Innehåll

Ett problem

- Hur sparas data …
- T ex när man vill spara resultaten av en tävling
- Exempel med 3 deltagare:

```
public class Competition
{
   private int result1;
   private int result2;
   private int result3;
   …
}
```

- Håller detta för 10, 100 eller ett okänd antal?

Lösningen

- Deklarerar utrymme för många variabler i en enda deklaration
- ➨ Fält (***array***)
- En ***array*** är en sekvens (ordnad lista) av värden
- Varje värde har ett numeriskt ***index***
- `N` element indexeras med `0` till `N-1`
- Exempel:

	0	1	2	3	4	5	6	7	8	9
results	79	87	94	82	67	98	87	81	74	91

Fält

- Alla ***värden har samma datatyp***
- Får vara primitiva datatyper eller klasstyper
- I Java behandlas fält som objekt
- ➨ Måste instansieras med `new`
- ➨ Namnet på fältet är en referens
- ➨ Index är av datatyp `int`

OBS! Vid instansiering av ett fält med objekt instansieras ***ej*** objekten. Det skapas bara utrymmet för rätt antal referenser.

Fält och hakparanteser

- Deklaration

```
typename[] arrayName;
```

- Instansiering

```
arrayName = new typename[numberOfElements];
```

- Åtkomst av element

```
arrayName[index]

// index >= 0  &&  index < numberOfElements
```


Exempel

```
int[] results;
results = new int[10];  // 0..9

int i = 1000;
char[] koder = new char[i];

Triangle triangles[] = new Triangle[i+123];
```

- Se exempel 5.1 (s. 275)

Fält som datatyp

- Fält är en datatyp
- För varje datatyp finns en motsvarande fält-datatyp
- Typen är *typename*[]
- Storleken ingår inte i datatypen
- ➡ Grundtypen bestämmer kompatibiliteten
- ➡ Även fält referenser är polymorfa

Fler exempel

```
int[] results1 = new int[100];
int[] results2 = new int[20];
results1 = results2;
results2 = results1;
```

OK, då samma datatyp

```
Figure figures[] = new Triangle[123];
```

OK, då `Triangle` subklass av `Figure`

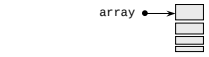
~~`Triangle triangles[] = new Figure[123];`~~

Typfel

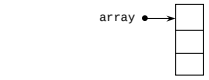
Instansiera fält 1

- Primitiva datatyper

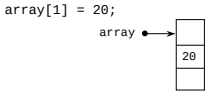
```
int[] array;
```




```
array = new int[3];
```

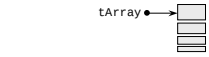



```
array[1] = 20;
```



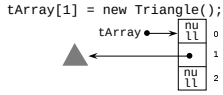
- Klasstyper

```
Triangle[] tArray;
```




```
tArray = new Triangle[3];
```


```
tArray[1] = new Triangle();
```



Instansiera fält 2

- Glöm inte att instansiera fältet före användningen

```
int[] results;

results[0] = 99;
```

- ➡ `NullPointerException`

ett *undantag*, dessa kan bevakas och fångas upp för att hanteras, vilket beskrivs i nästa föreläsning

Initialisera med listor

Hela fältet kan initialiseras vid deklarationen

```
int[] enheter = {147, 323, 89, 933, 540, 269, 97};
char[] kursNiva = {'A', 'B', 'C', 'D'};
```

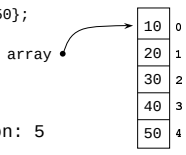
- Längden bestäms av antalet element
- Endast vid deklarationen
- Observera:
  - `New` används inte
  - Ingen explicit storlek

Indexkontroll

- Väl skapat är fältets storlek fixt
- Index måste referera till existerande element
- ➡ Index måste vara i intervallet `0…storlek-1`

- Index kontrolleras dynamiskt (under körning)

```
int[] array = {10, 20, 30, 40, 50};
int i;

i = array[5];
```



- ➡ `ArrayIndexOutOfBoundsException: 5`

Fältets storlek

- Varje fält objekt har ett publikt attribut `length`
- … som ***anger antalet element***, inte högsta index

OBS! Leder ofta till ***off-by-one fel***

- Används ofta i loopar
- Exempel:

Off-by-one fel, eftersom index löper från 0 till `array.length-1`

```
int[] array = …;

for (int i = 1; i <= array.length; i++)
   … array[i] …;

for (int i = 0; i < array.length; i++)
   … array[i] …;
```


Objekt som element

- Elementen i ett fält kan vara objektreferenser

```
Circle[] cirkel = new Circle[5];
```

➨ Fem referenser till objekt av typen `Circle`
OBS! Inga objekt har skapats (referensen är `null`)

- Objekten måste skapas separat

```
cirkel[0] = new Circle();
```

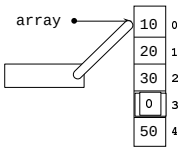
- Via loop

```
for (int i=0;i < cirkel.length,i++)
{
  cirkel[i] = new Circle();
  cirkel[i].changeSize((i+1)*10);
}
```


Fält som parametrar

- Fältreferensen överförs (“kopieras”) och den formella och aktuella parametern blir alias
- Ändringar påverkar båda
- Eftersom storleken inte är del av datatypen får den aktuella parametern ha godtycklig längd

```
int[] array = {10, 20, 30, 40, 50};
….aMethod (array);

public void aMethod (int[] numbers)
{
   …
   numbers[3] = 99;
   …
}
array[3] = 0;
```

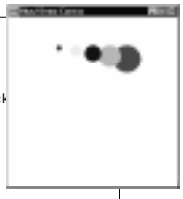


Fält som parametrar: Exempel

```
Object findObjectInArray(Object o, Object[] arrayOfObjects)
{
  int i;      // local variable
  for (i = 0; i < arrayOfObjects.length; i++)
  {
    if (arrayOfObjects[i] == o) // alias
      return o;
  }
  return null;
}
```



```
public class GeoFigurer
{
  Circle[] cirkel = new Circle[5];
  String[] farger = {"red", "yellow", "blue",
                     "green", "magenta", "black

  public GeoFigurer()
  {
    for (int i=0;i < cirkel.length; i++)
    {
      cirkel[i] = new Circle();
      cirkel[i].changeSize((i+1)*10);
      cirkel[i].moveHorizontal((i+1)*25);
    }
    for (int i=cirkel.length-1; i >=0 ; i--)
    {
      cirkel[i].changeColor(farger[i]);
    }
  }// GeoFigurer
…
```



Exempel……

```
public void letaCirkel()
{
   Circle c,funnen;

   c = cirkel[3];
   System.out.println("c = " + c.getColor());

   funnen = (Circle) findObjectInArray(c,cirkel);
   if (funnen == null)
     System.out.println("Fanns inte!" );
   else
     {
       funnen.changeColor("black");
       System.out.println("funnen = " + funnen.getColor());
     }
   System.out.println("c = " + c.getColor());
}// letaCirkel
```

Flerdimensionella fält

- Fält kan ha flera dimensioner
  - En-dimensionella fält motsvarar listor
  - Två-dimensionella fält motsvarar tabeller eller matriser med rader & kolumner
  - Fält av fält av fält av ...
- ➨ Varje dimension har ett eget index
- ➨ Varje dimension har sin egen `length`

- Kan initialiseras med listor

- Se Exempel 5.13 `sumMatrix` [327]

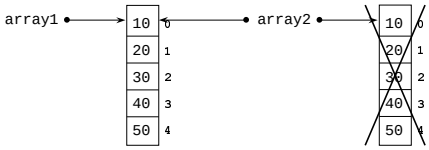
Ännu fler exempel

```
int[][] matrix1 = new int[10][20];
int[][] matrix2 = {{1}, {2, 22}, {3, 33, 333},
   {4, 44, 444, 4444}}
int[][][] matrix3 = new int[10][20][30];

matrix2[0].length == 1;
matrix2[3].length == 4;

matrix1[2][3] = matrix2[1][0];
matrix1[2] = matrix2[1];
matrix3[1] = matrix2;

Triangle[][][][][] fiveDimensional;
```


Kopiera fält 1

- Fältvariabler är referenser
- ➨ Tilldelning gör ingen kopia på elementen i fältet

```
int[] array1 = {10, 20, 30, 40, 50};
int[] array2 = new int[5];

array2 = array1;
```



Kopiera fält 2

- `System.arraycopy` gör kopia element för element
- Vid flerdimensionella fält måste det anropas flera gånger
- Gör lämpligtvis i en (nästad) loop

```
public static void arraycopy (
   Object source, int srcindex,
   Object dest, int destindex, int size)
   throws ArrayIndexOutOfBoundsException,
          ArrayStoreException
```


Klassen `Vector (java.util)`

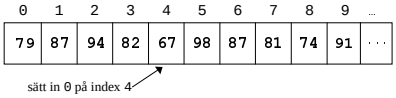
- Ett objekt av klassen `Vector` liknar ett fält
- Men
  - Har dynamisk längd, dvs längden utökas efter behov
  - Lagrar bara referenser till objekt av typen `Object`
  - Man får bara `"Object"` tillbaka
  - Inte samma syntax för indexering
- Implementeras med fält

`Vector`

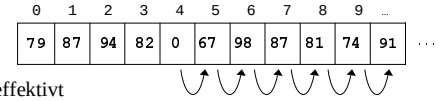
Vector
Vector ( ) + addElement (Object obj) + insertElementAt (Object obj, int index) + setElementAt (Object obj, int index) + removeElementAt (int index) + Object firstElement ( ) + Object lastElement ( ) + Object elementAt (int index) + boolean contains (Object obj) + boolean isEmpty ( ) + int indexOf (Object obj) + int size ( ) + clear ( ) ...

Effektiviteten av `Vector`

- När ett element sätts in flyttas (kopieras) alla efterföljande element en position åt höger



- När ett element tas bort flyttas (kopieras) alla efterföljande element en position åt vänster



➨ Ej effektivt