Zadanie domowe nr 1
Wycin(an)ki macierzowe

Programowanie Obiektowe, Informatyka
Wersja 1.00

Ogólny opis

Należy stworzyć klasy reprezentujące tablice liczb rzeczywistych (typu double). Interesują nas tu tablice wymiaru 0 (zwane dalej *skalarami*), wymiaru 1 (zwane dalej *wektorami*) i wymiaru 2 (zwane dalej *macierzami*). Wektory są poziome lub pionowe. Macierze są prostokątne (tj. nie muszą być kwadratowe). Przyjmujemy, że wektory i macierze nie mogą być puste.

Obiekty tych klas powinny udostępniać typowe operacje arytmetyczne:

- dodawanie (metody *suma* i *dodaj*),
- mnożenie (metody *iloczyn* i *przemnóż*),
- negowanie (metody *negacja* i *zaneguj*).

W przypadku dodawania i mnożenia chcemy mieć dwie operacje, jedna dająca nowy wynik, druga modyfikująca pierwszy argument (podobnie jak w Javie mamy operacje + i +=[1]). Przy czym operacje modyfikujące są możliwe tylko wtedy, gdy pierwszy argument ma taki sam lub większy wymiar niż drugi argument i taki sam kształt jak wynik. Dla negacji też chcemy mieć dwie operacje (jedną dającą tablicę tego samego wymiaru i drugą modyfikującą).

Operacje dwuargumentowe można także wykonywać na tablicach różnych wymiarów. W takiej sytuacji:

- gdy jeden argument jest skalarem, operację zawsze da się wykonać, a wykonanie polega na wykonaniu tej operacji na każdym elemencie tablicy i skalarze (np. na dodaniu skalara do każdego elementu tablicy).
- gdy jeden argument jest macierzą, a drugi[2] wektorem, to w zależności od kolejności argumentów i tego, czy wektor jest poziomy, czy pionowy z długością wektora musi się zgadzać liczba kolumn/wierszy macierzy. Wskazaną operację arytmetyczną przeprowadzamy dla wektora i po kolei dla każdego wiersza/każdej kolumny macierzy, wykonując wskazane działanie po kolei dla każdej pary elementów wektora i wiersza/kolumny macierzy.

Operacja dodawania dwu wektorów jest możliwa, o ile oba mają tę samą orientację (czyli oba są poziome lub oba są pionowe) i są tego samego rozmiaru. Efektem jest wektor orientacji i rozmiaru takiego jak dodawane, elementy zaś są sumami odpowiednich elementów dodawanych wektorów.

Mnożenie dwu wektorów wygląda podobnie jak dodawanie, ale mnożenie wektorów tej samej orientacji rozumiemy tu jako iloczyn skalarny (i wynikiem jest skalar), wektorów innej orientacji zaś jako iloczyn wektorowy (i wynikiem jest macierz[3]).

Przypisania

Do tablic można przypisywać (metoda *przypisz*). Argument musi mieć ten sam lub mniejszy wymiar. W tym drugim przypadku przypisanie skalara oznacza przypisanie go do każdego elementu, przypisanie zaś wektora (do macierzy) oznacza skopiowanie jego elementów do elementów każdego wiersza/kolumny (zależnie od orientacji wektora). Aby przypisania się powiodło, tablice muszą mieć zgodną liczbę elementów w odpowiednich wymiarach:

[1] Na wszelki wypadek przypominamy, że w Javie nie można przeciążać operatorów, więc te operacje należy zaimplementować jako metody.

[2] 'Drugi' w sensie 'pozostały', może to być np. pierwszy argument operacji.

[3] Ta macierz może być bardzo mała, np. z jedną tylko kolumną lub jednym wierszem, gdy któryś z mnożonych wektorów miał długość jeden. Ale nawet gdy oba wektory miały długość jeden wynikiem jest macierz (rozmiaru 1 * 1), a nie skalar (czy wektor).

- skalar można przypisać zawsze,
- w przypadku tej samej liczby wymiarów kształty muszą być takie same (w przypadku wektorów dozwalamy na zmianę orientacji),
- dla przypisania wektora do macierzy rozmiar jej wierszy/kolumn (zależnie od orientacji wektora) musi być równy długości wektora.

Indeksowanie

Na wszystkich tablicach możliwe jest indeksowanie. Należy zaimplementować zarówno odczytywanie (metoda *daj*), jak i ustawianie (metoda *ustaw*) wartości wskazanego elementu (typu double). Macierz wymaga podania dwu indeksów, wektor jednego, skalar żadnego. Należy również zaimplementować ogólne indeksowanie, korzystając ze składni metod ze zmienną liczbą parametrów, np.:

```
public void mm(int... tab)  {
   for (int i=0; i<tab.length; i++) {
       System.out.print(tab[i] + " ");
   }
}
```

Taką metodę można wywołać: bez argumentów, z jednym, dwoma itd. Parametr (powyżej *tab*) musi być ostatnim. Jego typem jest w tym przykładzie int[], zatem składnia z kropkami polega po prostu na tym, że zmienna liczba argumentów jest automatycznie pakowana przez kompilator w tablicę.

Wycinki

Operacja *wycinek* zastosowana do tablicy daje jej wycinek tego samego wymiaru, ale nie większego kształtu. Wycinek też jest tablicą, więc można wykonywać na nim takie same operacje jak na tablicy. Jeśli są to operacje modyfikujące (np. *przypisz*), to modyfikują tablicę, z której jest to wycinek. Wycinek ma wymiar taki jak tablica, z której powstał, ale (zapewne) inny kształt i liczbę elementów (nie większe niż w tablicy). Tworząc wycinek, trzeba podać zero (skalar), dwie (wektor) lub cztery (macierz) liczby. Oznaczają one, odkąd dokąd wybieramy elementy oryginału. Np. *macierz.wycinek(2,4,1,5)* oznacza, że wycinek dotyczy wierszy 2..4 oraz kolumn 1..5 podanej macierzy. Otrzymany wycinek będzie miał wiersze 0..2 oraz kolumny 0..4 (takich zakresów indeksów należy używać w operacjach *daj* i *ustaw* tego wycinka, a nie 2..4 i 1..5). Należy (oczywiście) sprawdzić poprawność argumentów. Można robić wycinki wycinków.

Inne operacje

Poza wymienionymi chcemy mieć operacje:

- int *wymiar*() - liczba wymiarów tablicy (0, 1 lub 2),
- int *liczba_elementów*() - liczba elementów (typu double) tablicy,
- int[] *kształt* - tablica zawierająca wymiar() liczb, podających liczbę elementów w danym wymiarze. Dla wektora (niezależnie od tego, czy jest pionowy, czy poziomy) jedna liczba równa *liczba_elementów*, dla macierzy najpierw liczba wierszy, potem liczba kolumn.
- *toString*() - dającą tekstową reprezentację tablicy, z podziałem na wiersze (macierze) i informacją o kształcie, a dla wektorów o ich orientacji.

- *kopia*() - daje (głęboką[4]) kopię argumentu,

- *transponuj*() - dokonuje transpozycji odbiorcy. Dla skalara nie robi nic, dla wektora zmienia jego orientację, dla macierzy (być może) zmienia jej kształt.

Wyjątki

W tym zadaniu chcemy nacieszyć się wyjątkami i ich nadzorowaniem przez kompilator. Zatem należy definiować i zgłaszać własne nadzorowane wyjątki (nie tylko zgłaszać Exception z odpowiednim komunikatem).
Przy podejmowaniu decyzji, czy wyjątek powinien być obsłużony w metodzie, czy przesłany wyżej, prosimy kierować się następującą regułą:

- jeśli wyjątek wynika z błędu logicznego w metodzie, to nie przekazujemy go dalej, metoda ma go przechwycić, wypisać czytelny komunikat podający kontekst błędu (wartości istotnych zmiennych/parametrów/atrybutów, jeśli dotyczą błędu) i zakończyć działanie programu (to może być wołanie System.exit(kodBłędu), albo asercja).

- jeśli wyjątek wynika z niepoprawnej wartości argumentu metody, to ten wyjątek ma być przekazany na zewnątrz metody.

Podmienianie i przeciążanie metod

Obie te techniki należy wykorzystać w tym zadaniu. Na wszelki wypadek dla przypomnienia (omawialiśmy obie techniki na osobnym wykładzie): podmienianie to kluczowy element programowania obiektowego, pozwala realizować polimorfizm, odbywa się w czasie wykonywania programu, przeciążanie zaś występuje w różnych paradygmatach programowania, nie ma związku z obiektowością i jest realizowane w czasie kompilacji. W sytuacji, gdy działanie zależy od typu argumentu (np. przypisanie), zastosujmy przeciążanie (metody przypisz(Skalar), przypisz(Wektor), przypisz(Macierz)) i podmienianie (np. metoda przypisz(Skalar) będzie miała odmienną implementację w klasie Skalar niż w klasie Wektor).

Zadanie

Należy zaprojektować i zaimplementować klasy realizujące opisane w tym zadaniu tablice oraz napisać kod wywołujący poszczególne operacje (p. Testy). Następnie należy wgrać do Moodle'a kod źródłowy (pliki .java) wraz z podziałem na pakiety (katalogi). Czyli należy spakować (do formatu zip) zawartość katalogu src. Prosimy nie wgrywać innych plików.
W treści zadania podano (część) nazw klas i metod, prosimy zachować te nazwy w swoim rozwiązaniu.

Dodatkowe uwagi

- Prosimy pamiętać, że to zadanie z programowania obiektowego, zdecydowanie nie należy się obawiać korzystania z klas, obiektów, konstruktorów, zakresów widoczności itp. Wykazanie się umiejętnością unikania konstrukcji obiektowych nie jest celem tego zadania. Np. idealnie działający program z jedną tylko klasą będzie odczytany jako nieograniczona sympatia dla okrągłych liczb parzystych mających nieskończenie wiele dzielników.

- To większe zadanie, prosimy o rozwiązywanie go w sposób przyrostowy. Np. zaimplementowanie operacji na jednym rodzaju tablic wydaje się być dobrym etapem.

- Kod większych programów wymaga refaktoryzacji. Często dopiero po napisaniu działającego programu zauważa się możliwe usprawnienia kodu, poprawiające czytelność i jakość kodu (typowy przykład - po

[4]Głęboka kopia nie dzieli żadnych elementów z oryginałem. Np. płytka kopia tablicy tablic liczb typu int jest nową tablicą zawierającą referencje do tablic składowych oryginału, głęboka zaś jest nową tablicą nowych tablic.

napisaniu rozwiązania zauważa się, że dwie metody mają podobny fragment kodu, który można, a nawet należy, wydzielić w postaci osobnej metody). Ta faza tworzenia oprogramowania wymaga czasu. Prosimy więc nie zostawiać pisania programów na ostatnią chwilę.

- Zadanie zostało tak zaprojektowane, żeby dało się je łatwo zrobić, korzystając z tablic (tych Java'owych). Ponieważ nie znamy jeszcze kontenerów ze standardowej biblioteki Javy, nie możemy ich w tym zadaniu wymagać. A ponieważ wszyscy powinni mieć takie samo zadanie do zrobienia, to nie wolno w tym zadaniu korzystać z kontenerów innych niż tablice.

- Przyjmujemy indeksowanie wszystkiego (wiersze, kolumny) od zera.

- Należy sprawdzać poprawność wykonywanych operacji (p. Wyjątki).

Życzymy powodzenia!

Testy

W ramach programu sprawdzającego działanie stworzonego systemu obiektowego, należy zaprogramować testy obejmujące wszystkie następujące przykłady.

Informacje o obiekcie

- wielkość skalarna [1.0] ma

  wymiar równy 0, kształt równy [], liczbę elementów równą 1;

- wektory $\begin{bmatrix} 1.0 & 2.0 & 1.0 \end{bmatrix}$ i $\begin{bmatrix} 2.0 \\ 2.0 \\ 3.0 \end{bmatrix}$ oba mają

  wymiar równy 1, kształt równy [3], liczbę elementów równą 3;

- macierz $\begin{bmatrix} 1.0 & 0.0 & 2.0 \\ 2.0 & 1.0 & 3.0 \\ 1.0 & 1.0 & 1.0 \\ 2.0 & 3.0 & 1.0 \end{bmatrix}$ ma

  wymiar równy 2, kształt równy $\begin{bmatrix} 4 & 3 \end{bmatrix}$, liczbę elementów równą 12.

Operacje arytmetyczne

- dodawanie / mnożenie skalar - skalar:

[3.5] + [11.5] powinno dać wielkość skalarną [15.0],

[3.0] * [12.0] powinno dać wielkość skalarną [36.0];

- dodawanie / mnożenie skalar - wektor:

[3.0] + $\begin{bmatrix} 1.0 & 2.5 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 4.0 & 5.5 \end{bmatrix}$,

[4.0] * $\begin{bmatrix} 1.5 & 2.25 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 6.0 & 9.0 \end{bmatrix}$,

[3.0] + $\begin{bmatrix} 1.0 \\ 2.5 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 4.0 \\ 5.5 \end{bmatrix}$,

[4.0] * $\begin{bmatrix} 1.5 \\ 2.25 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 6.0 \\ 9.0 \end{bmatrix}$;

- dodawanie / mnożenie wektor - skalar:

  wyniki j.w. ale dla odwrotnej kolejności operandów;

- dodawanie / mnożenie skalar - macierz:

$[3.0] + \begin{bmatrix} 1.25 & 3.0 & -12.0 \\ -51.0 & 8.0 & 3.5 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 4.25 & 6.0 & -9.0 \\ -48.0 & 11.0 & 6.5 \end{bmatrix}$,

$[-3.0] * \begin{bmatrix} 1.25 & 3.0 & -12.0 \\ -51.0 & 8.0 & 3.5 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} -3.75 & -9.0 & 36.0 \\ 153.0 & -24.0 & -10.5 \end{bmatrix}$;

- dodawanie / mnożenie macierz - skalar:

  wyniki j.w. ale dla odwrotnej kolejności operandów;

- dodawanie / mnożenie wektor - wektor:

$\begin{bmatrix} 1.0 & 2.0 & 3.0 \end{bmatrix} + \begin{bmatrix} 1.0 & 1.0 & -2.0 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 2.0 & 3.0 & 1.0 \end{bmatrix}$,

$\begin{bmatrix} -2.0 \\ 5.0 \end{bmatrix} + \begin{bmatrix} -5.0 \\ 2.0 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} -7.0 \\ 7.0 \end{bmatrix}$,

$\begin{bmatrix} 3.0 & 2.0 & -1.0 \end{bmatrix} * \begin{bmatrix} -2.0 & 2.0 & 1.0 \end{bmatrix}$ powinno dać wielkość skalarną $\begin{bmatrix} -3.0 \end{bmatrix}$,

$\begin{bmatrix} -2.0 \\ -5.0 \\ 1.0 \\ 3.0 \end{bmatrix} * \begin{bmatrix} -5.0 \\ 1.0 \\ 2.0 \\ -3.0 \end{bmatrix}$ powinno dać wielkość skalarną $\begin{bmatrix} -2.0 \end{bmatrix}$,

$\begin{bmatrix} 1.0 & 2.0 & 3.0 \end{bmatrix} * \begin{bmatrix} 1.0 \\ 1.0 \\ -2.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} -3.0 \end{bmatrix}$,

$\begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \end{bmatrix} * \begin{bmatrix} 1.0 & 1.0 & -2.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 1.0 & 1.0 & -2.0 \\ 2.0 & 2.0 & -4.0 \\ 3.0 & 3.0 & -6.0 \end{bmatrix}$;

- dodawanie wektor - macierz:

$\begin{bmatrix} 3.0 & 1.5 & -2.0 \end{bmatrix} + \begin{bmatrix} 1.0 & 3.5 & -12.0 \\ -5.0 & 8.0 & 3.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 4.0 & 5.0 & -14.0 \\ -2.0 & 9.5 & 1.0 \end{bmatrix}$,

$\begin{bmatrix} 7.5 \\ -5.0 \end{bmatrix} + \begin{bmatrix} 1.0 & 3.5 & -12.0 \\ -5.0 & 8.0 & 3.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 8.5 & 11.0 & -4.5 \\ -10.0 & 3.0 & -2.0 \end{bmatrix}$,

- dodawanie macierz - wektor:

  wyniki j.w. ale dla odwrotnej kolejności operandów;

- mnożenie macierz - wektor i wektor - macierz:

$\begin{bmatrix} 1.0 & 2.0 \\ 3.0 & -2.0 \\ 2.0 & 1.0 \end{bmatrix} * \begin{bmatrix} -1.0 \\ 3.0 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 5.0 \\ -9.0 \\ 1.0 \end{bmatrix}$,

$\begin{bmatrix} 1.0 & -1.0 & 2.0 \end{bmatrix} * \begin{bmatrix} 1.0 & 2.0 \\ 3.0 & -2.0 \\ 2.0 & 1.0 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 2.0 & 6.0 \end{bmatrix}$;

- dodawanie / mnożenie macierz - macierz:

$\begin{bmatrix} 1.0 & -2.0 & 3.0 \\ 2.0 & 1.0 & -1.0 \end{bmatrix} + \begin{bmatrix} 3.0 & -1.0 & 2.0 \\ 1.0 & 1.0 & -2.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 4.0 & -3.0 & 5.0 \\ 3.0 & 2.0 & -3.0 \end{bmatrix}$,

$\begin{bmatrix} 2.0 & 0.5 \\ 1.0 & -2.0 \\ -1.0 & 3.0 \end{bmatrix} * \begin{bmatrix} 2.0 & -1.0 & 5.0 \\ -3.0 & 2.0 & -1.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 2.5 & -1.0 & 9.5 \\ 8.0 & -5.0 & 7.0 \\ -11.0 & 7.0 & -8.0 \end{bmatrix}$;

- negowanie:

  wielkości skalarnej [17.0] powinno dać [−17.0],

  wektora [10.0 −45.0 0.0 29.0 −3.0] powinno dać [−10.0 45.0 0.0 −29.0 3.0],

  macierzy $\begin{bmatrix} 0.0 & 0.5 & -1.25 \\ 11.0 & -71.0 & -33.5 \\ -2.0 & -1.75 & -99.0 \end{bmatrix}$ powinno dać $\begin{bmatrix} 0.0 & -0.5 & 1.25 \\ -11.0 & 71.0 & 33.5 \\ 2.0 & 1.75 & 99.0 \end{bmatrix}$.

Przypisania

- przypisanie wielkości skalarnej [0.5] do:
  - wielkości skalarnej [1.0] powinno dać skalar [0.5],
  - wektora [1.0 2.0 3.0] powinno dać wektor [0.5 0.5 0.5],
  - macierzy $\begin{bmatrix} 1.0 & 2.0 \\ -3.0 & -4.0 \\ 5.0 & -6.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$;
- przypisanie wektora $\begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \end{bmatrix}$ do:
  - wektora $\begin{bmatrix} -1.0 \\ 0.0 \\ 1.0 \end{bmatrix}$ powinno dać wektor $\begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \end{bmatrix}$,
  - wektora [−1.0 0.0 1.0] powinno dać wektor $\begin{bmatrix} 1.5 \\ 2.5 \\ 3.5 \end{bmatrix}$,
  - macierzy $\begin{bmatrix} 1.0 & 2.0 & -1.0 & -2.0 \\ -3.0 & -4.0 & 3.0 & 4.0 \\ 5.0 & -6.0 & -5.0 & 6.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 1.5 & 1.5 & 1.5 & 1.5 \\ 2.5 & 2.5 & 2.5 & 2.5 \\ 3.5 & 3.5 & 3.5 & 3.5 \end{bmatrix}$;
- przypisanie macierzy $\begin{bmatrix} 10.5 & 20.5 & 30.5 \\ -1.5 & 0.0 & 1.5 \end{bmatrix}$:
  - do macierzy $\begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 3.0 & 2.0 & 1.0 \end{bmatrix}$ powinno dać macierz $\begin{bmatrix} 10.5 & 20.5 & 30.5 \\ -1.5 & 0.0 & 1.5 \end{bmatrix}$.

Wycinki

- wycinek wielkości skalarnej [13.125] powinien być skalarem [13.125] ;
- wycinek wektora [1.0 21.0 32.0 43.0 54.0] w zakresie od 2 do 3 powinien być wektorem [32.0 43.0] ;
- wycinek macierzy $\begin{bmatrix} 7.0 & -21.0 & 15.0 & -31.0 & 25.0 \\ -21.0 & 15.0 & -31.0 & 25.0 & 7.0 \\ 15.0 & -31.0 & 25.0 & -7.0 & -21.0 \\ -31.0 & 25.0 & 7.0 & -21.0 & 15.0 \end{bmatrix}$ w zakresach od 1 do 3 (wiersze) i od 1 do 2 (kolumny) powienien być macierzą $\begin{bmatrix} 15.0 & -31.0 \\ -31.0 & 25.0 \\ 25.0 & 7.0 \end{bmatrix}$.

Historia zmian:

- 1.0: 5 * V = 2025, pierwsza wersja.
- 1.01: 11 V 2025, w przykładowym kodzie w podrozdziale o indeksowaniu początkowa wartość zmiennej ‘i’ została zmieniona z 1 na 0.
- 1.02: 13 V 2025, w rozdziale “Ogólny opis” w zdaniu “Przy czym operacje modyfikujące są możliwe tylko wtedy, gdy pierwszy argument ma taki sam lub większy wymiar niż drugi argument i taki sam jak wynik.” dodano słowo “kształt” po “taki sam”.