Arrayer

Arrayer i Java

- Alla *värden har samma datatyp*
- Får vara primitiva datatyper eller klasstyper
- I Java behandlas arrayer som objekt
  - ➡ 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.

Arrayer och hakparanteser

- Deklaration

```
typename[] arrayName;
```

- Instansiering

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

  - *numberOfElements* godtyckligt uttryck av typ *int*

- Åtkomst av element

```
arrayName[index]

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

  - indexerar man sig utanför godkända index inträffar ett undantag (programmet avslutas om vi inte aktivt undviker detta)

Exempel

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


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


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

Arrayer som datatyp

- Arrayer är en datatyp
- För varje datatyp finns en motsvarande array-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;
```

Fler exempel

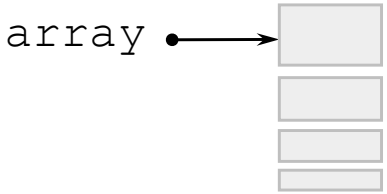
```
int[] results1 = new int[100];
int[] results2 = new int[20];


results1 = results2;
results2 = results1;
```

OK, då samma datatyp

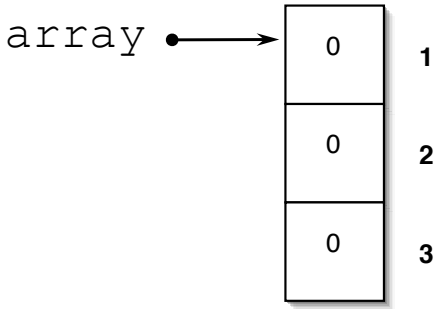
Instansiera arrayer

- Primitiva datatyper

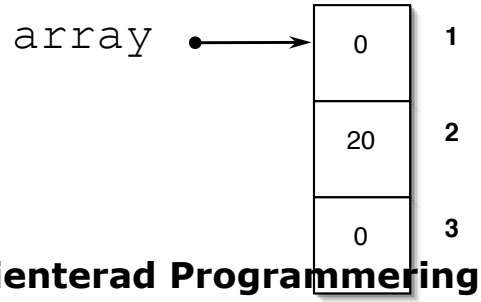
```
int[] array;
```




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

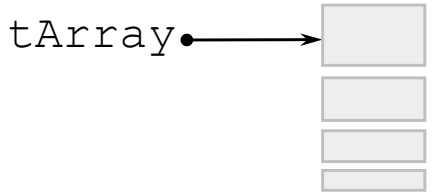



```
array[1] = 20;
```

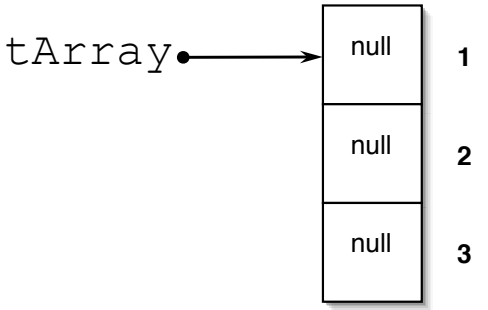


- Klasstyper

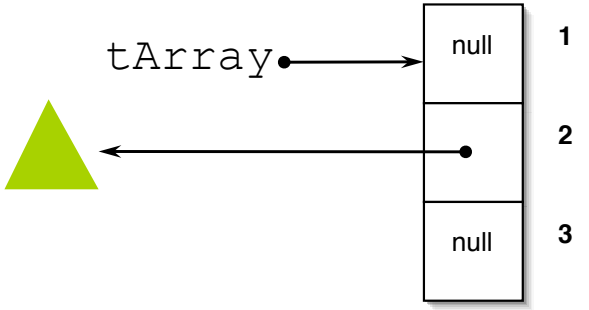
```
Triangle[] tArray;
```




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




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



Instansiera arrayer

- Glöm inte att instansiera arrayen före användningen

```
int[] results;


results[0] = 99;
```

➡ `NullPointerException`

ett *undantag*, dessa kan bevakas och fångas upp för att hanteras, vilket vi kommer titta på på en senare 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

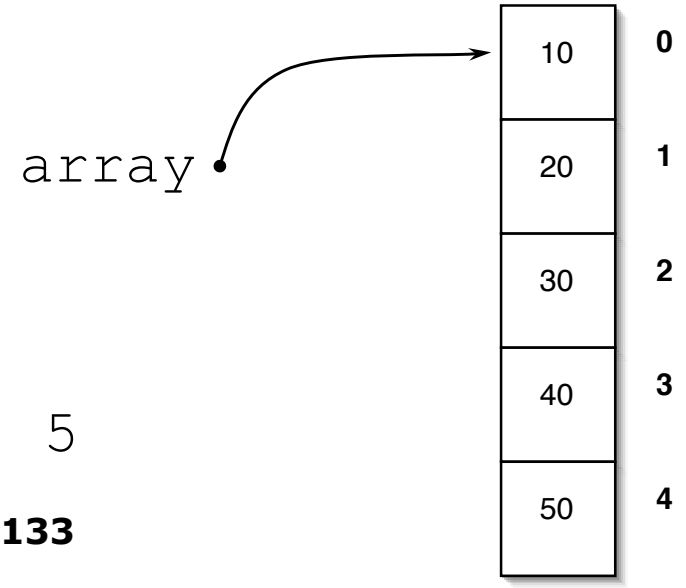
- Väl skapat är arrayens storlek fix
- Index måste referera till existerande element

➡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`



Arrayers storlek

- Varje Array 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:

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

Objekt som element

- Elementen i en array kan vara objektreferenser

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

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

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

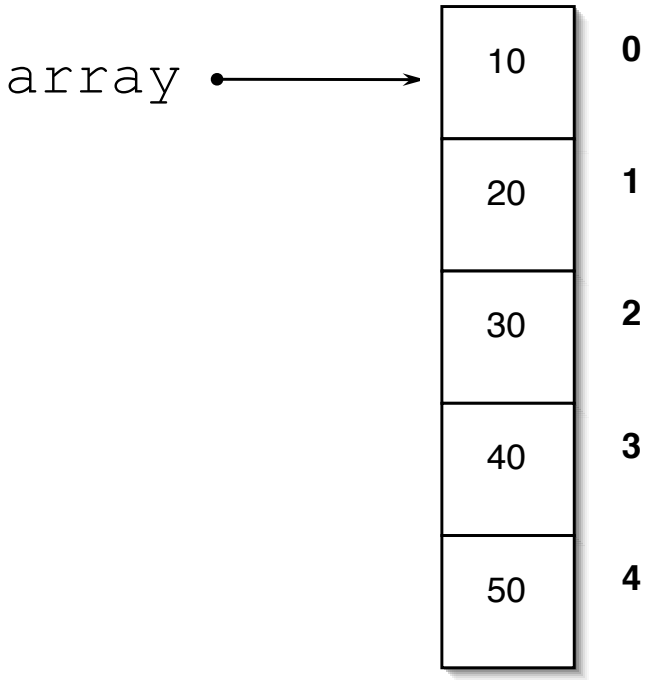
- Objekten måste skapas separat t.ex. m.h.a. loop

```
for (int i=0;i < cirkel.length;i++) {
  cirkel[i] = new Circle();

  cirkel[i].changeSize((i+1)*10);

}
```

Arrayer som parametrar

- Arrayreferensen 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

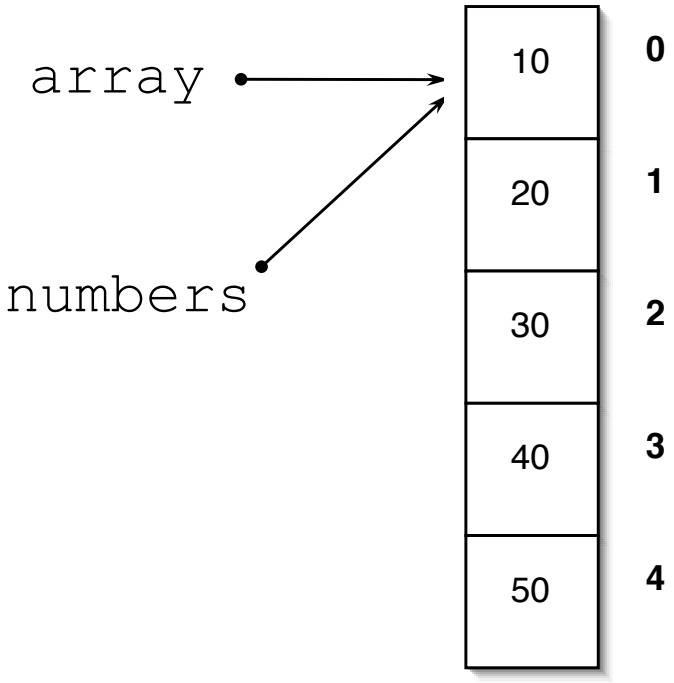
```
int[] array = {10, 20, 30, 40, 50};
….aMethod (array);


public void aMethod (int[] numbers)
{
   …
   numbers[2] = 99;
   …
}
array[2] = 0;
```



Arrayer som parametrar

- Arrayreferensen 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

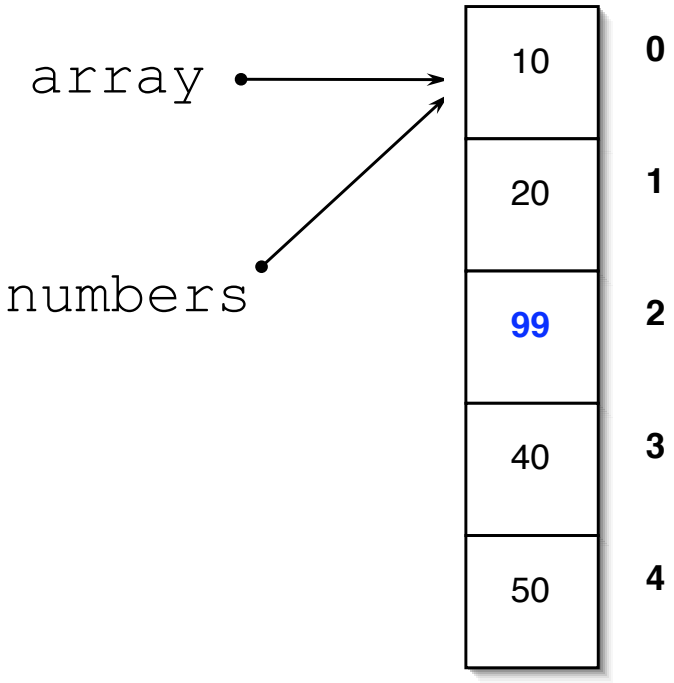
```
int[] array = {10, 20, 30, 40, 50};
….aMethod (array);


public void aMethod (int[] numbers)
{
    …
    numbers[2] = 99;
    …
}
array[2] = 0;
```



Arrayer som parametrar

- Arrayreferensen 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

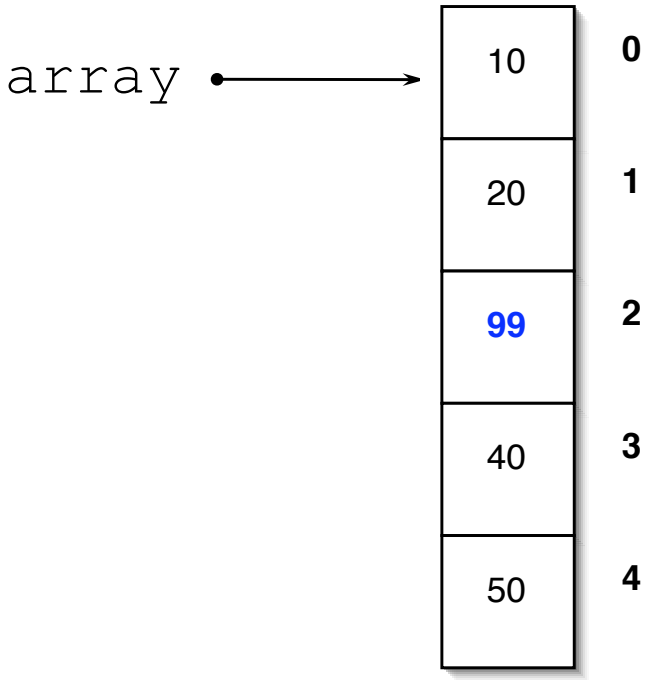
```
int[] array = {10, 20, 30, 40, 50};
….aMethod (array);


public void aMethod (int[] numbers)
{
    …
    numbers[2] = 99;
    …
}
array[2] = 0;
```



Arrayer som parametrar

- Arrayreferensen 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

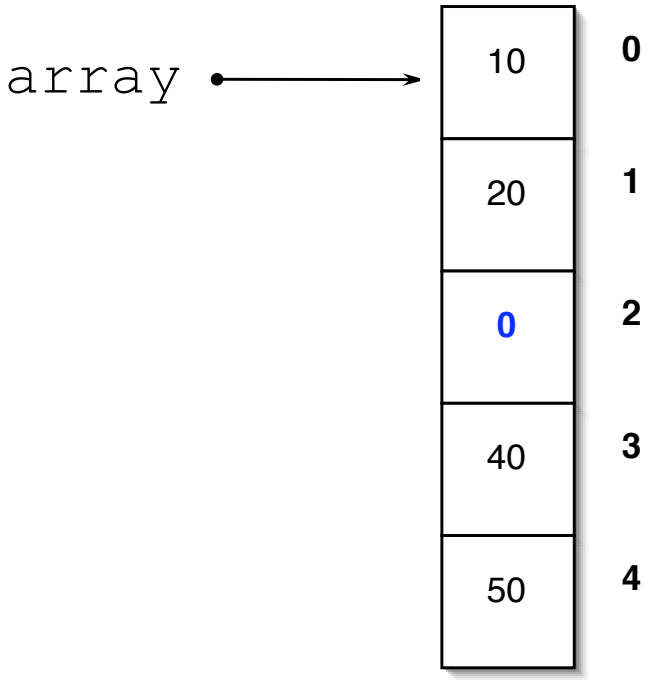
```
int[] array = {10, 20, 30, 40, 50};
….aMethod (array);


public void aMethod (int[] numbers)
{
   …
   numbers[2] = 99;
   …
}
array[2] = 0;
```



Arrayer som parametrar

- Arrayreferensen 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[2] = 99;
   …
}
array[2] = 0;
```



Arrayer som parametrar:

```
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
…
```

Flerdimensionella arrayer

- Arrayer kan ha flera dimensioner
  - En-dimensionella arrayer motsvarar listor
  - Två-dimensionella arrayer motsvarar tabeller eller matriser med rader & kolumner
  - Array av array av array av ...

➡Varje dimension har ett eget index

➡Varje dimension har sin egen `length`

- Kan initialiseras med listor

Ännu fler exempel

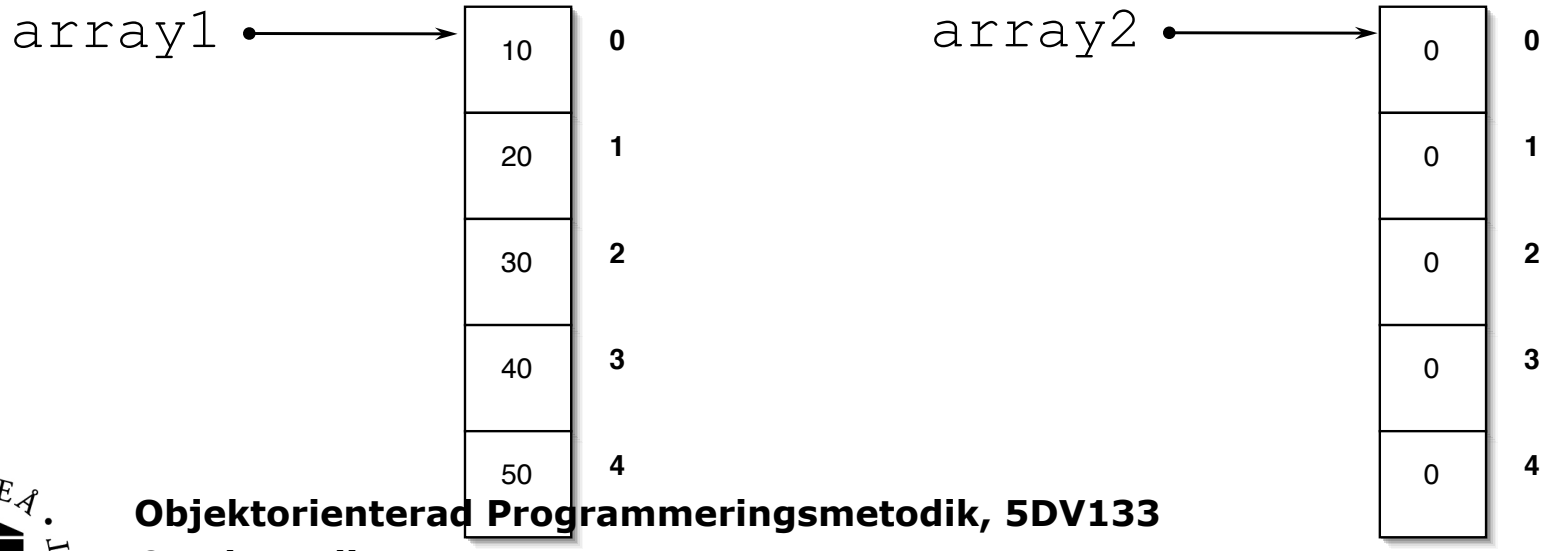
```
int[][] matrix1 = new int[10][20];
int[][] matrix2 = {{1}, {2, 22}, {3, 33, 333},
   {4, 44, 444, 4444}}
int[][][] matrix3 = new int[10][20][30];

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

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

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

Kopiera arrayer

- Arrayvariabler är referenser

➡Tilldelning gör ingen kopia på elementen i fältet

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

int[] array2 = new int[5];

array2 = array1;
```

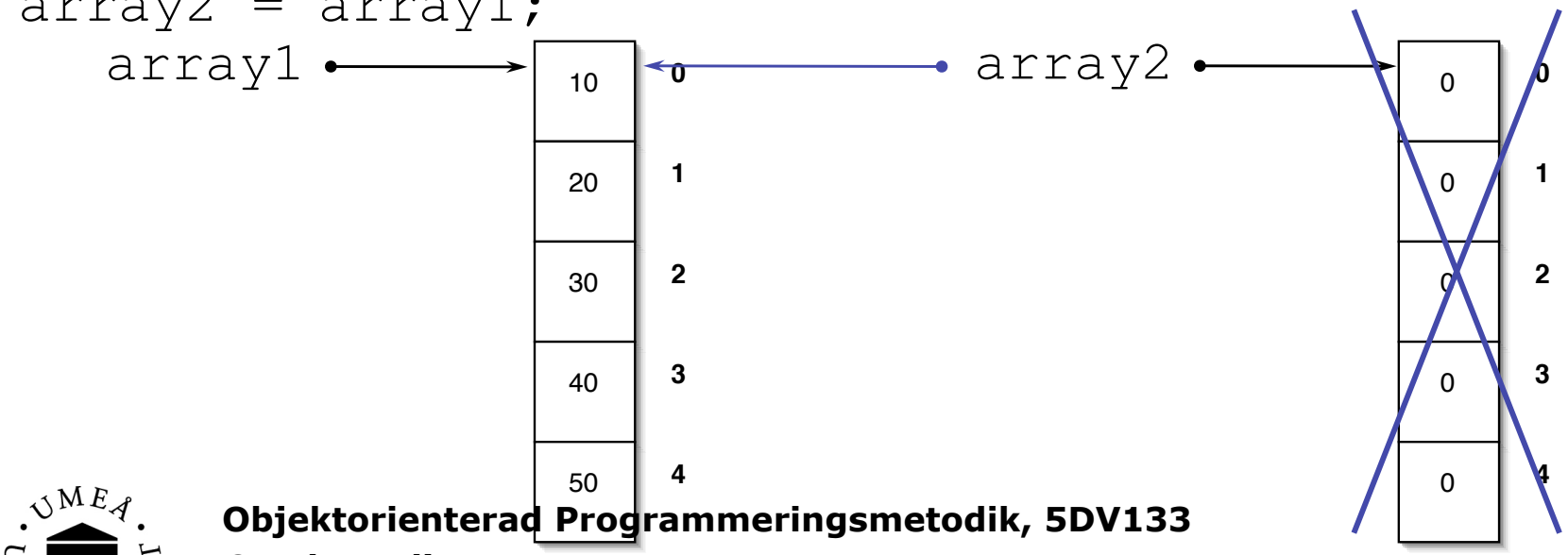


Kopiera arrayer

- Arrayvariabler 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 arrayer

- `System.arraycopy` gör kopia element för element
- Vid flerdimensionella arrayer måste den 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
```

- Vi kan självklart göra det "manuellt" också genom att löpa över alla element och kopiera dessa.

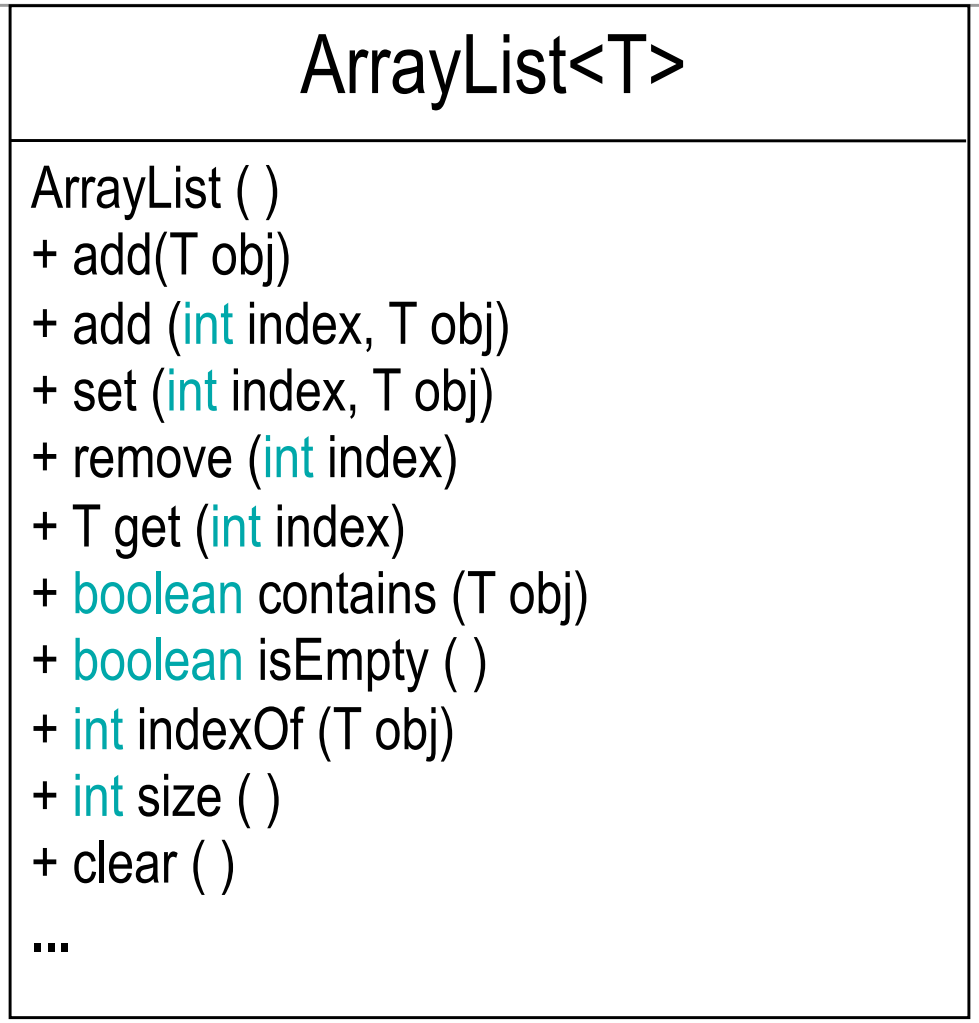
Klassen `ArrayList` (`java.util`)

- Ett objekt av klassen `ArrayList` liknar en array
- Men
  - Har dynamisk längd, dvs längden utökas efter behov
  - Lagrar bara intern referenser till objekt av typen `Object`
    - `Dvs vi kan ej lagra primitiva typer som tex int och double (dessa kan dock konverteras till och från objekttyper automatiskt (Tex int<->Integer)`
  - Inte samma syntax för indexering
- Implementeras med arrayer

Generics

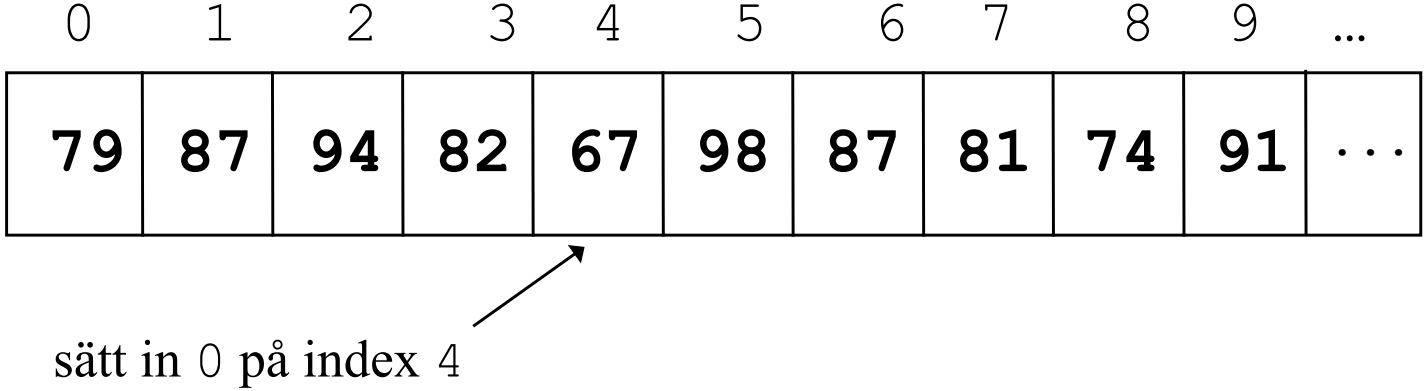
- I tidigare versioner av Java kan man göra generella klasser (tex `ArrayList`) med hjälp av `Object`.
- Fördel är att man behöver skriva koden till listan en enda gång och sen använda den oavsett vad som ska lagras.
- Nackdel är att man i en och samma lista kan lagra olika objekt "huller om buller". (Först en bok, sen ett bibliotek följt av ett hus etc)
- En annan nackdel är att man behöver göra många "typecasts" i kod som använder sig av en klass som utnyttjar `Object`.
- Mha generics så kan man istället specificera (vid konstruktion av objektet) vad som ska lagras

ArrayList

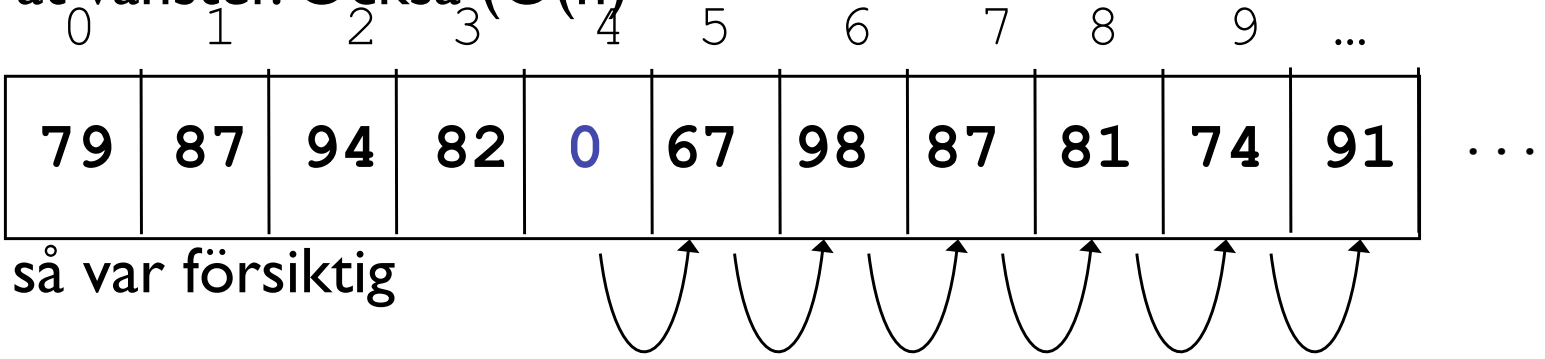


Effektiviteten av `ArrayList`

- När ett element sätts in flyttas (kopieras) alla efterföljande element en position åt höger. Operationen är O(n)



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



➡ Ej effektivt så var försiktig

Autoboxing

- Automatisk konvertering mellan primitiva typer och wrapper-klasserna Integer, Float, Boolean etc
- Ex:

```
Integer i = new Integer(9);
Integer j = new Integer(13);
Integer k = i + j;
```

- Kod kommer att infogas för att göra konverteringen så det kommer inte gå snabbare än om man gör det "manuellt", men koden blir mindre plottrig

Generics och autoboxing

```
import java.util.*;
public class ArrayListTest{
  public static void main(String[] args){
    ArrayList<Integer> myList = new
                      ArrayList<>();
    myList.add(103);
    // Hade inte gått att lägga till en sträng som p nedanstående
   rad!!
    // myList.add(”Hejsan svejsan");
    int testInt = myList.get(0);
    System.out.println(testInt);
  }
}
```

Iteratorer

- Kan användas för att gå igenom samlingar av värden tex i Vector, ArrayList m.fl.

```
Vector<Integer> v=new Vector<Integer>();
for(int i =0;i<10;i++)
  v.add(new Integer(i));
Iterator<Integer> it = v.iterator();
while(it.hasNext())
  … it.next() …
```

For-each

```
ArrayList<Integer> v=new ArrayList<>();
for(int i =0;i<10;i++)
  v.add(new Integer(i));
for(Integer value:v)
  … value …
```

- Funkar för arrayer och alla klasser som implementerar interfacet `Iterable`