy a matematické operace, které lze s polynomy provádět. Autor bakalářské práce svým řešením vytvořil ucelenou aplikaci obsahující implementaci polynomů jedné, dvou a tří proměnných. Všechny body zadání byly splněny v plném rozsahu.

Práce byla rozdělena na 3 samostatné kapitoly.

V první kapitole byl proveden teoretický rozbor. Bylo definováno, co jsou to polynomy a byly podrobně popsány základní matematické operace týkající se polynomů.

Druhá kapitola se zabývala problematikou objektově orientovaného programování (OOP) jako základního programátorského přístupu k psaní programů. Byly zde popsány základní pojmy OOP třída, metoda, datové členy, dědičnost, polymorfismus a další.

Ve třetí, nejrozsáhlejší kapitole byla popsána vlastní implementace. Podrobně je popsán způsob uložení polynomů do paměti. Byl zvolen speciální způsob uložení, který zajistí nejrychlejší přístup k potřebným datům a tím i nejvyšši rychlost vykonávaného kódu. Implementace algoritmů potřebných pro provádění základních matematických operací je pak podřízena tomuto způsobu uložení. Vzhledem k unikátnosti použitého řešení byl pro názornost popis jednotlivých operací doplněn názornými grafickými ukázkami.

Práce vycházela z požadavků definovaných metodikou DF²EM [3], která definuje základní třídy a rozhraní pro implementaci metody konečných prvků. Tato práce představuje pouze malou dílčí část, kterou se metodika zabývá, ale podrobnost s jakou tato práce problém vyřešila požadavky metodiky přesahuje. Vznikl tak ucelený a dokumentovaný nástroj pro práci s polynomy.

5 Použitá literatura

[1] BARTSCH, Hans-Jochen. *Matematické vzorce*. Přeložil Z. Tichý. 1. evidované vyd. Praha : Státní nakladatelství technické literatury, 1983. 830 s.

[2] ECKEL, Bruce. *Myslíme v jazyce JAVA : knihovna programátora*. Praha : Grada Publishing, 2001. 432 s. ISBN 80-247-9010-6.

[3] FRYDRYCH, Dalibor. *Metodika implementace metody konečných prvků DF²EM* [online]. 2007 , 2. května 2007 [cit. 2007-05-02]. Kódováno ve windows-1250. Dostupný z WWW: <http://acervus.kmo.tul.cz/DF2EM>.

[4] PITNER, Tomáš. *Programování v jazyce Java* [online]. c2001-2003 , 14. prosince 2003 [cit. 2007-04-24]. Kódováno ve windows-1250. Text v češtině. Dostupný z WWW: <http://www.fi.muni.cz/~tomp/slides/pb162/printable.html>.

[5] REKTORYS, Karel. *Variační metody v inženýrských problémech a v problémech matematické fyziky*. 6. upr. vyd. Praha : Academia Praha, 1999. 104 s. ISBN 80-200-0714-8.

[6] SEMECKÝ, Jiří. *Naučte se Javu* [online]. 26. 4. 2002 [cit. 2007-04-24]. Kódováno ve windows-1250. Text v češtině. Dostupný z WWW: <http://interval.cz/clanky/naucte-se-javu-uvod/>. ISSN 1212-865.

[7] VIRIUS, Karel. *Java pro zelenáče*. 2. upr. vyd. Praha : Neocortex spol. s r. o., 2005. 267 s. ISBN 80-86330-17-6.

[8] ZIENKIEWICZ, O.C., TAYLOR, R.L. *Finite Element Method (5th Edition) Volume1 - The Basis*. [s.l.] : Butterworth Heinemann, srpen 2000. 690 s. ISBN 0-7506-5049-4.