Objektorienterad programmering

Föreläsning 6: fält och klassen `String`

Dr. Alex Gerdes | Dr. Carlo A. Furia

Hösttermin 2016

Chalmers University of Technology

Sammanfattning föreläsning 5

CHALMERS

- Abstraktion

- **Abstraktion**

- **Abstraktion**

Fält

Fält

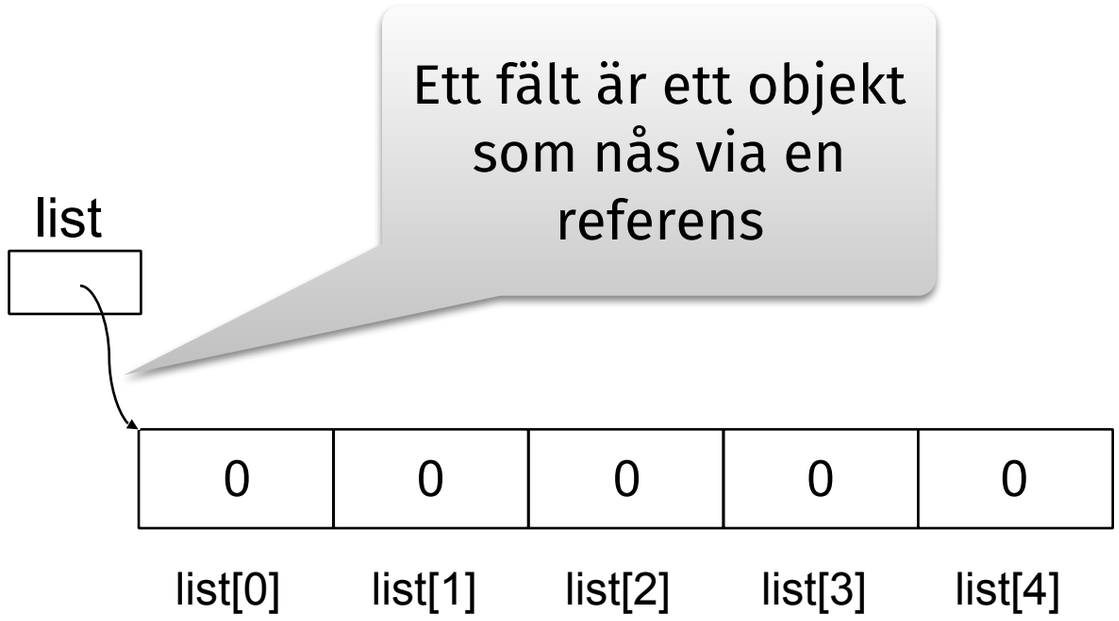
- I ett program hantera man ofta samlingar av objekt av samma typ
- Sådana samlingar vill man vanligtvis kunna *gruppera ihop* till en *sammanhängande struktur*
- För detta ändamål tillhandahåller Java språkkonstruktioner för att hantera *fält*

Fält

- Ett fält är en numrerad samling av element, där varje element är av *samma datatyp* och elementen selekteras med *index*

```
int[] list = new int[5];
```

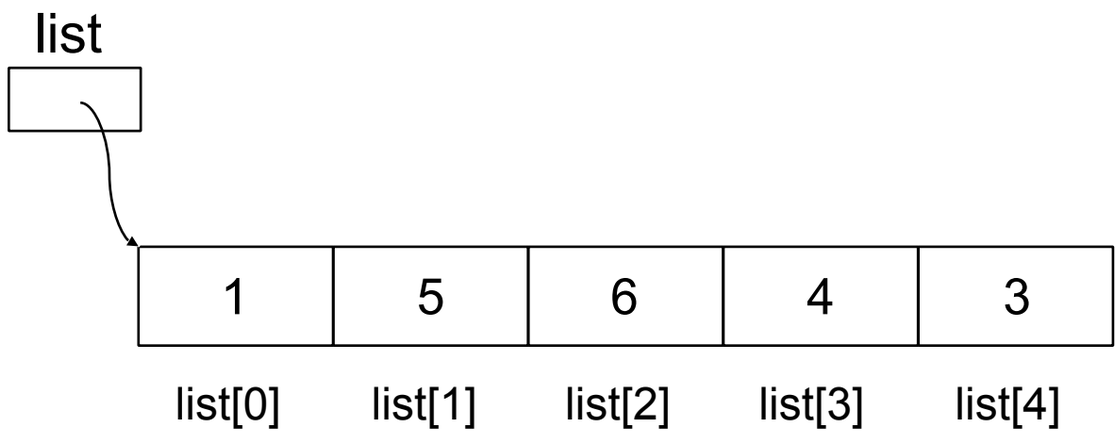
- Indexering sker alltid från 0



- Oinitierade heltal får värdet 0

- Varje enskilt element i ett fält kan handhas individuellt via sitt index:

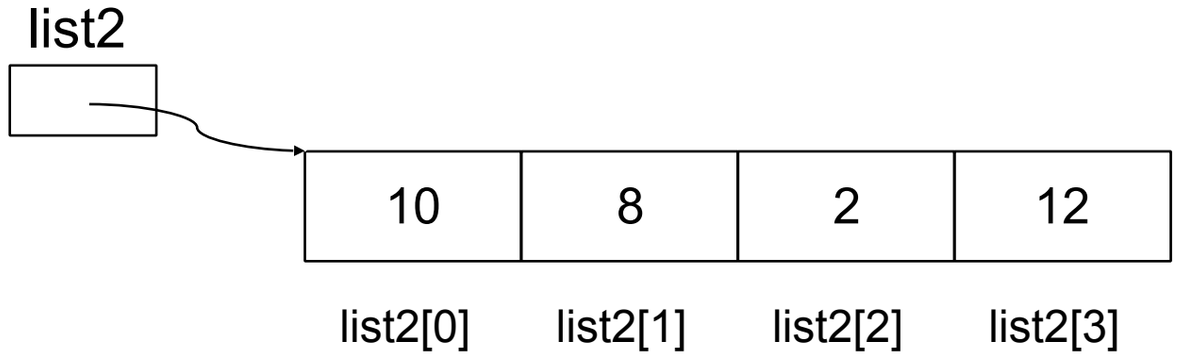
```
list[0] = 1;
list[1] = 5;
list[2] = 6;
list[3] = 4;
list[4] = 3;
```



Fält

- Istället för att skapa ett fält med `new` kan fältet skapas genom att initiera värden till fältet vid deklarationen

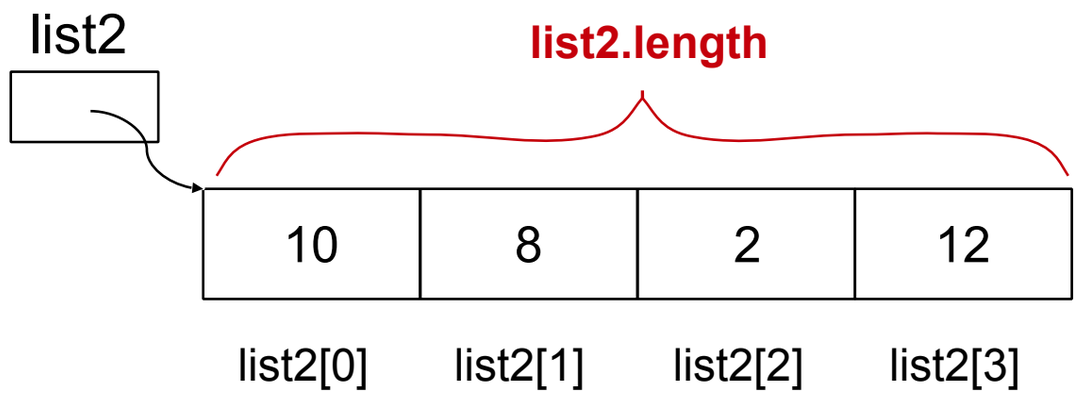
```
int[] list2 = {10, 8, 2, 12};
```



- Antalet värden som då deklareras bestämmer fältets storlek

- Längden av ett fält fås av instansvariabeln `length`

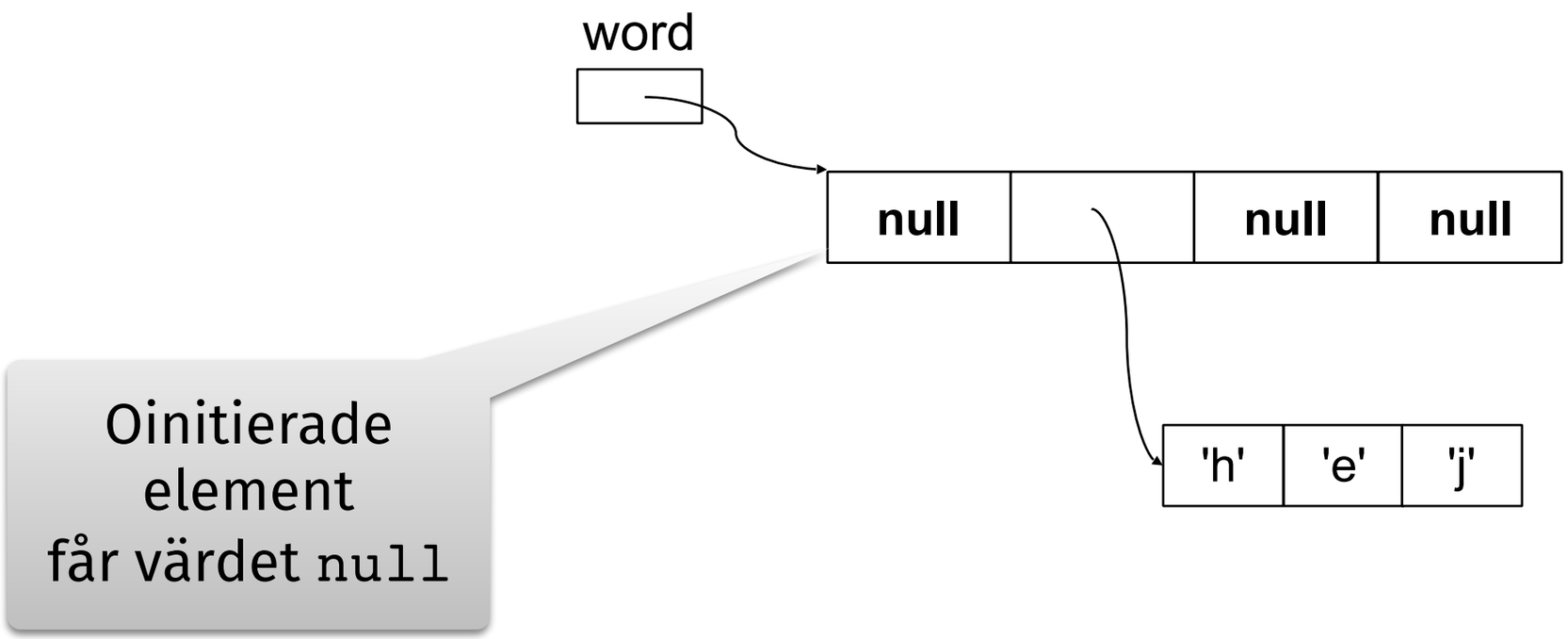
```
int nrOfElements = list2.length;
```



Fält

- Fält kan skapas av godtycklig typ

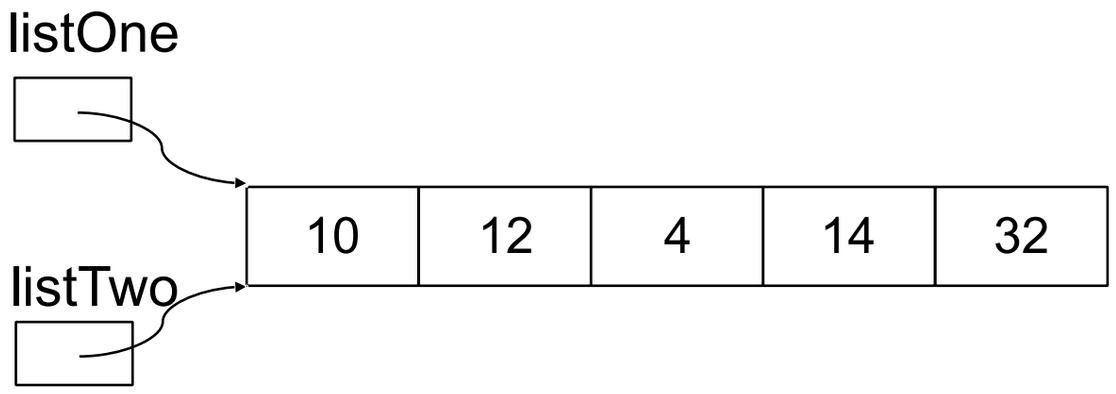
```
double[] numbers = new double[20];
String[] names = {"Adam", "Beda", "Cesar", "David", "Erik"};
String[] word = new String[4];
word[1] = "hej";
```



Fält

- Tilldelning ger ingen kopia, utan en referens till samma fält!

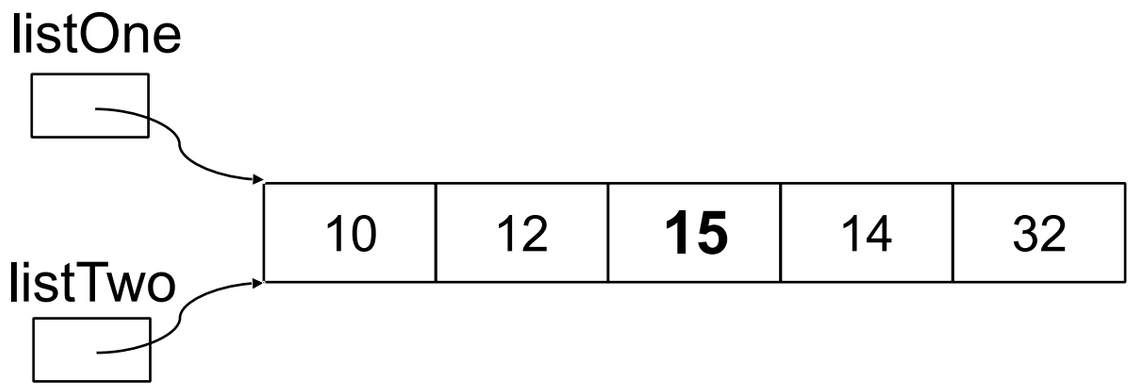
```
int[] listOne = {10, 12, 4, 14, 32};
int[] listTwo = listOne;
```



- Satsen

```
listTwo[2] = 15;
```

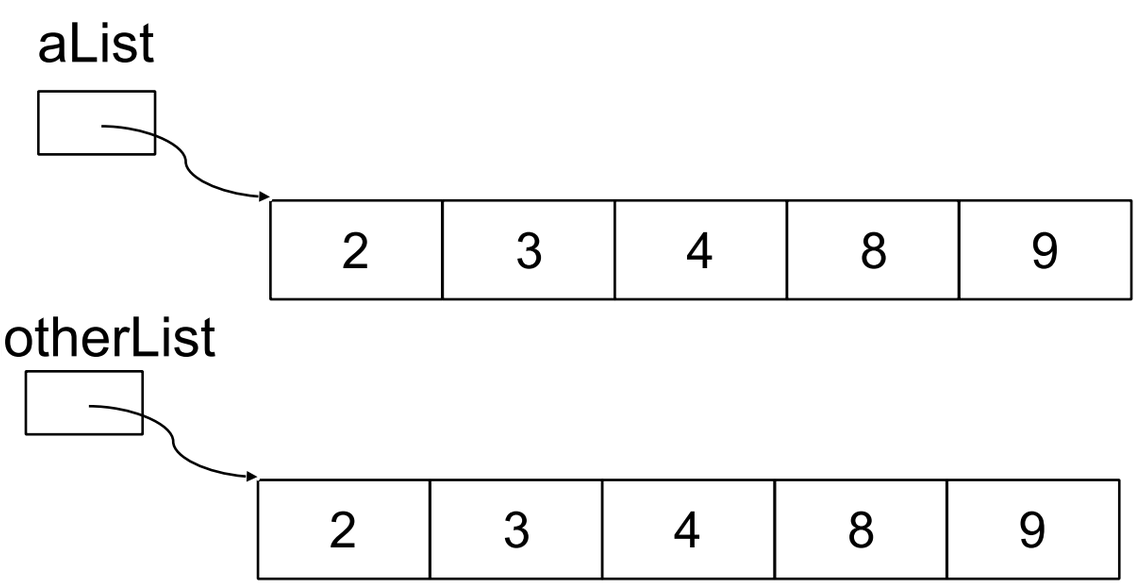
resulterar således i att även elementet `listOne[2]` får värdet 15!



Fält

- Olika referenser även med samma data:

```
int[] aList = {2, 3, 4, 8, 9};
int[] anotherList = {2, 3, 4, 8, 9};
```



- Jämförelsen

  ```
  aList == anotherList
  ```

  ger resultatet `false`, eftersom det är värdet i variablerna `aList` och `anotherList` som jämförs och inte värdet som dessa variabler refererar till!

Att genomlöpa ett fält

- För att genomlöpa alla elementen i ett fält används normalt en `for`-loop

```
int[] list = new int[20];
...
for (int i = 0; i < list.length; i = i + 1) {
  // gör de bearbetningar av elementen
  // som skall göras
}
```

- Summera talen i heltalsfältet `list`

```
// before: list != null
public int sumOfElements(int[] list) {
  int sum = 0;
  for (int i = 0; i < list.length; i = i + 1) {
    sum = sum + list[i];
  }
  return sum;
}
```

Alternativt:
`for-each`

Live code

Quiz!

Klassen `java.util.Arrays`

- I klassen `java.util.Arrays` finns ett antal klassmetoder som är användbara när man arbetar med endimensionella fält:

Metod	Beskrivning
`boolean equals(int[] a, int[] b)`	returnerar `true` om fälten `a` och `b` är lika långa och motsvarande komponenter är lika, annars returneras `false`.
`int[] copyOf(int[] f, int length)`	returnerar en kopia av fältet `f` med längden `length`, om kopian är kortare än `f` sker en trunkering, om kopian är längre fylls kopian ut med 0:or
`int[] copyOfRange(int[] f, int from, int to)`	returnerar ett fält som innehåller elementen från index `from` till index `to` i fältet `f`
`String toString(int[] f)`	returnerar en textrepresentation av fältet `f`, på formen $[e_1, e_2, \ldots, e_n]$
`void fill(int[] f, int value)`	sätter alla element i fältet `f` till värdet `value`
`void fill(int[] f, int from, int to, int value)`	sätter alla elementen från index `from` till index `to` i fältet `f` till värdet `value`

Klassen `java.util.Arrays`

Metod	Beskrivning
`void sort(int[] f)`	sorterar elementen i fältet f i stigande ordning
`void sort(int[] f, int from, int to)`	sorterar element från index `from` till index `to` i fältet `f` i stigande ordning
`int binarySearch(int[] f, int key)`	returnerar index för `key` om `key` finns i fältet `f`, annars returneras ett värde < 0; observera att fältet `f` måste vara sorterat!
`int binarySearch(int[] f, int from, int to, int key)`	returnerar index för `key` om `key` finns i fältet `f` mellan index `from` och index `to` annars returneras ett värde < 0

- Samtliga dessa metoder finns också för andra typer av fält, t.e. `double[]`, `boolean[]` och `char[]`!

Användning av metoder i `java.util.Arrays`

- Att sortera och skriva ut ett fält

```
import java.util.Arrays;
...
int[] list = {5, 4, 3, 8, 1, 9, 6, 7, 2};
Arrays.sort(list);
System.out.println(Arrays.toString(list));
```

- Utskriften som erhålls blir:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Att lokalisera ett element i ett sorterat fält

```
import java.util.Arrays;
...
int[] list = {5, 4, 3, 8, 1, 9, 6, 7, 2};
Arrays.sort(list); // sortera fältet
int index = Arrays.binarySearch(list, 9);
if (index >= 0)
  System.out.println("Talet 9 finns i index " + index);
else
  System.out.println("Talet 9 finns INTE i fältet!");
```

Observera att sortering kan innebära att man ”förstör” fältet, ifall om den inbördes ordningen av elementen i fältet har betydelse för applikationen.

Några exempel

Att söka i ett *osorterat* fält

- **Uppgift**: implementation av en metod som returnerar `true` om ett visst värde finns i ett givet fält, annars returnerar metoden `false`

- Ett första försök: använd en `for`-sats för genomsökning av hela listan

```
// before: list != null
public static boolean isInList(int[] list, int target) {
  boolean found = false;
  for (int index = 0; index < list.length; index = index + 1) {
    if (target == list[index]) {
      found = true;
    }
  }
  return found;
}
```

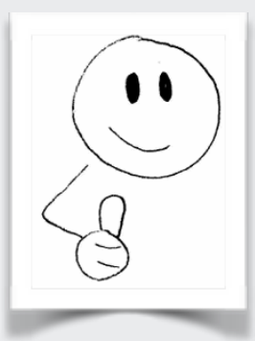
När vi funnit vad vi söker fortsätter sökningen ändock till slutet av listan! Onödigt! Sluta när vi hittat vad vi söker!

Att söka i ett *osorterat* fält

- En bättre lösning: använd en `while`-sats och sluta när vi funnit det vi söker

```
// before: list != null
public static boolean isInList(int[] list, int target) {
  int index = 0;
  boolean found = false;
  while (index < list.length && !found) {
    if (target == list[index]) {
      found = true;
    }
    index = index + 1;
  }
  return found;
}
```

Avbryter sökningen när vi funnit det vi söker eller då hela fältet är genomsökt



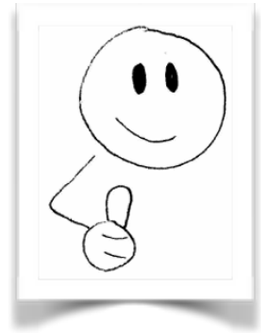
Att söka i ett *osorterat* fält

- Alternativa implementationer

```
// before: list != null
public static boolean isInList(int[] list, int target) {
  int index = 0;
  while (index < list.length &&  target != list[index]) {
    index = index + 1;
  }
  return index < list.length;
}
```

```
// before: list != null
public static boolean isInList(int[] list, int target) {
  return isAt(list, 0, target);
}

private static boolean isAt(int[] list, int i, int target) {
  boolean res = false;
  if (i < list.length)
    res = target == list[i] || isAt(list, ++i, target);
  return res;
}
```



Att söka i ett *osorterat* fält

- Implementation av metoder som returnerar första respektive sista index för ett givet värde om värdet finns i ett givet fält, annars returneras -1 *(Varför -1?)*

```java
// before: list != null
public static int firstIndexOf(int[] list, int target) {
  int index = 0;

  while (index < list.length && target != list[index])
    index = index + 1;

  if (index >= list.length)
    index = -1;

  return index;
}
```

Uppdatera `isInList`!

```java
// before: list != null
public static int lastIndexOf(int[] list, int target) {
  int index = list.length - 1;
  while (index >= 0 && target != list[index])
    index = index - 1;
  return index;
}
```

Eget bibliotek med fältmetoder

- De metoder vi implementerat ovan är handhar situationer som är vanligt återkommande delproblem i många skilda sammanhang
- Det är därför mycket lämpligt all lägga dessa metoder i en och samma klass så att de kan återanvändas i olika tillämpningar
- Här placerar vi metoderna i en klass med namnet `ArrayUtils`

```
public class ArrayUtils {
  // Metoden returnerar värdet true om target finns i fältet list,
  // annars returnerar metoden värdet false.
  // förvillkor: list ≠ null
  public static boolean isInList(int[] list, int target) {...}
  // Metoden returnerar index för första förekomsten av target
  // i fältet list, finns inte target i list returneras värdet -1
  // förvillkor: list ≠ null
  public static int firstIndexOf(int[] list, int target) {...}
  // Metoden returnerar index för sista förekomsten av target
  // i fältet list, finns inte target i list returneras värdet -1
  // förvillkor: list ≠ null
  public static int lastIndexOf(int[] list, int target) {...}
  ...
}
```

Problemexempel

- Skriv en metod

```
public static int[] removeAllDuplicates(int[] list)
```

som tar ett heltalsfält list och returnerar ett nytt fält vilket innehåller samma element som list där alla eventuella dubbletter är borttagna

- **Exempel**: antag att följande deklaration har gjorts

```
int[] vekt = {1, 4, 1, 2, 4, 5, 12, 3, 2, 4, 1};
```

ett anrop av `removeAllDuplicates(vekt)` skall returnera ett fält med följande utseende `{1, 4, 2, 5, 12, 3}`

Design

- **Förvillkor**: `list != null`

- **Diskussion**: vi börjar med att skapar ett nytt tomt fält, som vi kan kalla `res`. Sedan tar vi ett element i taget från fältet list och lägger i detta element i `res` om elementet inte redan finns i `res`

- **Algoritm**:
    1. så länge det finns fler element kvar i `list`
        1.1. `if` (nästa element i `list` inte finns i `res`)
            1.1.1. lägga till nästa element i `res`
    2. Returnera `res`

- **Datarepresentation**:
    - `res` är av datatypen `int[]`

Implementation

```java
import java.util.Arrays;

public class NoDuplicates {
  public static int[] removeAllDuplicates(int[] list) {
    int[] res = new int[0];

    for (int i = 0; i < list.length; i = i + 1)
      if (!isInList(list[i], res))
        res = add(list[i], res);

    return res;
  }

  // Add x to end of the array xs, return result in new array
  public static int[] add(int x, int[] xs) {
    int[] ys = new int [xs.length + 1];

    // Copy all elements from xs
    for (int i = 0; i < xs.length; i = i + 1)
      ys[i] = xs[i];

    // Add x at the end
    ys[xs.length] = x;

    return ys;
  }
}
```

Parallella fält

- Två fält kallas för *parallella fält* om den data som finns i motsvarande index i de båda fälten är logiskt relaterade till varandra på något sätt

- **Exempel**: antag att vi har en golftävling med 5 deltagare; vi kan lagra deltagarnas namnen i en fält, deltagarnas startnummer i ett annat fält och deltagarnas resultat i ett tredje fält:

name	startNr	score
"Kalle"	4	74
"Anna"	5	67
"Stina"	2	73
"Åsa"	1	70
"Sven"	3	68

```
String[] name = new String[5];
int[] startNr = new int[5];
int[] score   = new int[5];
```

- Är dessa fält parallella gäller att varje index *k* i fältet namn är relaterat till index *k* i fälten `startNr` och score, dvs "Stina" hade startnummer 2 och gick banan på 73 slag

I just detta exempel hade det varit bättre att skapat en klass `GolfPlayer` med instansvariablerna `name`, `startNr` samt `score` och istället använt ett enda fält med objekt av denna klass

Fält som uppslagstabeller

- Fält kan användas som uppslagstabeller

```
// before: none
public static String getWeekday(int dayNumberOfWeek) {
  if      (dayNumberOfWeek == 1) return "Monday";
  else if (dayNumberOfWeek == 2) return "Tuesday";
  else if (dayNumberOfWeek == 3) return "Wednesday";
  else if (dayNumberOfWeek == 4) return "Thursday";
  else if (dayNumberOfWeek == 5) return "Friday";
  else if (dayNumberOfWeek == 6) return "Saturday";
  else if (dayNumberOfWeek == 7) return "Sunday";
  else return "Illegal day number!";
}
```

```
// before: dayNumberOfWeek >= 1 && dayNumberOfWeek <= 7
public static String getWeekday(int dayNumberOfWeek) {
  final String[] weekdays = {"Monday", "Tuesday", "Wednesday",
                             "Thursday", "Friday", "Saturday", "Sunday"};
  return weekdays[dayNumberOfWeek - 1];
}
```

Parametrar till `main`

- Metoden main har en parameterlista som utgörs av ett fält av strängar:

```
public static void main(String[] args)
```

- Detta innebär att man kan ge indata till `main`-metoden via parameterlistan
- Betrakta `main`-metoden i klassen `Demo` nedan; vad `main` gör är att skriva ut de strängar som finns i dess parameter `args`

```
public class Demo {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i = i + 1)
      System.out.println("Argument " + i + " = " + args[i]);
  }
}
```

- Innehåller parametern args strängarna "Anna", "Beda" och "Doris" blir utskriften

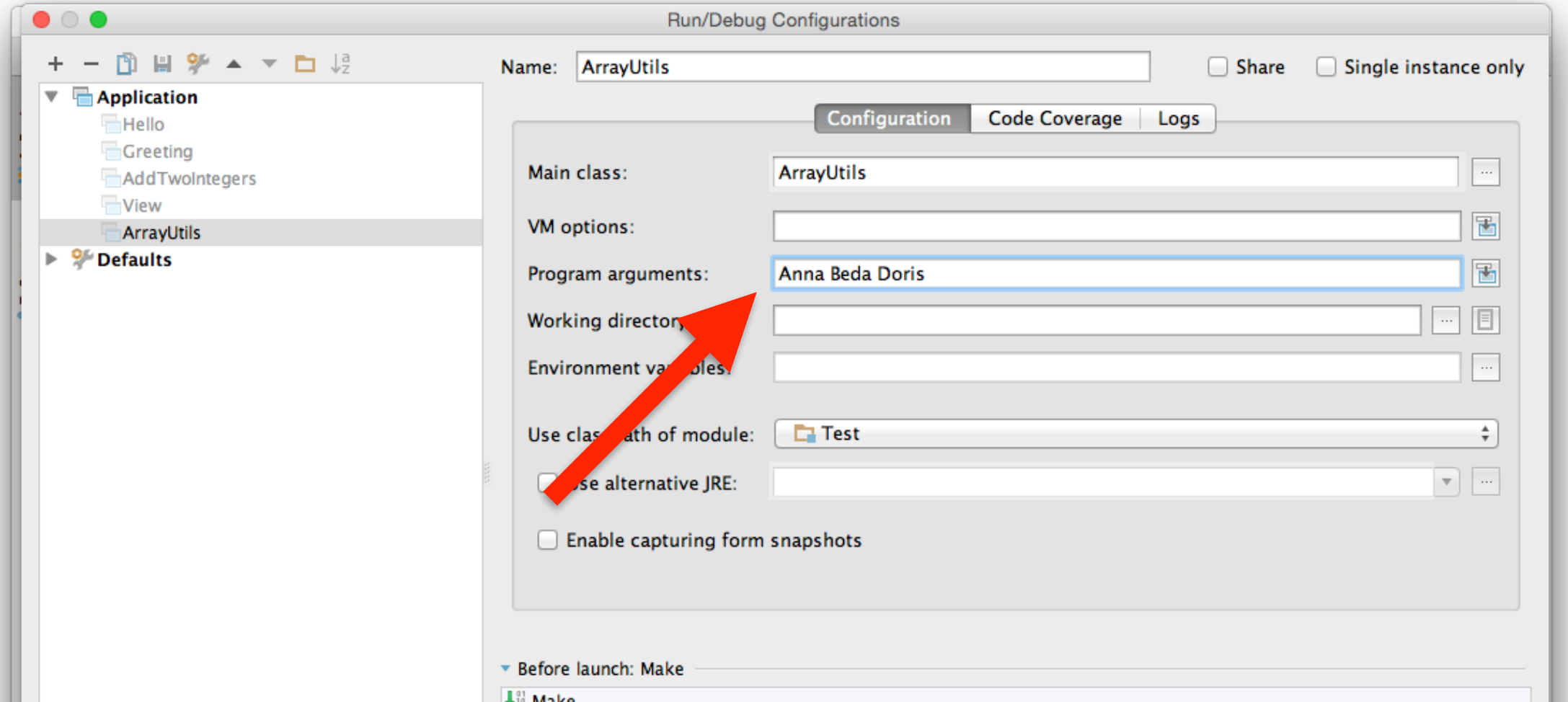
```
Argument 0 = Anna
Argument 1 = Beda
Argument 2 = Doris
```

Parametrar till `main`

- Argumenten till main-metoden ges när exekveringen av programmet startas. Startas exekveringen från kommandofönstret skriver man alltså

```
java Demo Anna Beda Doris
```

- Sker exekveringen från IntelliJ skapar man ett kommandorad i vilket argumenten ges, genom att trycka på ”Edit Configurations" i menyn ”Run" och sedan fylla i ”Program Arguments”



Paus

Klassen `String`

Standardklassen `String`

- Texter handhas i Java med standardklassen `String`
- Ett objekt av klassen `String` består av en följd av tecken, dvs element av typen `char`
- Ett objekt av klassen `String` kan inte förändras efter att det har skapats, dvs objekten är *icke-muterbara*

```
  String
 ....................................
- char[] value
- int count
 ....................................
+ String()
+ String(char[] value)

+ length() : double
+ charAt(int index) : int
+ …
```

Internt i klassen `String` lagras teckensträngen i ett teckenfält

Metoder i standardklassen `String`

Metod	Beskrivning
`int length()`	ger antal tecken i strängen
`char charAt(int pos)`	ger tecknet i position `pos`
`int indexOf(char c)`	ger index för första förekomsten av tecknet `c`, om `c` inte finns returneras -1
`int lastIndexOf(char c)`	ger index för sista förekomsten av tecknet c, om c inte finns returneras -1
`int indexOf(String str)`	ger index för första förekomsten av strängen `str`, om `str` inte finns returneras -1
`int lastIndexOf(String str)`	ger index för sista förekomsten av strängen `str`, om `str` inte finns returneras -1
`boolean equals(String str)`	ger `true` om den aktuella strängen och strängen `str` är lika, annars returneras `false`
`boolean equalsIgnoreCase(String str)`	jämför aktuell sträng med strängen `str` utan hänsyn till versaler och germaner; ger `true` om den aktuella strängen och strängen `str` är lika, annars returneras `false`
`String trim()`	ger en kopia av strängen där inledande och avslutande blanktecken är borttagna

Metoder i standardklassen `String`

Metod	Beskrivning
`int compareTo(String str)`	gör alfabetisk jämförelse mellan aktuell sträng och `str`; ger värdet 0 om aktuell sträng och argumentet är alfabetiskt lika, ett värde mindre än 0 om argumentet är större och ett värde större än 0 om argumentet är mindre
`String concat(String str)`	ger en sträng där `str` har lagts till efter aktuell sträng
`String replace(char old, char new)`	ger en kopia av den aktuella strängen där alla förekomster av tecknet `old` har bytts ut mot tecknet `new`
`String substring(int start)`	ger delsträngen från position `start` till slutet av strängen
`String substring(int start, int end)`	ger delsträngen börjar i position `start` och slutar i position `end - 1`
`String toLowerCase(String str)`	ger en kopia av strängen där alla versaler har bytts mot gemener
`String toUpperCase(String str)`	ger en kopia av strängen där alla gemener har bytts mot versaler
`char[] toCharArray()`	returnerar strängen som ett fält av tecken
`static String value(int n)`	returnerar texten som motsvarar heltalet `n`

Fler metoder finns i klassen `String`

Metoder i standardklassen `Character`

CHALMERS

Metod	Beskrivning
`int getNumericValue(char ch)`	ger Unicode för ch
`boolean isDigit(char ch)`	ger värdet `true` om `ch` är en siffra, annars returneras värdet `false`
`boolean isLetter(char ch)`	ger värdet `true` om `ch` är en bokstav, annars returneras värdet `false`
`boolean isLetterOrDigit(char ch)`	ger värdet `true` om `ch` är en siffra eller bokstav, annars returneras värdet `false`
`boolean isLowerCase(char ch)`	ger värdet `true` om `ch` är en liten bokstav, annars returneras värdet `false`
`boolean isUpperCase(char ch)`	ger värdet `true` om `ch` är en stor bokstav, annars returneras värdet `false`
`char toLowerCase(char ch)`	om `ch` är en stor bokstav returneras motsvarande lilla bokstav annars returneras `ch`
`char toUpperCase(char ch)`	om `ch` är en liten bokstav returneras motsvarande stora bokstav annars returneras `ch`

Problemexempel

Problemexempel

- Skriv ett Java-program som läser en mening från terminalen och räknar antalet vokaler, konsonanter och siffror

- **Analys**:
  - Indata: en sträng av godtyckliga tecken
  - Utdata: utskrift av hur många vokaler, konsonanter respektive siffror som finns i den inlästa strängen

- **Exempel**: strängen ”1xfg2ÄåÖabcdeHI,% 3” ger utskriften:

```
Antalet vokaler är 6
Antalet konsonanter är 7
Antalet siffror är 3
```

Design

- **Design**: Att beräkna antalet siffror, antalet konsonanter respektive antalet vokaler är tre separata delproblem, varför vi skriver en metod för var och ett av dessa delproblem:
  - `int getNrOfDigits(String str)` returnerar antalet siffror i strängen `str`
  - `int getNrOfVowels(String str)` returnerar antalet vokaler i strängen `str`
  - `int getNrOfConsonants(String str)` returnerar antalet konsonanter i strängen `str`

- **Algoritm för huvudprogrammet**:
  1. Läs strängen `indata`
  2. `antSiffror = getNrOfDigits(indata);`
  3. `antVokaler = getNrOfVowels(indata);`
  4. `antKonsonanter = getNrOfConsonants(indata);`
  5. Skriv ut `antVokaler`, `antKonsonanter` och `antSiffror`

- **Datarepresentation**:
  - `indata` är av datatypen `String`
  - `antSiffror`, `antVokaler`, `antKonsonanter` är av typen `int`

Design

- För att avgöra om ett tecken är en siffra finns metoden `isDigit` i klassen `Character`

- **Algoritmen** för metoden `int getNrOfDigits(String str)` blir därför:

  1. Sätt `number = 0`
  2. För varje tecken `ch` i `str`

     2.1.
     ```
     if (Character.isDigit(ch))
          number = number + 1;
     ```

  3. `return number`

- För att avgöra om ett tecken är en bokstav finns metoden `isLetter` i klassen `Character`. Vi behöver dock särskilja om en bokstav är en vokal eller en konsonant varför vi inför en metod:

  `boolean isVowel(char ch)` returnerar `true` om `ch` är en vokal, annars `false`

- **Algoritmen** för metoden `int getNrOfVowels(String str)` blir därför:

  1. Sätt `number = 0`
  2. För varje tecken `ch` i `str`

     2.1.
     ```
     if (isVowel(ch))
          number = number + 1;
     ```

  3. `return number`

Design

- **Algoritmen** för metoden `int getNrOfConsonants(String str)` blir:

  1. Sätt `number = 0`
  2. För varje tecken ch i str

     ```
     2.1. if (Character.isLetter(ch) && !isVowel(ch))
             number = number + 1;
     ```

  3. `return number`

- I metoden boolean `isVowel(char ch)` deklarerar vi en sträng som innehåller de bokstäver som är vokaler:

  ```
  String vowels = "aeiouyåääö";
  ```

- För att avgöra om tecknet `ch` är en vokal använder vi metoden `indexOf` på strängen `vowels`. Anropet

  ```
  vowels.indexOf(ch)
  ```

  ger värdet -1 om tecknet `ch` inte finns bland de tecken som ingår i strängen `vowels`, annars ger anropet ett heltal större eller lika med 0

- Eftersom strängen `vowels` innehåller endast de "små" vokalerna, men `ch` givetvis kan vara en "stor" vokal måste `ch` översättas till sin "lilla" motsvarighet; detta görs med metoden `toLowerCase` som finns i klassen `Character`

Implementation

```
import javax.swing.*;
public class Count {
  public static void main(String[] args) {
    String indata = JOptionPane.showInputDialog("Ge en mening: ");

    int nrOfVowels      = getNrOfVowels(indata);
    int nrOfConsonants = getNrOfConsonants(indata);
    int nrOfDigits      = getNrOfDigits(indata);

    String msg = "Antalet vokaler är " + nrOfVowels
               + "\nAntalet konsonanter är " + nrOfConsonants
               + "\nAntalet siffror är " + nrOfDigits;

    JOptionPane.showMessageDialog(null, msg);
  }

  // before: str != null
  private static int getNrOfDigits(String str) {
    int number = 0;
    for (int pos = 0; pos < str.length(); pos = pos + 1) {
      if (Character.isDigit(str.charAt(pos)))
        number = number + 1;
    }
    return number;
  }
```

Implementation huvudprogram

CHALMERS

```
private static boolean isVowel(char ch) {
  final String VOCALS = "aeiouyåäö";
  return VOCALS.indexOf(Character.toLowerCase(ch)) != -1;
}

// before: str != null
private static int getNrOfVowels(String str) {
  int number = 0;
  for (int pos = 0; pos < str.length(); pos = pos + 1) {
    if (isVowel(str.charAt(pos)))
      number = number + 1;
  }
  return number;
}

// before: str != null
private static int getNrOfConsonants(String str) {
  int number = 0;
  for (int pos = 0; pos < str.length(); pos = pos + 1) {
    char ch = str.charAt(pos);
    if (Character.isLetter(ch) && !isVowel(ch))
      number = number + 1;
  }
  return number;
}
}
```

Optimalisering

CHALMERS

```
public class Count {
  public static void main(String[] args) {
    String indata = JOptionPane.showInputDialog("Ge en mening: ");
    int[] stats   = analyzeString(indata);

    String msg = "Antalet vokaler är " + stats[0]
               + "\nAntalet konsonanter är " + stats[1]
               + "\nAntalet siffror är " + stats[2];

    JOptionPane.showMessageDialog(null, msg);
  }

  private static int[] analyzeString(String str) {
    int digits = 0, vowels = 0, consonants = 0;

    for (int pos = 0; pos < str.length(); pos = pos + 1) {
      char ch = str.charAt(pos);
      if (Character.isDigit(ch))
        digits++;
      else if (Character.isLetter(ch)) {
        if (isVowel(ch))
          vowels++;
        else
          consonants++;
      }
    }
    int[] result =  {vowels, consonants, digits};
    return result;
  }
  …
```