

EDA016 Programmeringsteknik för D

Läsvecka 9: Matriser

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

9 Matriser

- Att göra denna vecka
- Repetition: Vektorer
- Skapa matriser
- Använda matriser

Att göra i Vecka 9: Förstå matriser. Komma ikapp med övningar etc.

- 1 Läs följande kapitel i kursboken: 8.6-8.7
Begrepp: matris, vektor av vektorer, matrisindexering, rader, kolonner/kolumner, nästlade for-satser
- 2 Gör övning 8: matriser, `StringBuilder`
- 3 Förslag: skriv kontrollskrivningsuppgiften själv i Eclipse.
Övning: implementera en klass `ChordView` som har ett `Chord` och ett `SimpleWindow` som attribut och metoden `draw()` som ritar ackordbilder. Skapa privata hjälpmetoder för olika delar av bilden (rutnät, band noll, kvadrat för nedtryckt sträng, etc.)
- 4 Träffas i samarbetsgrupper och hjälp varandra
- 5 Gör Lab 7: Maze

Repetition: Vektorer

- Deklarera en vektor-referensvariabel:

```
int[] v; //ej intialliserad
```

- Vektorer är (speciella) objekt; vi kan använda **null** och **new**:

```
v = null; //inga element är allokerade förrän vi gör new  
v = new int[5]; //allokera plats för 5 heltal, init 0
```

- Deklarera, allokera och initialisera på en och samma gång:

```
int[] v = new int[5]; //elementen initialiseras med nollor
```

- Indexera i en vektor:

```
int firstElement = v[0]; //indexering från noll  
int lastElement = v[4];  
int error = v[5]; //ger ArrayIndexOutOfBoundsException
```

- Elementen kan användas som "vanliga" variabler:

```
v[3] = v[3] + 1; //öka fjärde elementet med 1
```

- Deklarera referens, allokera och initialisera vektor från konstanter:

```
int[] xs = new int[] {42, 43, 44, 45};  
int[] ys = {42, 43, 44, 45}; //kortform av ovan
```

Övning: Vektorer

Rita bild av minnet efter rad 1 och rad 4.

```
1  int[] v = new int[5];
2  for (int i = 0; i < 5; i++){
3      v[i] = 2 * i + 1;
4  }
5  System.out.println(v[0]);
6  System.out.println(v[v[0]]);
7  System.out.println(v[v[v[0]]]);
8  System.out.println(v[v[v[v[0]]]]);
```

Vad skrivs ut på raderna 5-8?

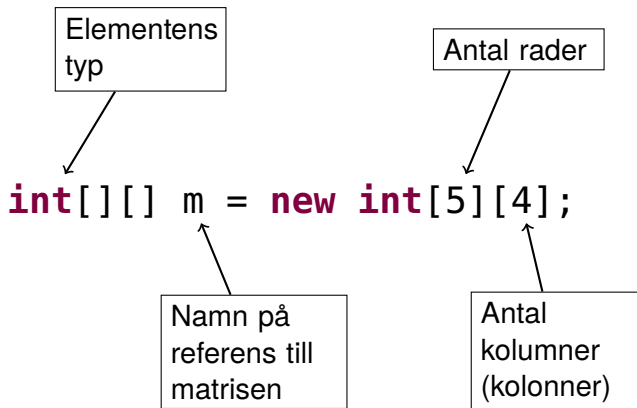
Vad är en matris?

- En matris är ett tvådimensionellt fält; en tabell; ett "rutnät".
- Exempel 5x4-matris:

kolumn	0	1	2	3
rad 0	7	9	123	41
rad 1	22	-18	12	2
rad 2	11	16	-4	0
rad 3	1	1	1	1
rad 4	2	2	2	2

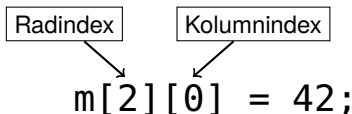
Deklarera och allokerar en matris i Java med vektorer

I Java kan matriser representeras som vektorer av vektorer:



Använda matriser (vektorer av vektorer)

- Indexera i en matris:



Radindex Kolumnindex

`m[2][0] = 42;`

- Ta reda på antalet rader:
`m.length`
- Ta reda på antalet kolumner på tredje raden:
`m[2].length`
- Man *kan* ha varierande radlängd, genom att allokeras resp radvektor enskilt, men det är ganska ovanligt:
`m[1] = new int[] { 42, 43 };`
- Man *kan* ha vektorer av vektorer av vektorer i godtyckligt djup, t.ex. för att representera en kub (3 dimensioner), men det är inte så vanligt.

Skriva ut alla element

lecture-examples/src/week09/matrix:

```
1      int m[][] = new int[5][4];
2      m[1] = new int[] { 42, 43 };
3
4      for (int row = 0; row < m.length; row++) {
5          System.out.print("|");
6          for (int col = 0; col < m[row].length; col++) {
7              System.out.print(m[row][col] + "|");
8          }
9          System.out.println();
10     }
```

```
|0|0|0|0|
|42|43|
|0|0|0|0|
|0|0|0|0|
|0|0|0|0|
```

Beräkna radsummorna

lecture-examples/src/week09/matrix:

```
1      int matrix[][] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, {9, 10, 11, 12} };
2
3      for (int row = 0; row < matrix.length; row++) {
4          int sum = 0;
5          for (int col = 0; col < matrix[row].length; col++) {
6              sum = sum + matrix[row][col];
7          }
8          System.out.println(sum);
9      }
```

10

26

42

Användning av nästlade for-satser och matriser

Pseudokod:

```
för varje rad  
  för varje kolumn  
    bearbeta m[rad][kolumn]
```

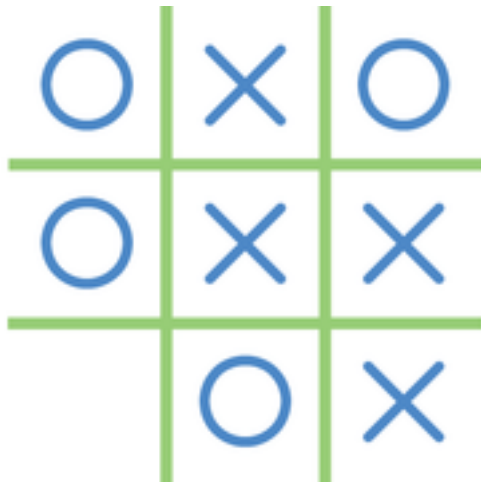
Implementation i Java:

```
// deklarera och initisalisera m  
for (int row = 0; row < matrix.length; row++) {  
  // deklarera och initisalisera  
  for (int col = 0; col < matrix[row].length; col++) {  
    // bearbeta varje element m[rad][kolumn]  
  }  
  // bearbeta varje rad vid behov  
}
```

Övning: matris för schakbräde

Deklarera och skapa en matris som representerar ett schackbräde med 8x8 rutor. På rutorna ska man kunna placera schackpjäser som beskrivs av klassen ChessPiece.

Designexempel: Tre-i-rad



Tic Tac Toe

Tre-i-rad med MVC-designmönstret

Vi ska använda ett designmönstret som kallas *Model-View-Controller* där vi har dessa tre klasser med sina respektive speciella ansvar:

- **Model:** Har hand om spelets datamodell, här spelbrädet och pjäsernas placering. Har metoder som kan avläsa och ändra data om spelet.
- **View:** Har hand om den för användaren synliga representationen av modellen och användarens interaktion med modellen. Visar brädet och läser användarens input.
- **Controller:** Styr spelet enligt dess regler och har hand om logiken för en spelomgång.

Idén är att man renodlar ansvaret så att man lätt kan ändra och byta delar utan att det påverkar de andra delarna.

Implementera Model enligt specifikationen

```
/** Skapar en modell av ett Tre-i-rad-spel med ett tomt bräde */
Model();

/** Tömmer brädet från pjäser */
void clear();

/** Returnerar pjäsen på platsen (row, col): X eller 0 eller blank om ledig */
char getMark(int row, int col);

/** Returnerar true om platsen är ledig */
boolean isFree(int row, int col);

/** Returnerar true om platsen är innanför brädets rader och kolumner */
boolean isInside(int row, int col);

/** Returnerar true om alla platserna är upptagna */
boolean isFull();

/** Placerar en ring på platsen (row, col) */
void markNought(int row, int col);

/** Placerar en kryss på platsen (row, col) */
void markCross(int row, int col);

/** Returnerar X om kryss är vinnare, 0 om ring är vinnare och blank vinnare saknas */
char getWinner();
```

Vilka **attribut** behöver vi?

Klassen Model med tomma metodkroppar: Gör klart!

```
public class Model {  
    private char[][] board = new char [3][3];  
  
    public Model(){    }  
  
    public void clear(){    }  
  
    public char getMark(int row, int col){    }  
  
    public boolean isFree(int row, int col){    }  
  
    public boolean isInside(int row, int col){    }  
  
    public boolean isFull(){    }  
  
    public void markNought(int row, int col){    }  
  
    public void markCross(int row, int col){    }  
  
    public char getWinner(){    }  
}
```


Implementera View enligt specifikationen

View

```
/** Skapar en presentationsvy av Tre-i-rad-spel utifrån en modell */  
View(Model model);  
  
/** Skriver ut en textuell representation av brädet */  
void showBoard();  
  
/** Skriver ut eventuella vinnare och om brädet eventuellt är fullt */  
showStatus();  
  
/** Uppmanar rätt spelare (KRYSS eller RING) att göra ett drag och läser positionen.  
 * Om isCrossMove är true är det KRYSS som får göra ett drag annars RING.  
 * Returenerar en vektor med {row, col}, där row och col är i intervallet [0, 2] */  
int[] getMove(boolean isCrossMove);
```

Vilka **attribut** behöver vi?

Implementera View enligt specifikationen

Controller

```
/** Skapar en styrenhet för ett Tre-i-rad-spel. */  
Controller();  
  
/** Gör en spelomgång där KRYSS gör första draget om isCrossStarting är true,  
 * annars börjar RING. */  
void play(boolean isCrossStarting);  
  
/** Skapar en Controller-instans och kör en spelomgång. */  
static void main(String[] args);
```

Vilka **attribut** behöver vi?

Spela spelet! Implementera sedan själv. Se sedan hela lösningen här:
[lecture-examples/week09/tictactoe](https://github.com/EDA016/lecture-examples/week09/tictactoe)

Extrauppgift:

Implementera brädet med en alternativ View som använder SimpleWindow