Föreläsning 7

Fält
Klassen String

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*.

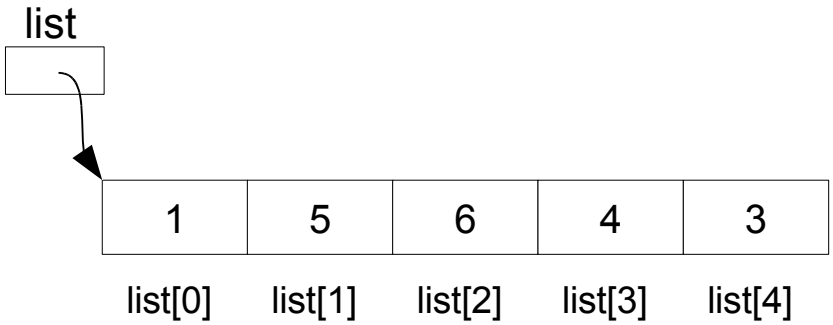
```
int[] list = new int[5];
```

Indexering sker *alltid* från 0.

Oinitierade heltal får värdet 0.

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

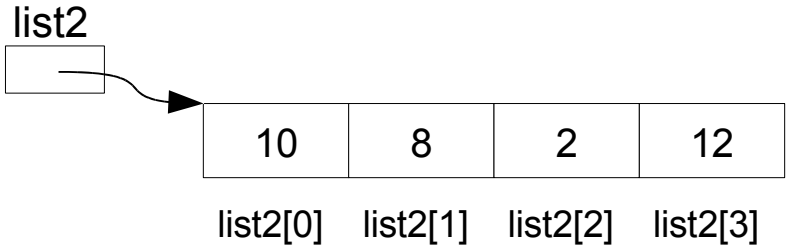
```
list[0] = 1;
list[1] = 5;
list[2] = 6;
list[3] = 4;
list[4] = 3;
```



Fält

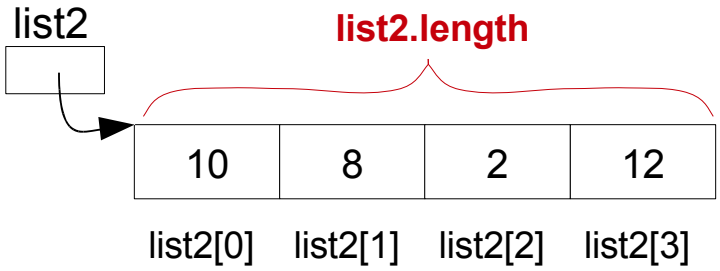
Istället för att skapa ett fält med **new** kan fältet skapas genom att initiera värden till fältet vid deklarationen. Antalet värden som då deklareras bestämmer fältets storlek.

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



Längden av ett fält fås av instansvariabeln *length*

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



Fält

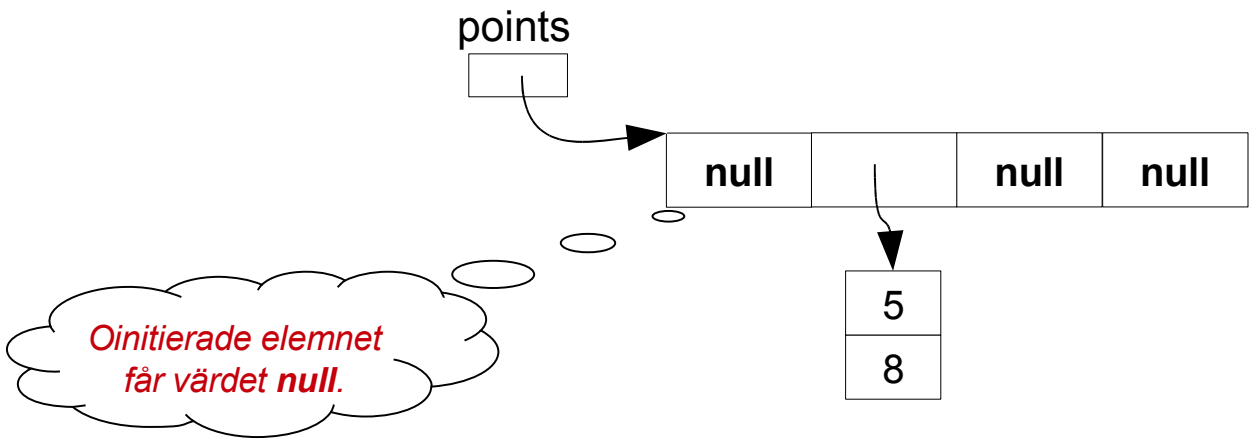
Fält kan skapas av godtycklig typ.

```
double [] numbers = new double[20];
String[] names = {"Adam", "Beda", "Cesar", "David", "Erik", "Fabian"};
Point[] points= new Point[4];
```

Klassen `Punkt` är definierad enligt:

```
public class Point {
   private int x, y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   //fler metoder
}//Point
```

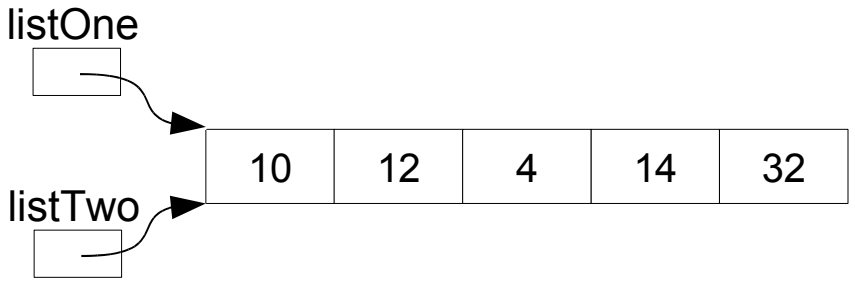
```
Point[] points = new Point[4];
points[1] = new Point(5, 8);
```



Fält

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

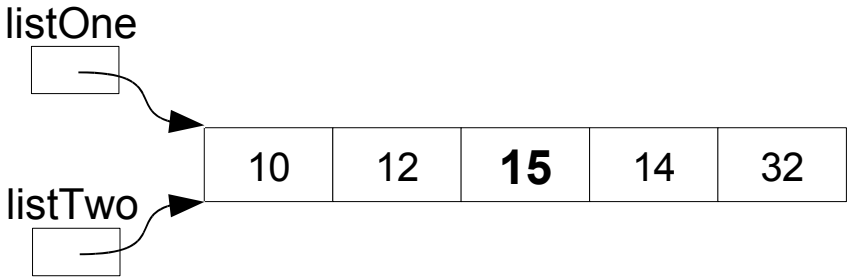
```
int[] listOne = {10, 12, 4, 14, 32};
int[] listTwo = listOne;
```



Satsen

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

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



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 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.
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.ex. **double**[], **boolean**[] och **char**[]!

Användning av metoder i java.util.Arrays

Exempel: 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]

Exempel: 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.

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
}
```

Exempel: Summera talen i heltalsfältet list

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

Att söka i ett *osorterat* fält.

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

```
//pre: 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;
}//isInList
```

Innan vi börjar sökningen har vi **inte** funnit vad vi söker

Nu har vi funnit det vi söker

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

```
//pre: 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;
}//isInList
```

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

Att söka i ett *osorterat* fält.

Alternativa implementationer

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

```
//pre: 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;
}//isInList
```

Att söka i ett *osorterat* fält.

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

```
//pre: 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)
        return index;
    else
        return - 1;
}//firstIndexOf
```

```
//pre: 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;
}//firstLastOf
```

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) {. . .}

   . . .
}//ArrayUtils
```

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:

Diskussion: Vi börjar med att skapar 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. *nrOfStoredElements* = 0;
2. så länge det finns fler element kvar i *list*
   2.1. **if** (nästa elenent i *list* inte finns i *newList*)
      2.1.1. *newList[nrOfStoredElements] = nästa element i list;*
      2.1.2. *nrOfStoredElements* = *nrOfStoredElements* + 1;
3. Returnera den ett fält som innehåller elementen som lagrats i *newList*

Datarepresentation:

newList är av datatypen **int**[].

nrOfStoredElements är av datatypen **int**.

Implementation:

```
import java.util.*;
public class NoDublicate {
   . . .
   //pre: list != null
   private static int[] removeAllDuplicates(int[] list) {
      int[] newList = new int[list.length];
      int nrOfStoreElements = 0;
      for (int i = 0; i < list.length; i = i + 1) {
         int nextElement = list[i];
         if (!ArrayUtils.isInList(Arrays.copyOf(newList, nrOfStoreElements), nextElement)) {
            newList[nrOfStoreElements] = nextElement;
            nrOfStoreElements = nrOfStoreElements + 1;
         }
      }
      return Arrays.copyOf(newList, nrOfStoreElements);
   }//removeAllDuplicates
   . . .
}//NoDublicate
```

Genomsök endast den del av fältet som innehåller element

Returnera endast den del av fältet som innehåller element

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:

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

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

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.

Obs: I just detta exempel hade det givetvis 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 användas som uppslagstabeller

```
//pre: 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!";
}//getWeekday
```

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

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

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

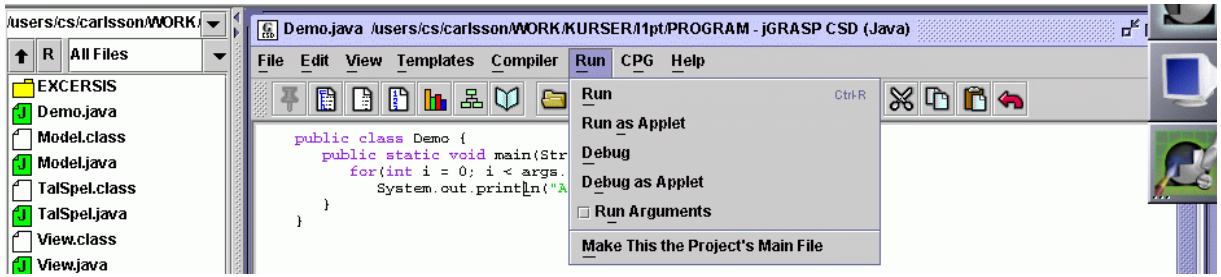
```
Argument 0 = Anna
Argument 1 = Beda
Argument 2 = Doris
```

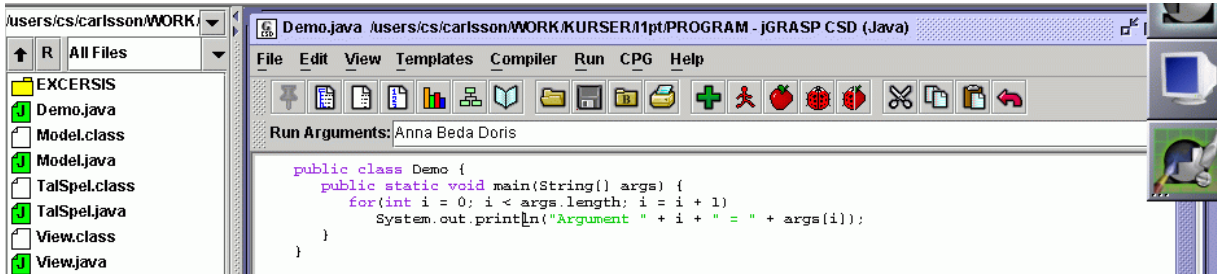
Parametrar till main.

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 jGrasp skapar man ett kommandorad i vilket argumenten ges, genom att trycka på "Run Arguments" i menyn "Run".



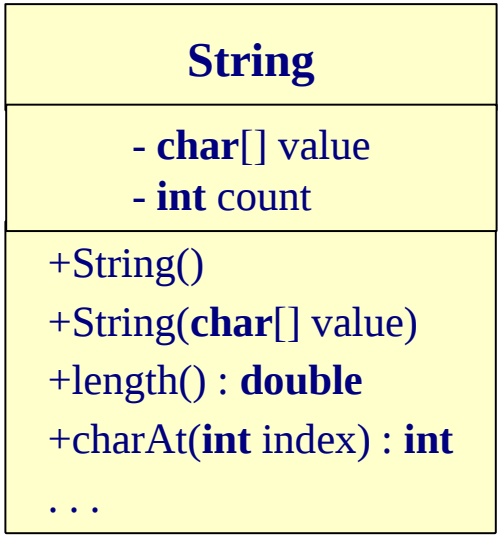


Standardklassen String

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.



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 lastndexOf(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

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

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 deklareras 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.

Standardklassen String

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)	Skiljer endast på olika bokstäver.
co.setStrength(Collator.SECONDARY)	Skiljer på utsmyckade bokstäver.
co.setStrength(Collator.TERTIARY)	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

"1xfg2ÄåÖabcdeHI,%3"

ger utskriften:

Antalet vokaler är 6
Antalet konstanter 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 getNrOfConsonants(String str)	returnerar antalet konsonanter i strängen str
int getNrOfVowels(String str)	returnerar antalet vokaler i strängen str

Algoritm för huvudprogrammet:

1. Läs strängen *indata*
2. Sätt *antSiffror* = getNrOfDigits(indata);
3. Sätt *antVokaler* = getNr och *antKonsonanter* = 0
4. Skriv ut *antVokaler, antKonsonanter och antSiffror*

Datarepresentation:

indata är av datatypen String

antSiffror, *antVokaler*, *antKonsonanter* är av typen **int.**

Diskussion: fortsättning

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*

Diskussion: fortsättning

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

1. Sätt *number* = 0
2. För varje tecken *ch* i *str*
   2.1. **if** (Character.isLetter() && !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) {
      while(true) {
         String indata = JOptionPane.showInputDialog("Ge en mening: ");
         if (indata == null)
            break;
         int nrOfVowels = getNrOfVowels(indata);
         int nrOfConsonants = getNrOfConsonants(indata);
         int nrOfDigits = getNrOfDigits(indata);
         JOptionPane.showMessageDialog(null, "Antalet vokaler är " + nrOfVowels
                                                + "\nAntalet konsonanter är " + nrOfConsonants
                                                + "\nAntalet siffror är " + nrOfDigits);
      }//while
   }//main

   private static boolean isVowel(char ch) {
      String vocals = "aeiouyåäö";
      char smallCh = Character.toLowerCase(ch);
      if (vocals.indexOf(smallCh) != -1)
         return true;
      else
         return false;
   }//isVowel
```

Implementation: fortsättning

```
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;
}//getNrOfVowels

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;
}//getNrOfDigits
```

```
    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;
    }//getNrOfConsonants
}//Count
```