



Objektorienterad programmering

Föreläsning 6: fält och klassen `String`

Dr. Alex Gerdes | Dr. Carlo A. Furia

Hösttermin 2016

Chalmers University of Technology

- Final deadline nära för labb 1
- Göra något annat än TDA540 i labbsalar MT0/9

- Abstraktion
- **Abstraktion**
- **Abstraktion**

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*

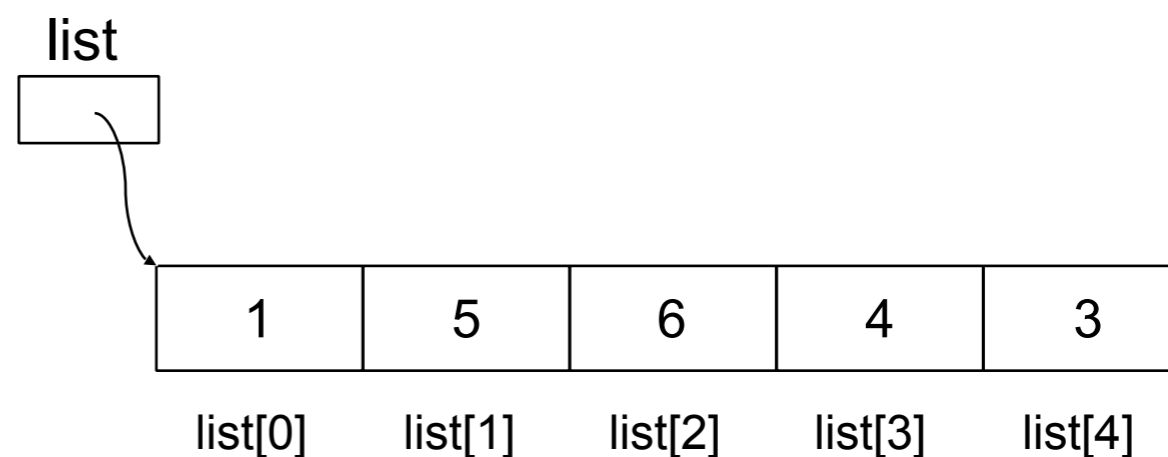
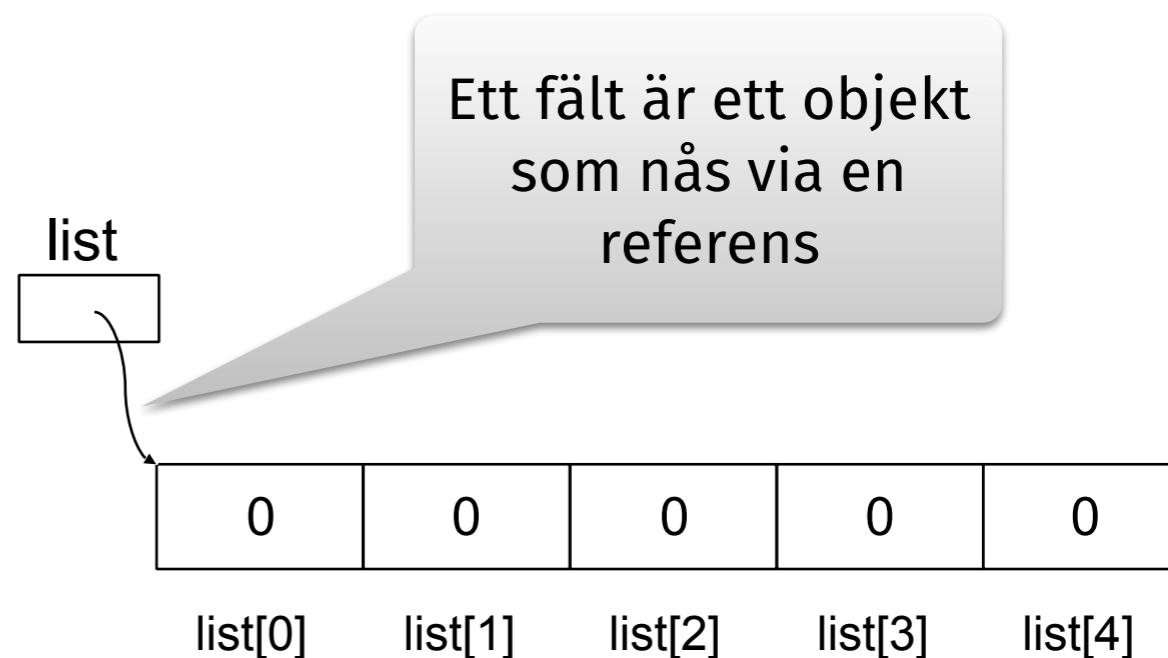


- Ett fält är en numrerad samling av element, där varje element är av samma datatyp och elementen selekteras med *index*

```
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;
```

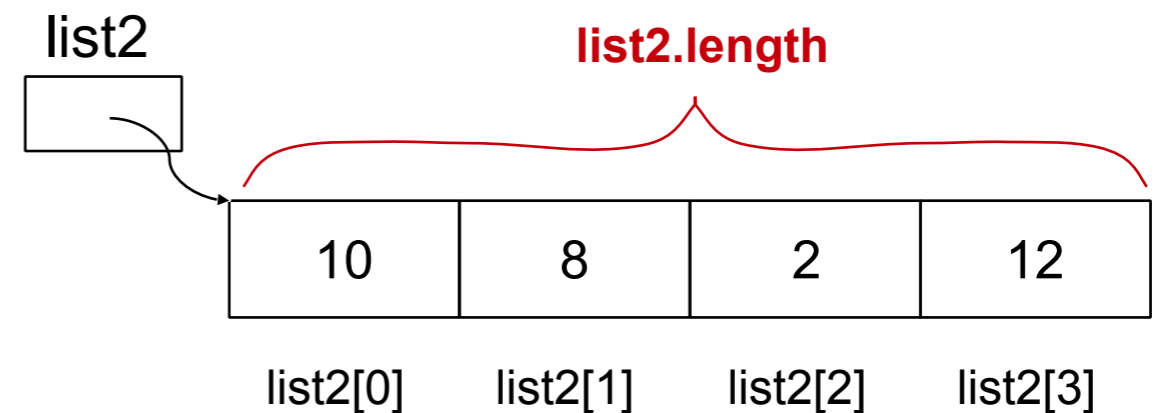
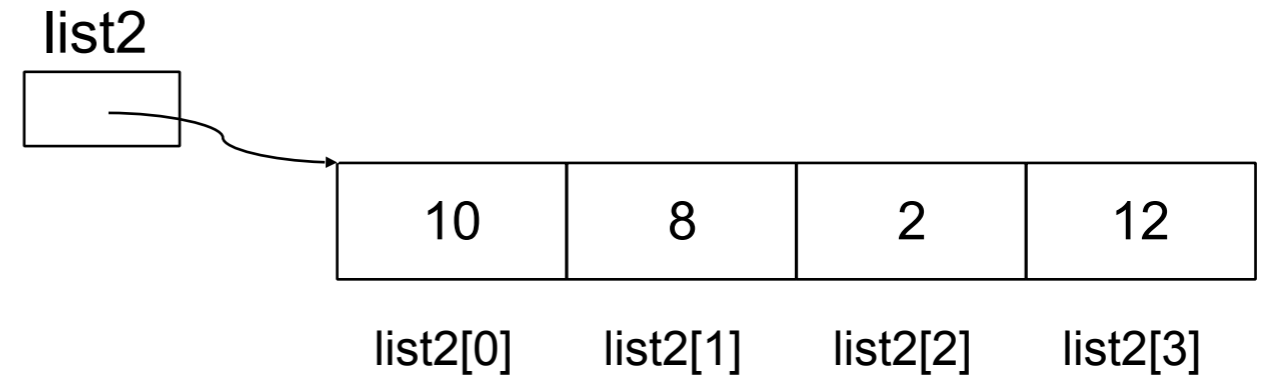


- Istället för att skapa ett fält med `new` kan fältet skapas genom att initiera värden till fältet vid deklarationen

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

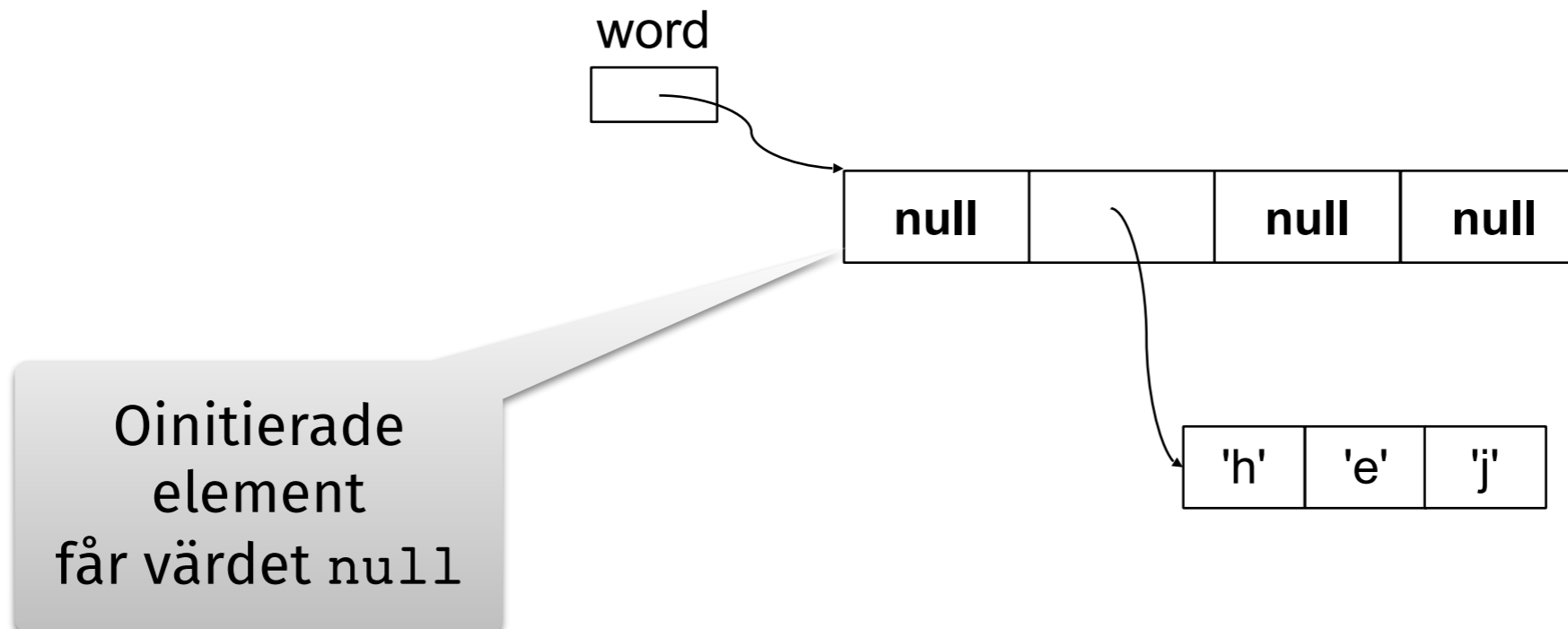
- Antalet värden som då deklareras bestämmer fältets storlek
- Längden av ett fält fås av instansvariabeln `length`

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



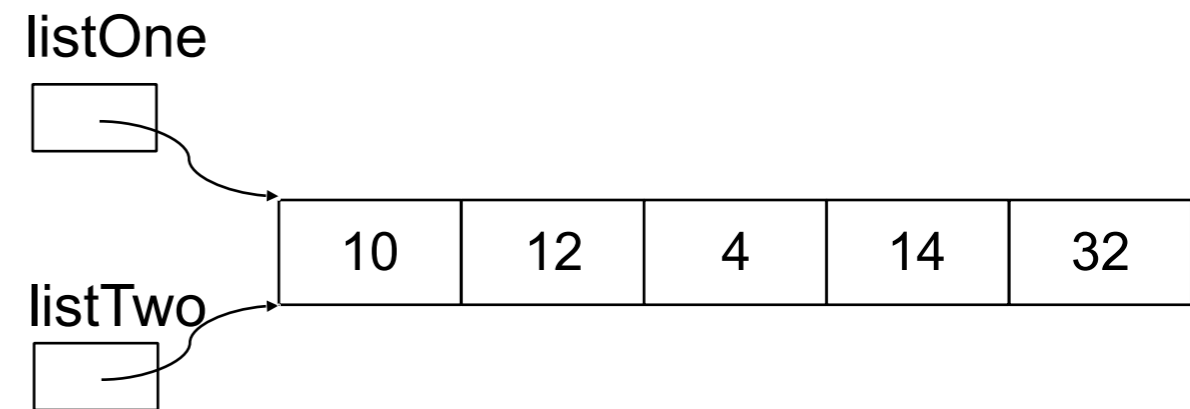
- Fält kan skapas av godtycklig typ

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



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

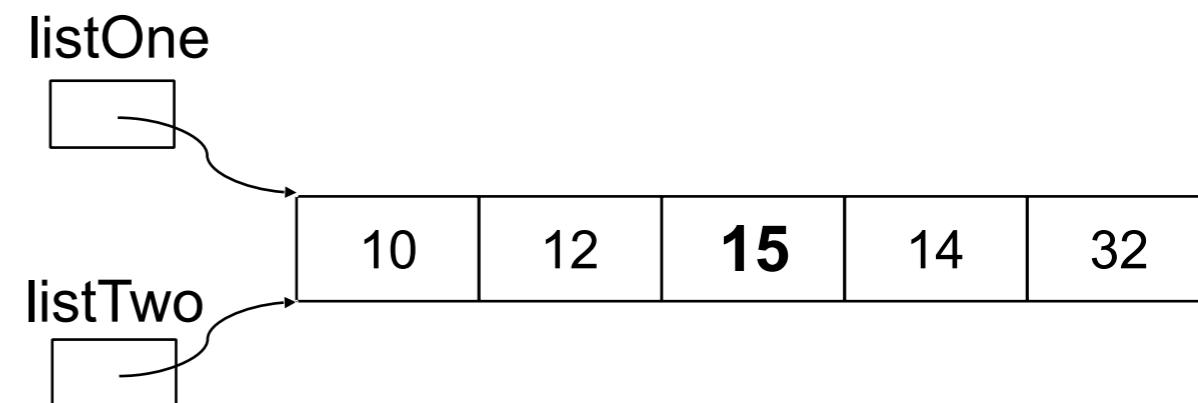
```
int[] listOne = {10, 12, 4, 14, 32};  
int[] listTwo = listOne;
```



- Satsen

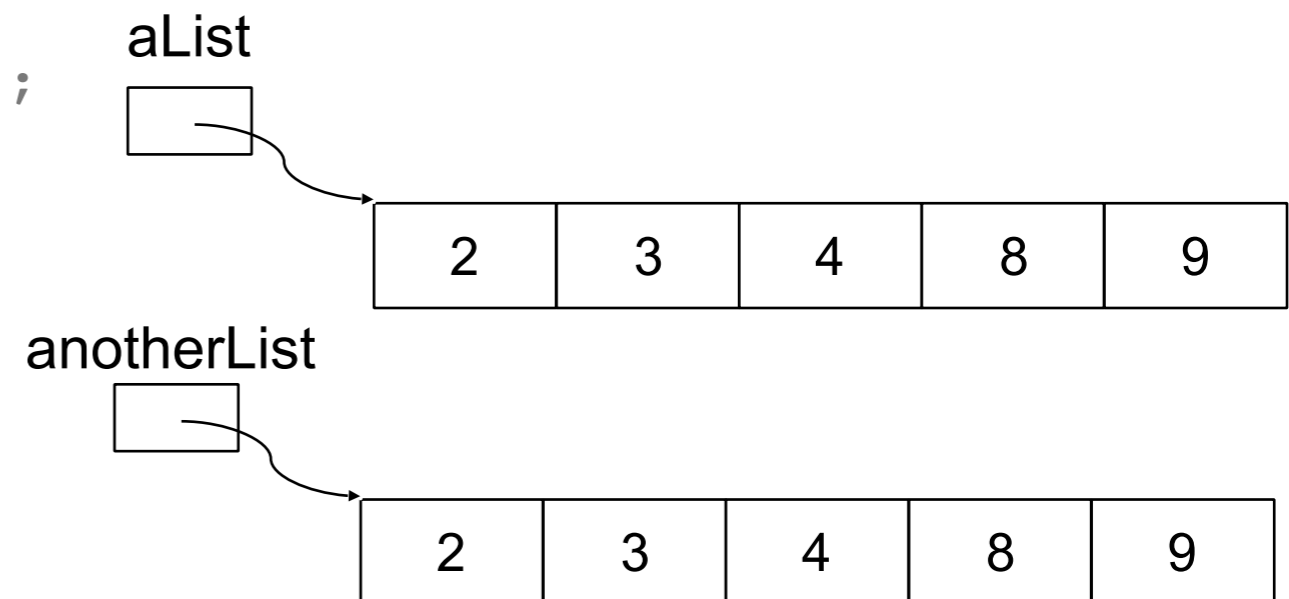
```
listTwo[2] = 15;
```

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



- 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!

Klassen `java.util.Arrays`

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

Metod	Beskrivning
<code>boolean equals(int[] a, int[] b)</code>	returnerar <code>true</code> om fälten <code>a</code> och <code>b</code> är lika långa och motsvarande komponenter är lika, annars returneras <code>false</code> .
<code>int[] copyOf(int[] f, int length)</code>	returnerar en kopia av fältet <code>f</code> med längden <code>length</code> , om kopian är kortare än <code>f</code> sker en trunkering, om kopian är längre fylls kopian ut med 0:or
<code>int[] copyOfRange(int[] f, int from, int to)</code>	returnerar ett fält som innehåller elementen från index <code>from</code> till index <code>to</code> i fältet <code>f</code>
<code>String toString(int[] f)</code>	returnerar en textrepresentation av fältet <code>f</code> , på formen <code>[e₁, e₂, ..., e_n]</code>
<code>void fill(int[] f, int value)</code>	sätter alla element i fältet <code>f</code> till värdet <code>value</code>
<code>void fill(int[] f, int from, int to, int value)</code>	sätter alla elementen från index <code>from</code> till index <code>to</code> i fältet <code>f</code> till värdet <code>value</code>

Klassen `java.util.Arrays`

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

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

- 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 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`

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 ändå 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

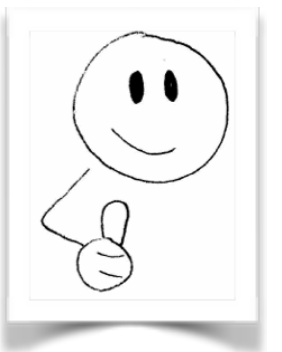
Att söka i ett osorterat fält

- 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?*)

```
// 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!

```
// 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;
}
```

- De metoder vi implementerat ovan är handhar situationer som är vanligt återkommande delproblem i många skilda sammanhang
- Det är därför mycket lämpligt att 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) {...}  
    ...  
}
```

- 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 dubletter 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}`

- **Förvillkor:** `list != null`
- **Diskussion:** vi börjar med att skapa ett nytt fält, som vi kan kalla `newList`. Sedan tar vi ett element i taget från fältet `list` och lägger i detta element i `newList` om elementet inte redan finns i `newList`
- **Algoritm:**
 1. `nrElems = 0;`
 2. så länge det finns fler element kvar i `list`
 - 2.1. `if (nästa element i list inte finns i newList)`
 - 2.1.1. `newList[nrElems] = nästa element i list;`
 - 2.1.2. `nrElems = nrElems + 1;`
 3. Returnera den ett fält som innehåller elementen som lagrats i `newList`
- **Datarepresentation:**
 - `newList` är av datatypen `int[]`
 - `nrElems` är av datatypen `int`

Implementation

```
import java.util.*;

public class NoDuplicate {
    // before: list != null
    private static int[] removeAllDuplicates(int[] list) {
        int[] newList = new int[list.length];
        int nrElems = 0;
        for (int i = 0; i < list.length; i = i + 1) {
            int next = list[i];
            if (!ArrayUtils.isInList(Arrays.copyOf(newList, nrElems), next)) {
                newList[nrElems] = next;
                nrElems = nrElems + 1;
            }
        }
        return Arrays.copyOf(newList, nrElems);
    }

    public static void main(String[] args) {
        int[] vekt = {1, 4, 1, 2, 4, 5, 12, 3, 2, 4, 1};
        int[] result = removeAllDuplicates(vekt);
        System.out.println(Arrays.toString(result));
    }
}
```

Alternative implementation

```
import java.util.*;

public class NoDuplicate {
    private static int[] removeAllDuplicates(int[] xs) {
        int[] ys = new int[0]; // Start with empty list
        for (int x : xs) // For all elements
            if (!ArrayUtils.isInList(ys, x)) // If not in list
                ys = add(x, ys); // Add to new list
        return ys;
    }

    private static int[] add(int x, int[] xs) {
        int[] ys = Arrays.copyOf(xs, xs.length + 1); // Make one larger
        ys[xs.length] = x; // Add x at end of list
        return ys;
    }

    public static void main(String[] args) {
        int[] vekt = {1, 4, 1, 2, 4, 5, 12, 3, 2, 4, 1};
        int[] result = removeAllDuplicates(vekt);
        System.out.println(Arrays.toString(result));
    }
}
```


- 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:

```
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

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

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 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];
}
```

- 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

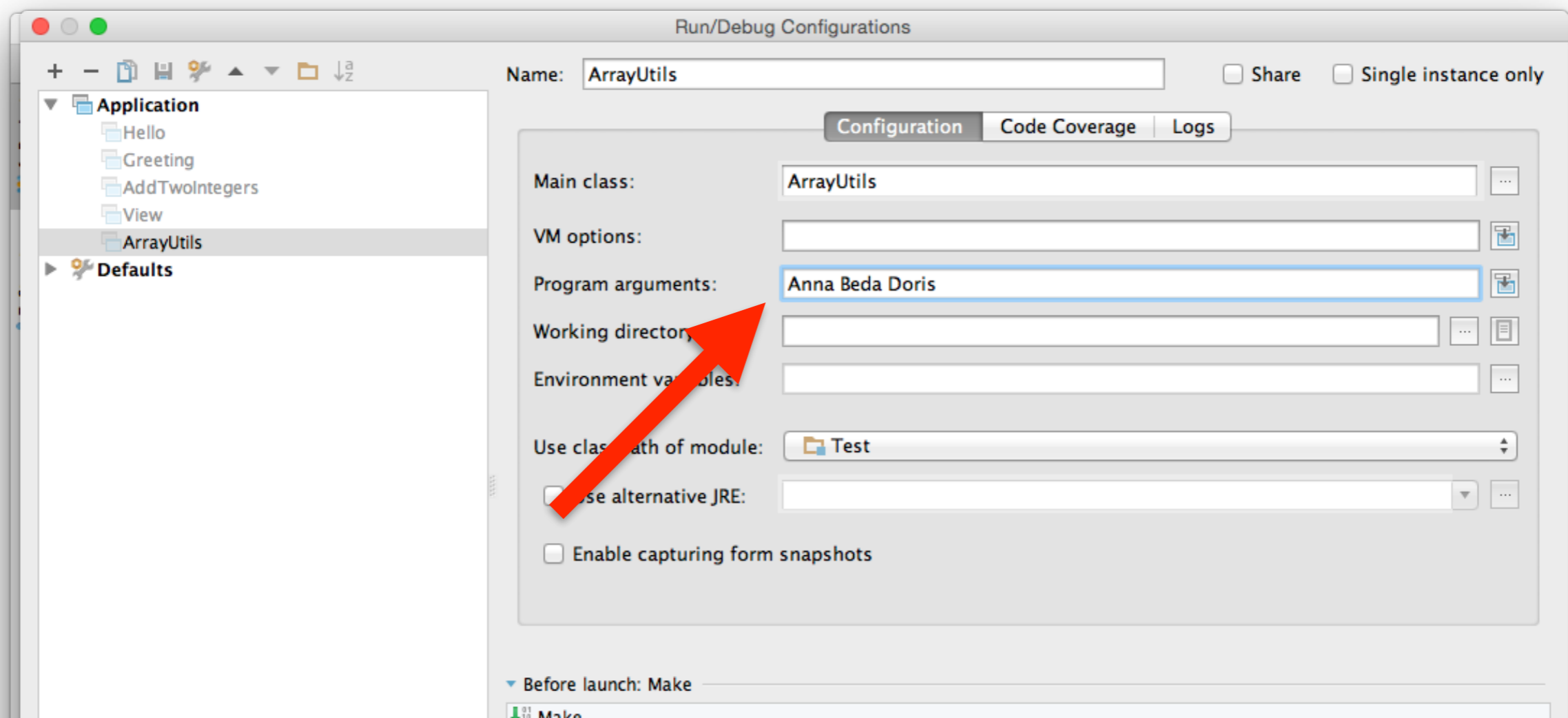
```
Argument 0 = Anna  
Argument 1 = Beda  
Argument 2 = Doris
```

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

Metoder i standardklassen `String`

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

Fler metoder finns i klassen `String`

Metoder i standardklassen Character

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

Standardklassen `String`

- I Java används en internationell standard för att lagra tecken som kallas *Unicode*
- De svenska tecknen ligger inte i följd i denna standard, varför inte svenska tecken och svenska ord kan jämföras enligt alfabetisk ordning med metoden `compareTo` i klassen `String`, istället måste en särskild *jämförare* användas
- En jämförare är ett objekt av klassen `Collator`, som finns i paketet `java.text`
- En jämförare deklarerar på följande sätt:

```
Collator co = Collator.getInstance(new Locale("sv", "SE"));
```

där

```
new Locale("sv", "SE")
```

definierar de språkkonventioner som skall användas

- Klassen `Locale` finns i paketet `java.util`

- Texter kan jämföras med metoderna `compare` eller `equals`:

```
co.compare(s1, s2)
```

ger värdet 0 om `s1` och `s2` är alfabetiskt lika, ett värde < 0 om `s1` kommer för `s2` och ett värde > 0 om `s2` kommer för `s1`

```
co.equals(s1, s2)
```

ger `true` om `s1` och `s2` är alfabetiskt lika, annars fås värdet `false`

- Det är möjligt att sätta "nivån" på jämföraren:

```
co.setStrength(Collator.PRIMARY)
```

```
co.setStrength(Collator.SECONDARY)
```

```
co.setStrength(Collator.TERTIARY)
```

skiljer endast på olika bokstäver

skiljer på utsmyckade bokstäver

skiljer på utsmyckade bokstäver,

samt på små och stora bokstäver

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 "1xfg2ÄåÖabcdeHl,% 3" ger utskriften:

Antalet vokaler är 6

Antalet konsonanter är 7

Antalet siffror är 3

- **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`

- 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`

- **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 = "aeiouyåäö";
```

- 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

```
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;
}
}
```

```
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;
    }
    ...
}
```