Klasse

versus

Interface

Wat is een interface ?

- Een type
- Enkel ‘declaratie’ van een of meerdere methoden
- Geen implementatie
  - geen ‘{…}’,
  - enkel signatuur, gevolgd door ‘;’ in de interface definitie!

```
public interface InterfaceX  {
            public void methodeA() ;
            public String methodeB(int a, char b) ;
}
```

```
public interface InterfaceY  {
            public double methodeC() ;
}
```

- Een interface is een ‘contract’ dat door een klasse concreet kan geïmplementeerd worden

```
class KlasseXX implements InterfaceX {
        public void methodeA() {
                …
        }
        public String methodeB(int a, char b) {
                …
        }
}
```

- Een klasse kan meerdere interfaces implementeren

```
class KlasseXX implements InterfaceX, InterfaceY {
        public void methodeA() {  ... }
        public String methodeB(int a, char b) { ... }

        public double methodeC() { ... }
}
```

Interfaces in package java.lang

java.lang

Interfaces

Appendable
AutoCloseable
CharSequence
Cloneable
Comparable
Iterable
Readable
Runnable
Thread.UncaughtExceptionHandler

java.lang

Interface CharSequence

All Known Subinterfaces:

Name

All Known Implementing Classes:

CharBuffer, Segment, String, StringBuffer, StringBuilder

java.lang

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

All Known Subinterfaces:

Delayed, Name, Path, RunnableScheduledFuture<V>, ScheduledFuture<V>

All Known Implementing Classes:

AbstractRegionPainter.PaintContext.CacheMode, AccessMode, AclEntryFlag, AclEntryP
Authenticator.RequestorType, BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calen
Character.UnicodeScript, CharBuffer, Charset, ClientInfoStatus, CollationKey, Compon
CRLReason, CryptoPrimitive, Date, Desktop.Action, Diagnostic.Kind, Dialog.Mod
ElementKind, ElementType, Enum, File, FileTime, FileVisitOption, FileVisitResult, Float

String

De klasse String
en zijn interface implementaties

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

Klasse String en de invulling van zijn interface implementaties

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

Methods

Methods

Modifier and Type	Method and Description
char	charAt(int index) Returns the char value at the specified index.
int	length() Returns the length of this character sequence.
CharSequence	subSequence(int start, int end) Returns a new CharSequence that is a subsequence of this sequence.
String	toString() Returns a string containing the characters in this sequence in the same order as this sequence.

Methods

Modifier and Type	Method and Description
int	compareTo(T o) Compares this object with the specified object for order.

Klasse String implementeert de (0 + 4 + 1) methoden volgens ‘contract’

```
public class String implements Serializable, CharSequence, Comparable<String> {
    public char charAt(int index) {
            ...
    }
    public int length() {
            ...
    }
    public CharSequence subSequence(int beginIndex, int endIndex) {
            ...
    }
    public String toString() {
            ...
    }

    public int compareTo(String s) {
            ...
    }
}
```

Daarnaast een heel pak eigen methoden:

```
class String
    implements Serializable, CharSequence, Comparable<String> {
  …
  public boolean startsWith(String prefix)
  public boolean endsWith(String suffix)
  public String substring(int beginIndex)
  public String toUpperCase()
  public String toLowerCase()
  public String trim()
  public int indexOf(int ch)
  public int indexOf(int ch, int fromIndex)
  public String[] split(String regex, int limit)
  public String replace(CharSequence target, CharSequence replacement)
  public CharSequence subSequence(int beginIndex, int endIndex)
  public boolean equals(Object anObject)
  public String toString()
  …
}
```

Waar wordt overloading wordt toegepast ?
Waar wordt overriding wordt toegepast ?
Kan je een interface meegeven als parameter aan een methode ?
Kan je een interface teruggeven als resultaat van een methode ?
Kan een klasse meerdere interfaces implementeren ?

De Comparable interface

…

en zijn methode compareTo

…

(al dan niet generisch)

De interface `Comparable`

- Declareert slechts één methode, nl. compareTo
- Wordt door veel Java-klassen geïmplementeerd !

JAVA SE6

```
public interface Comparable {
    public int compareTo(Object o);
}
```

- Heeft een generische variant:

JAVA SE7

```
public interface Comparable<T> {
    public int compareTo(T t);
}
```

```
public interface Comparable {
    public int compareTo(Object o);
}
```

JAVA SE6

```
public interface Comparable<T> {
    public int compareTo(T t);
}
```

JAVA SE7

- ➔ Huidig object vergelijken met meegegeven ander object ***(van type Object, resp. T)***
- ➔ Rangorde bepalen: <neg>, 0, <pos>

- ➔ Vastleggen hoe objecten van een bepaald type gesorteerd moeten worden

Java API: `Comparable<T>`

compareTo

```
int compareTo(T o)
```

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y)>0 && y.compareTo(z)>0)` implies `x.compareTo(z)>0`.

Finally, the implementor must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero or positive.

Parameters:

`o` - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Throws:

`NullPointerException` - if the specified object is null

`ClassCastException` - if the specified object's type prevents it from being compared to this object.

Implementatie in klasse String

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

s1	s2	s1.compareTo(s2)
K**r**istien	K**a**tja	17
Krist**i**en	Krist**a**	8
Krist**i**en	Krist**o**f	-6
Krist**ien**	Krist	3
Kristien	Kristien	0

Specified by:

`compareTo` in interface `Comparable<String>`

Parameters:

`anotherString` - the `String` to be compared.

Returns:

the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.

compareToIgnoreCase

```
public int compareToIgnoreCase(String str)
```

Compares two strings lexicographically, ignoring case differences. This method returns an integer whose sign is that of calling `compareTo` with normalized versions of the strings

Sorteren van objecten in een verzameling

- a.d.h.v. een statische methode sort()

- Voor arrays: Arrays.sort(rij);
- Voor arraylists: Collections.sort(al);

Arrays.sort(*<verzameling>*); De onderliggende werking

```
String[] namen = { "Steven", "Bert", "Davy", "Charlotte"};
Arrays.sort(namen);
```

```
Punt[] coordinaten = {new Punt(0,3), new Punt(2, 4), new Punt(1, 4), new Punt(1,3) };
Arrays.sort(coordinaten);
```

- Je roept *zélf* nergens de methode `compareTo` op
- De methode `Arrays.sort` zal *intern* de `compareTo` methode oproepen voor elk koppel in de rij en telkens [via teruggave van <neg>, 0 of <pos>] bepalen welk object van beide ‘eerst’ komt in de rangorde. Op die manier wordt de ganse rij gesorteerd

RIJEN VAN OBJECT TYPE:

```
String[] namen = { "Kristien", "Katja", "Krista", "Kristoff", "Krist", "Kristien", "Bert"};
```

```
import java.util.*;
…
drukRij(namen);
Arrays.sort(namen);
drukRij(namen);
```

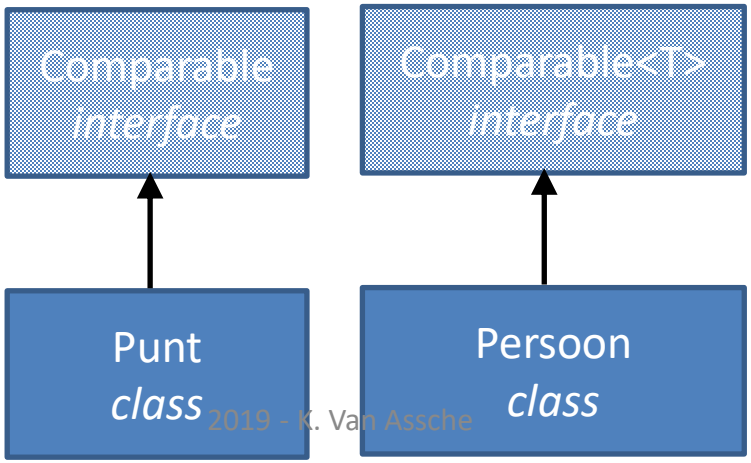
Kristien Katja Krista Kristoff Krist Kristien Bert

Bert Katja Krist Krista Kristien Kristien Kristoff

Wanneer en hoe toepassen bij je eigen klassen?

- Wanneer objecten in een verzameling gesorteerd moeten kunnen worden: Implementeer de Comparable interface!

- Bepaal hoe (op welke basis) objecten in een verzameling gesorteerd moeten worden: Compilatiefout zolang 'contract' niet correct ingevuld is, t.t.z. zolang de methode `compareTo(Object o)`, resp. `compareTo(T t)` niet gedefinieerd is.



Implementatie van de Comparable interface

Volg steeds de signatuur van de compareTo methode van de interface !

```
class Punt implements Comparable {
    …
    @Override
    public int compareTo(Object o) {
        //Hoe wil je objecten van klasse Punt sorteren?
    }
}
```

```
class Persoon implements Comparable<Persoon> {
    …
    @Override
    public int compareTo(Persoon p) {
        // Hoe wil je personen onderling sorteren?
    }
}
```

```
Persoon[] personen = {        new Persoon("Filip", "prins", 1960),
                              new Persoon("Mathilde", "prinses", 1973),
                              new Persoon("Elisabeth ", "prinses", 2001),
                              new Persoon("Gabriël", "prins", 2003),
                              new Persoon("Emmanuel", "prins", 2005),
                              new Persoon("Eléonore", "prinses", 2008    };
drukRij(personen);

java.util.Arrays.sort(personen); // sorteer oplopend op geboortejaar
drukRij(personen);
```

```
class Persoon implements Comparable<Persoon> {
      …
      @Override
      public int compareTo(Persoon p) {
          return this.geboortejaar - p.geboortejaar;
      }
}
```

```
Persoon[] personen = new Persoon[] {
                                    new Persoon("Filip", "prins", 1960),
                                    new Persoon("Mathilde", "prinses”, 1973),
                                    new Persoon("Elisabeth ", "prinses”, 2001),
                                    new Persoon("Gabriël", "prins”, 2003),
                                    new Persoon("Emmanuel", "prins”, 2005),
                                    new Persoon("Eléonore", "prinses”, 2008)     };
drukRij(personen);

java.util.Arrays.sort(personen); // sorteer alfabetisch op naam
drukRij(personen);
```

```
class Persoon implements Comparable<Persoon> {
      …
      @Override
      public int compareTo(Persoon p) {
          return this.naam.compareTo(p.naam);
      }
}
```


Debug-demo!

Wat als… je Arrays.sort oproept voor een rij van objecten waarbij de overeenkomstige klasse de Comparable interface niet implementeert…

```
Exception in thread "main" java.lang.ClassCastException:
    Persoon cannot be cast to java.lang.Comparable
    at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:290)
    at java.util.ComparableTimSort.sort(ComparableTimSort.java:157)
    at java.util.ComparableTimSort.sort(ComparableTimSort.java:146)
    at java.util.Arrays.sort(Arrays.java:472)
    at Console.sorteerRij(SorteerPersoonArrayProgram.java:42)
    at Console.main(SorteerPersoonArrayProgram.java:21)
```

Merk op:
Bij het vergelijken worden de elementen **geüpcast** naar de interface Comparable, t.t.z. beschouwd als een object dat aan de Comparable interface definitie voldoet. Wanneer dat niet het geval blijkt te zijn, dan krijg je een **ClassCastException**.

ArrayList van objecten sorteren met Collections.sort

```
public class Console {
   public static void main(String[] args) {
      ArrayList lijst = new ArrayList();
      lijst.add(new Persoon("Annemieke", "Pieters”, 10));
      lijst.add(new Persoon("Rozemieke", "Pieters”, 10));
      lijst.add(new Persoon("Steven", "Stevens”, 10));

      Collections.sort(lijst);
   }
}
```

Wat als… je Collections.sort oproept voor een verzameling van objecten waarbij de overeenkomstige klasse de Comparable interface niet implementeert…

```
Exception in thread "main" java.lang.ClassCastException:
    Persoon cannot be cast to java.lang.Comparable
    at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:290)
    at java.util.ComparableTimSort.sort(ComparableTimSort.java:157)
    at java.util.ComparableTimSort.sort(ComparableTimSort.java:146)
    at java.util.Collections.sort(Collections.java:472)
    at Console.sorteerArrayList(SorteerPersoonArrayListProgram.java:42)
    at Console.main(SorteerPersoonArrayListProgram.java:21)
```

Merk op:
Bij het vergelijken worden de elementen **geüpcast** naar de interface Comparable, t.t.z. beschouwd als een object dat aan de Comparable interface definitie voldoet. Wanneer dat niet het geval blijkt te zijn, dan krijg je een **ClassCastException**.

Collections.sort(*<verzameling>*); De onderliggende werking

```
ArrayList<Punt> coordinaten = new ArrayList<>();
coordinaten.add(new Punt(0,3));
coordinaten.add(new Punt(2, 4));
coordinaten.add(new Punt(1, 4));
coordinaten.add(new Punt(1,3));

Collections.sort(coordinaten);
```

- Je roept *zélf* nergens de methode `compareTo` op
- De methode `Collections.sort` zal *intern* de `compareTo` methode oproepen voor elk koppel in de rij en telkens [via teruggave van <neg>, 0 of <pos>] bepalen welk object van beide 'eerst' komt in de rangorde. Op die manier wordt de ganse arraylist gesorteerd

Cf. vb. klasse Punt, generisch

```
public class Punt implements Comparable<Punt> {
  …

   @Override
    public int compareTo(Punt p) {
            if ( this.y != p.y) {
                    return p.y – this.y;
            }
            else {
                    return this.x – p.x;
            }
      }
}
```

Cf. vb. klasse Persoon, niet generisch

```
public class Persoon implements Comparable {
  …

  @Override
   public int compareTo(Object o) {
      if (o instanceof Persoon) {
            Persoon p = (Persoon)o;
            if (!this.naam.equals(p.naam)) {
                return this.naam.compareTo(p.naam);
            }
            return this.voornaam.compareTo(p.voornaam);
      }

      throw new IllegalArgumentException("ongeldig type");
   }
}
```

Sorteert personen
op naam,
vervolgens op voornaam

Voorbeeld met klasse String

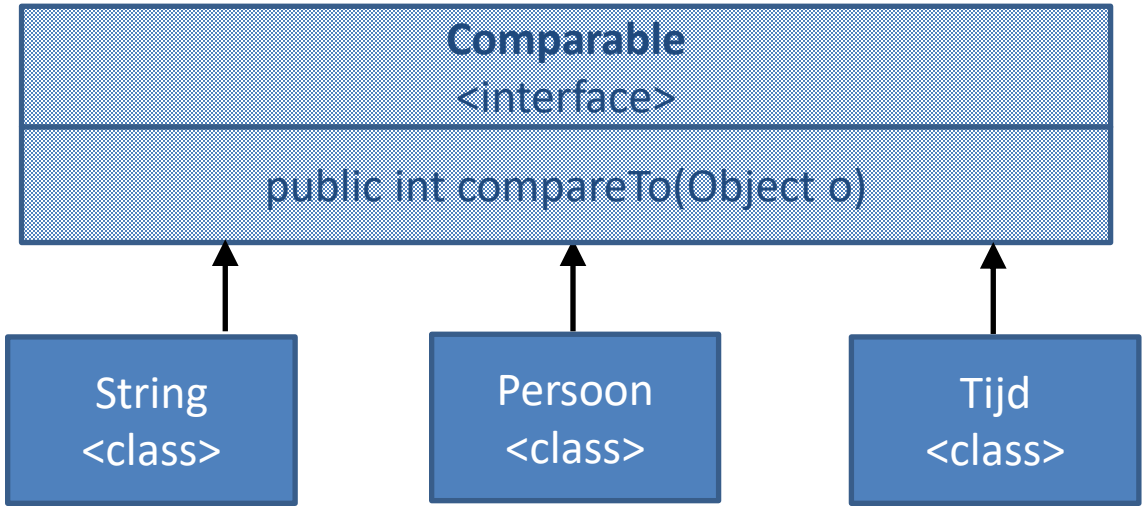
```
public class Console {
   public static void main(String[] args) {
      ArrayList<String> lijst = new ArrayList<>();
      lijst.add(new String("Aap”));
      lijst.add(new String("Zebra"));
      lijst.add(new String("Cavia"));

      drukInfo(lijst);
      Collections.sort(lijst);
      drukInfo(lijst);
   }

   public static void drukInfo(ArrayList<String> lijst) {
      for (String s : lijst) {
         System.out.println(s);
      }
   }
}
```

compareTo implementatie is voorzien in Java API zelf

```
Aap
Zebra
Cavia
```

```
Aap
Cavia
Zebra
```

Besluit: Sorteren van objecten



Objecten in Array => Arrays.sort(rij);
Objecten in ArrayList => Collections.sort(rij);

Polymorfe werking – niet generisch

```
public class Console {
   public static void main(String[] args) {
      ArrayList lijst = new ArrayList();
      lijst.add(new Ouder("Jo", "Smith", 1966));
      lijst.add(new Ouder("An", "Peters", 1968));
      lijst.add(new Kind("Marieke", "Smith", 2008));

      drukInfo(lijst);
      Collections.sort(lijst); //bv. op voornaam
      drukInfo(lijst);
   }

   public static void drukInfo(ArrayList lijst) {
      for (Object o : lijst) {
         System.out.println(o);
      }
   }
}
```

Upcast van Ouder naar Object
Upcast van Ouder naar Object
Upcast van Kind naar Object

Jo Smith (1966)
An Peters (1968)
Marieke (10 jaar)

An Peters (1968)
Jo Smith (1966)
Marieke (10 jaar)

Late binding

Late binding
Downcast naar resp. Ouder/Kind
Oproep van de resp. toString() methode

Polymorfisme & Sorteren

```
public class Console {
    public static void main(String[] args) {
        ArrayList lijst = new ArrayList();
        lijst.add(new Punt(0,1));
        lijst.add(new String("Hallo"));
        lijst.add(new Persoon("Bob", 24));

        drukInfo(lijst);
        Collections.sort(lijst);
        drukInfo(lijst);
    }

    public static void drukInfo(ArrayList lijst) {
        for (Object o : lijst) {
            System.out.println(o);
        }
    }
}
```

Upcast van Punt naar Object
Upcast van String naar Object
Upcast van Persoon naar Object

```
(0,1)
Hallo
Bob (2
```

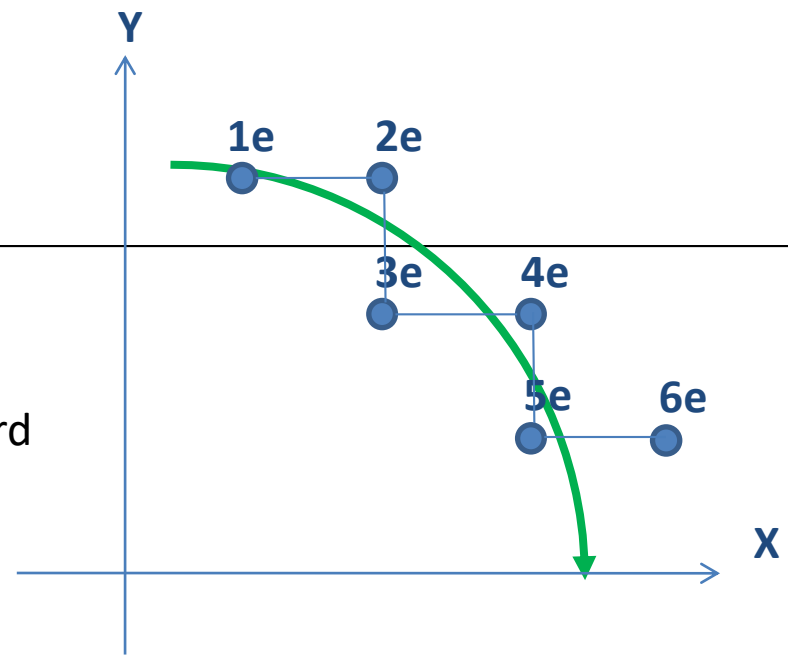
```
Exception in thread "main" java.lang.ClassCastException: logica.Punt  cannot be cast to java.lang.String
        at java.lang.String.compareTo(String.java:111)
        at java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:320)
        at java.util.ComparableTimSort.sort(ComparableTimSort.java:188)
        at java.util.Arrays.sort(Arrays.java:1312)
        at java.util.Arrays.sort(Arrays.java:1506)
        at java.util.ArrayList.sort(ArrayList.java:1454)
        at java.util.Collections.sort(Collections.java:141)
        at _2016_les9_ArrayListPolymorfisme.Console.main(Console.java:20)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Late binding
Downcast naar respectievelijk Punt, String en Persoon
Oproep van de resp. toString() methode

>>>>>>>> Vermijd Exceptions door generisch te werken

Oefening

```
import java.util.*;

Punt[] coordinaten = {new Punt(3,1), new Punt(4,1), new Punt(2, 2), new Punt(3, 2),
                      new Punt(1,3), new Punt(2,3) };

drukRij(coordinaten);
Arrays.sort(coordinaten); // sorteer
drukRij(coordinaten);
```



Gevraagd:
Pas de klasse Punt zó aan dat de punten gesorteerd kunnen worden zoals aangegeven op de figuur.
(De gewenste volgorde is dus 1e → 6e)

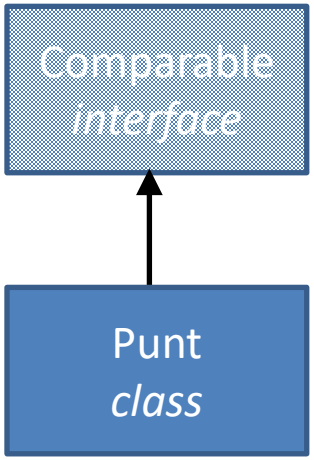
Vertaald:

- Kleinere x eerst
- Bij gelijke x, grotere y eerst

Of:

- Grotere y eerst
- Bij gelijke y, kleinere x eerst

Oplossing



Sorteerwijze:

- Grotere y eerst
- Bij gelijke y, kleinere x eerst

```
class Punt implements Comparable {
    @Override
    public int compareTo(Object o) {
        Punt p = (Punt)o;

        if ( this.y != p.y) {
                return p.y - this.y;
        }
        else {
                return this.x - p.x;
        }
    }
}
```

Bv. this.y = 5 en p.y = 12
⇒ 12 – 5 = 7
⇒ pos
⇒ this komt na p
⇒ i.e. grotere y eerst

Bv. this.x = 1 en p.x = 2
⇒ 1 – 2 = -1
⇒ neg
⇒ this komt vóór p
⇒ i.e. kleinere x eerst

Oefening

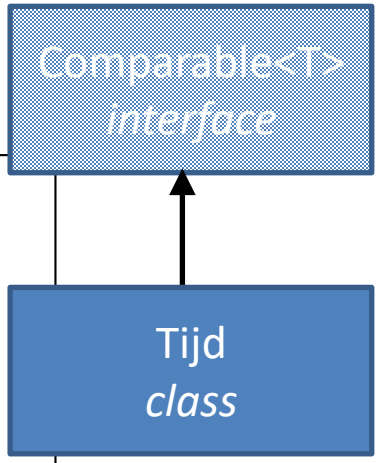
```
import java.util.*;

Tijd[] tijden = new Tijd[] {new Tijd(11,10), new Tijd(10,0), new Tijd(12, 45), new Tijd(12, 30 };

drukRij(tijden);
Arrays.sort(tijden); // sorteer
drukRij(tijden);
```

Gevraagd:
Druk de gegeven tijden in volgorde af.

Vertaald:

- Kleinere uren eerst
- Bij gelijke uren, kleinere minuten eerst

Oplossing




```
class Tijd implements Comparable<Tijd> {
    @Override
    public int compareTo(Tijd t) {
        if (this.uren > t.uren) {
            return 1;
        }
        else if (this.uren < t.uren) {
            return -1;
        }
        else if (this.minuten > t.minuten){
            return 1;
        }
        else if (this.minuten < t.minuten) {
            return -1;
        }
        return 0;
    }
}
```

Oplossing-bis

```
class Tijd implements Comparable<Tijd> {
    @Override
    public int compareTo(Tijd t) {
        if (this.uren != t.uren) {
            return this.uren - t.uren;
        }

        return this.minuten - t.minuten;
    }
}
```

Oplossing (compact)

```
class Tijd implements Comparable<Tijd> {
    @Override
    public int compareTo(Tijd t) {
        return (this.uren - t.uren) * 60
                + (this.minuten - t.minuten);
    }
}
```

Opmerking: Deze implementatie geeft het aantal minuten terug dat het huidig Tijd-object verschilt van het meegegeven ander Tijd-object t

Cf. eerder gegeven voorbeeld

Evalueer:

```
09u45
10u00
10u15
10u30
10u45
11u00
11u15
11u30
11u45
…
```

```java
public class Lestijd {
    public static void main(String[] args) {
        Tijd t1 = new Tijd(9, 45);
        Tijd t2 = new Tijd(11, 10);

        while (!t1.equals(t2)) {
            System.out.println(t1);
            t1.telTijdBij(new Tijd(0,15));
        }

        System.out.println(t1 + " - DE LES IS GEDAAN");
    }
}
```

En vergelijk met volgende:

```
09u45
10u00
10u15
10u30
10u45
11u00
11u15 – DE LES IS GEDAAN
```

```java
public class Lestijd {
    public static void main(String[] args) {
        Tijd t1 = new Tijd(9, 45);
        Tijd t2 = new Tijd(11, 10);

        while (t1.compareTo(t2) < 0) {
            System.out.println(t1);
            t1.telTijdBij(new Tijd(0,15));
        }

        System.out.println(t1 + " - DE LES IS GEDAAN");
    }
}
```

Voorbeeld: BigInteger – compareTo()

```
private static void testGroteGetallen() {
    int i = -1;
    BigInteger macht;

    while (true) {
        macht = BigInteger.valueOf(2).pow(++i);
        if (macht.compareTo(BigInteger.valueOf(Long.MAX_VALUE)) > 0) {
            System.err.println("!!! max long value bereikt");
            break;
        }
        else {
            System.out.println("2^" + i + " = " + macht);
        }
    }
}
```

```
public class BigInteger
extends Number
implements Comparable<BigInteger>
```

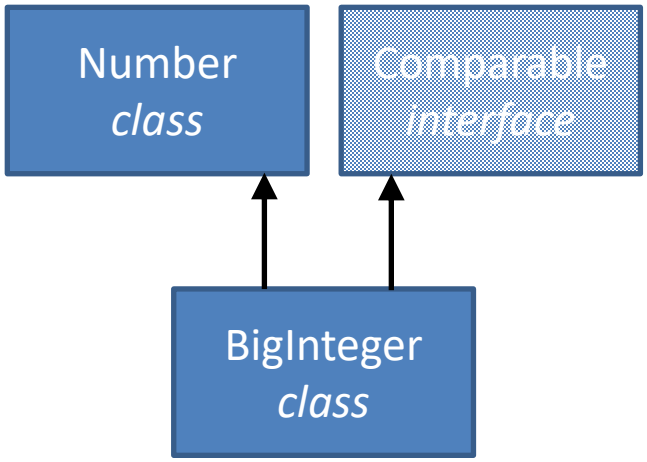
```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
2^10 = 1024
...
2^62 = 4611686018427387904
!!! max long value bereikt
```

Opm: Meest linkse bit is tekenbit!!!

```
01111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 //Long.MAX_VALUE
10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 //2^63
```

Dropbox\2016-2017 Java OO Programming\Clean Code Examples\PerfectNumberGenerator\PerfectNumberGenerator\src\perfectnumbergenerator

BigInteger - klassendiagramma



Voorbeeld: Student

```
public class Main {
   public static void main(String[] args) {
      Student[] studenten = new Student[] {
         new Student("Kristien"),
         new Student("Katja"),
         new Student("Peter")
      };
      drukStudenten(studenten);
      sorteerOpNaam(studenten);
      drukStudenten(studenten);
   }

   public static void sorteerOpNaam(Student[] studenten) {
      //bubblesort
      for (int i = 0; i < studenten.length; i++) {
         for (int j = i + 1; j < studenten.length; j++) {
            if (studenten[i].naam.compareTo(studenten[j].naam) > 0) {
               swap(studenten, i,j);
            }
         }
      }
   }
```

```
   public static void swap(Student[] a, int i, int j) {
      Student temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }

   public static void drukStudenten(Student[] studenten) {
      for (Student s : studenten) {
         System.out.println(s);
      }
   }
}
```

Het is onnodig om zelf een sorteeralgoritme te implementeren!!!

Betere versie

```
public class Main {
   public static void main(String[] args) {
      Student[] studenten = new Student[] {
         new Student("Kristien", 48),
         new Student("Katja", 42),
         new Student("Peter", 40)
      };

      druk(studenten);
      Arrays.sort(studenten);
      druk(studenten);
   }

   public static void druk(Student[] studenten) {
      for (Student s : studenten) {
         System.out.println(s);
      }
   }
}
```

```
public class Student
            implements Comparable<Student> {
   …
   @Override
   public String toString() {
      return studNr + ": " + naam
                        + " (" + leeftijd + ")";
   }

   @Override
   public int compareTo(Student s) {
      return this.naam.compareTo(s.naam);
   }
}
```

EXTRA

(informatief)

De Comparator interface

…

en zijn methode compare

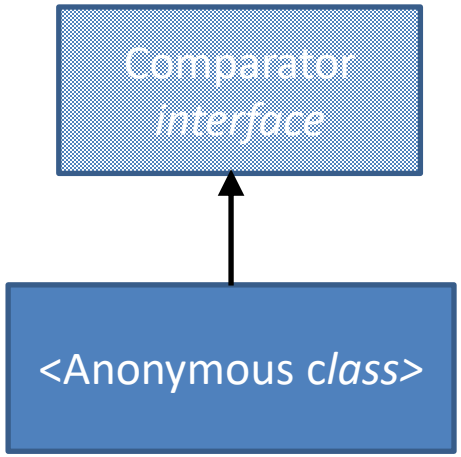
…

(al dan niet generisch)

Java8

Als je ook ‘anders’ wil kunnen sorteren

```
public class Main {
   public static void main(String[] args) {
      Student[] studenten = new Student[] {
         new Student("Kristien", 48), new Student("Katja", 42), new Student("Peter", 40) };

      Arrays.sort(studenten);
      druk(studenten); //volgens compareTo in klasse Student

      Comparator<Student> c = new Comparator<Student>() {
         @Override
         public int compare(Student s1, Student s2) {
            return s1.getLeeftijd() - s2.getLeeftijd();
         }
      };

      Arrays.sort(studenten, c);
      druk(studenten); //volgens de expliciet meegegeven comparator
   }
}
```



Korter met lambda expressie:

```
Comparator<Student> c = (Student s1, Student s2) -> s1.getLeeftijd() - s2.getLeeftijd();
Arrays.sort(studenten, c);
```

En nog korter:

```
Arrays.sort(studenten, (Student s1, Student s2) -> s1.getLeeftijd() - s2.getLeeftijd());
```

De Runnable interface

…

en zijn methode run

Interface Runnable

public interface **Runnable**

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, `Runnable` is implemented by class `Thread`. Being active simply means that a thread has been started and has not yet been stopped.

In addition, `Runnable` provides the means for a class to be active while not subclassing `Thread`. A class that implements `Runnable` can run without subclassing `Thread` by instantiating a `Thread` instance and passing itself in as the target. In most cases, the `Runnable` interface should be used if you are only planning to override the `run()` method and no other `Thread` methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

Since:

JDK1.0

See Also:

`Thread`, `Callable`

Method Summary

Methods

Modifier and Type	Method and Description
`void`	**`run()`** When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

Klasse Thread

Thread

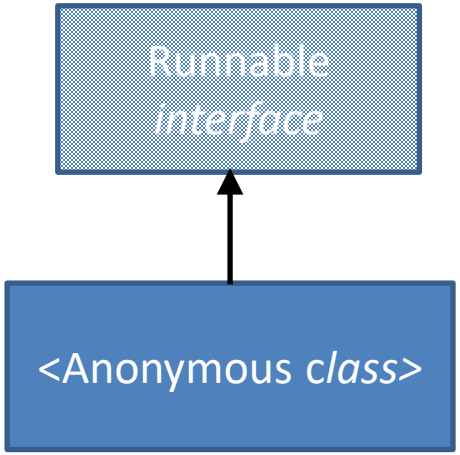
```
public Thread(Runnable target)
```

Allocates a new `Thread` object. This constructor has the same effect as `Thread(null, target, gname)`, where `gname` is a newly generated name. Automatically generated names are of the form "`Thread-`"+*n*, where *n* is an integer.

Parameters:

`target` - the object whose `run` method is invoked when this thread is started. If `null`, this classes `run` method does nothing.

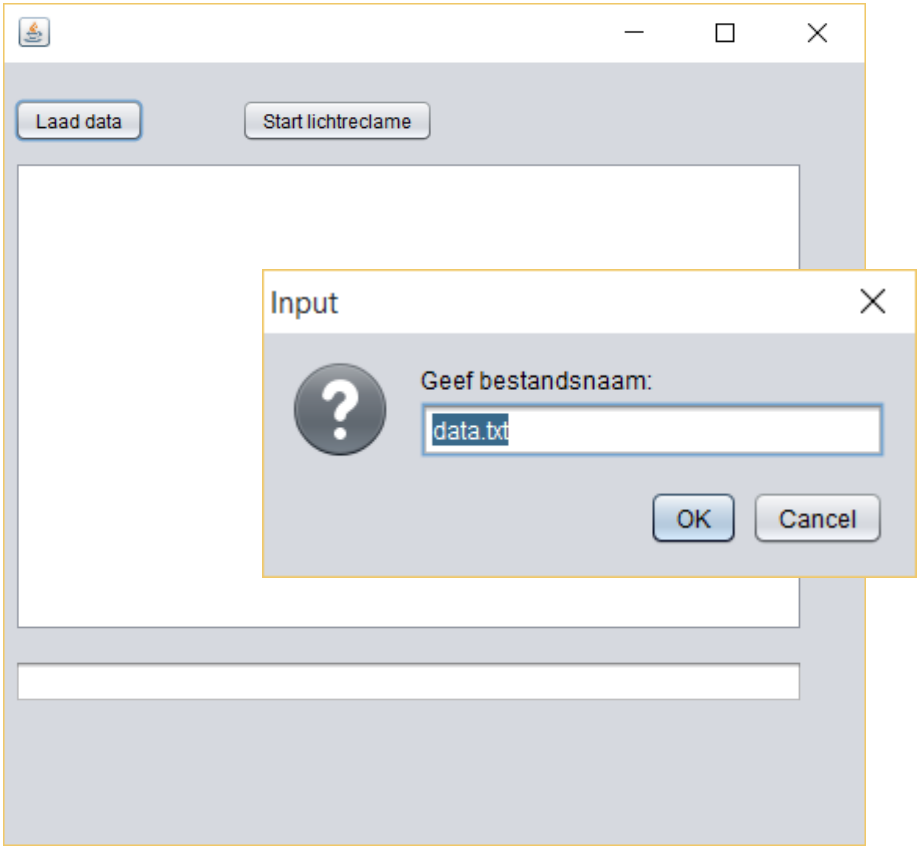
```
public class LichtkrantForm extends javax.swing.JFrame {
  private Thread t;

  private void jButtonStartLichtreclameActionPerformed(java.awt.event.ActionEvent evt) {
    if (t != null) {
      t.interrupt();
    }

    t = new Thread( new Runnable() {
      @Override
      public void run() {
          runLichtreclame();
      }
    } );

    t.start();
  }
}
```

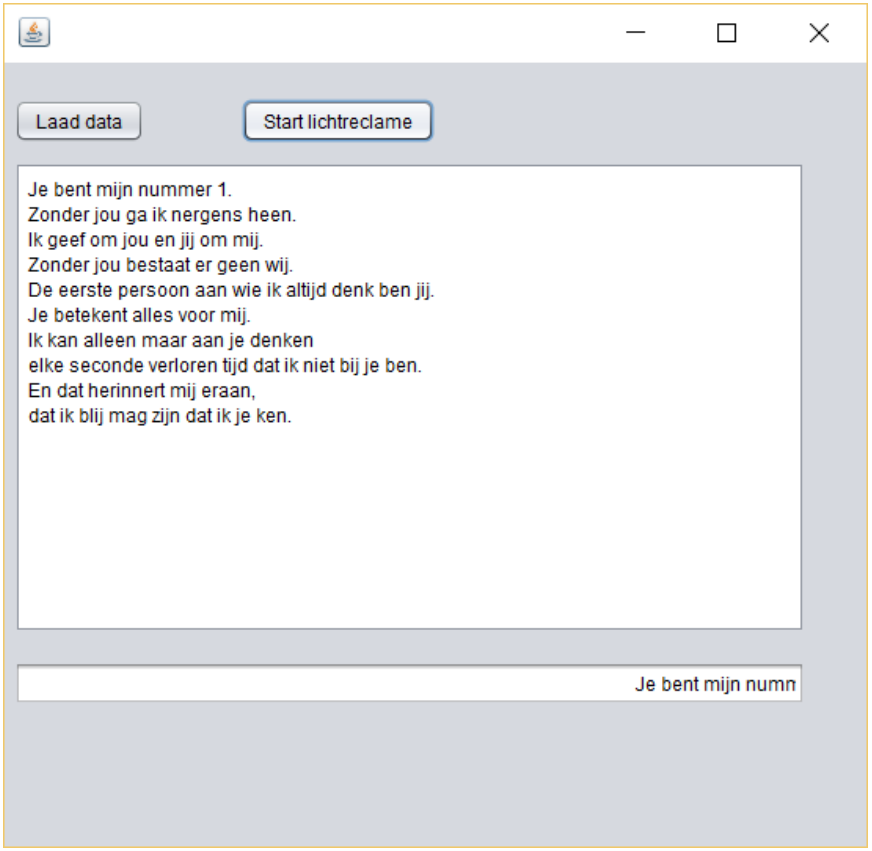


Korter met lambda expressie:

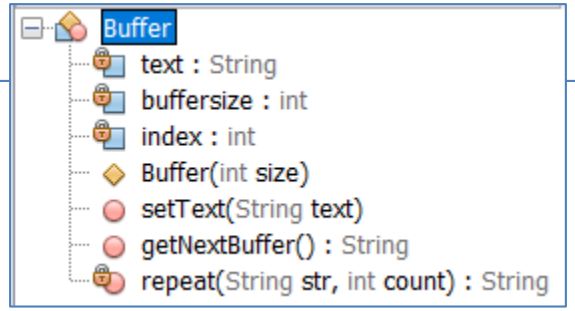
```
public class LichtkrantForm extends javax.swing.JFrame {
   private Thread t;

   private void jButtonStartLichtreclameActionPerformed(java.awt.event.ActionEvent evt) {
      if (t != null) {
         t.interrupt();
      }

      t = new Thread( () -> runLichtreclame() );

      t.start();
   }
}
```

Voorbeeld


Laad data
Start lichtreclame
Input
Geef bestandsnaam:
data.txt
OK
Cancel

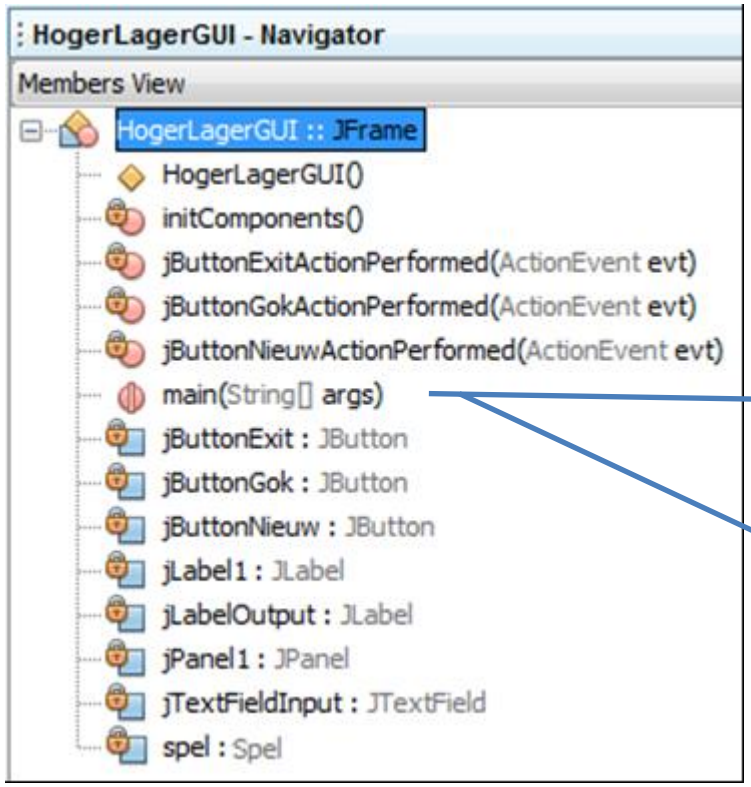

Laad data
Start lichtreclame
Je bent mijn nummer 1.
Zonder jou ga ik nergens heen.
Ik geef om jou en jij om mij.
Zonder jou bestaat er geen wij.
De eerste persoon aan wie ik altijd denk ben jij.
Je betekent alles voor mij.
Ik kan alleen maar aan je denken
elke seconde verloren tijd dat ik niet bij je ben.
En dat herinnert mij eraan,
dat ik blij mag zijn dat ik je ken.
Je bent mijn numm

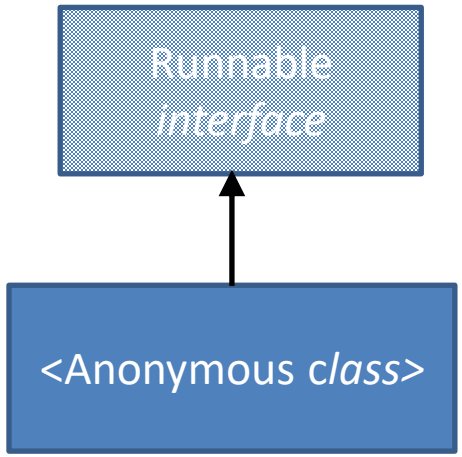
Gebruik eigen klasse Buffer




```
private void runLichtreclame() {
    Buffer b = new Buffer(150);
    b.setText(this.jTextArea1.getText());

    while(true) {
      this.jTextFieldRunningDisplay.setText(b.getNextBuffer());

      try {
        Thread.sleep(100);
      } catch (InterruptedException ex) {
        Logger.getLogger(LichtkrantForm.class.getName()).log(Level.SEVERE, null, ex);
      }
    }
  }
```

Opstarten GUI






```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            new HogerLagerGUI().setVisible(true);
        }
    });
}
```

This anonymous inner class creation can be turned into a lambda expression.

(Alt-Enter shows hints)

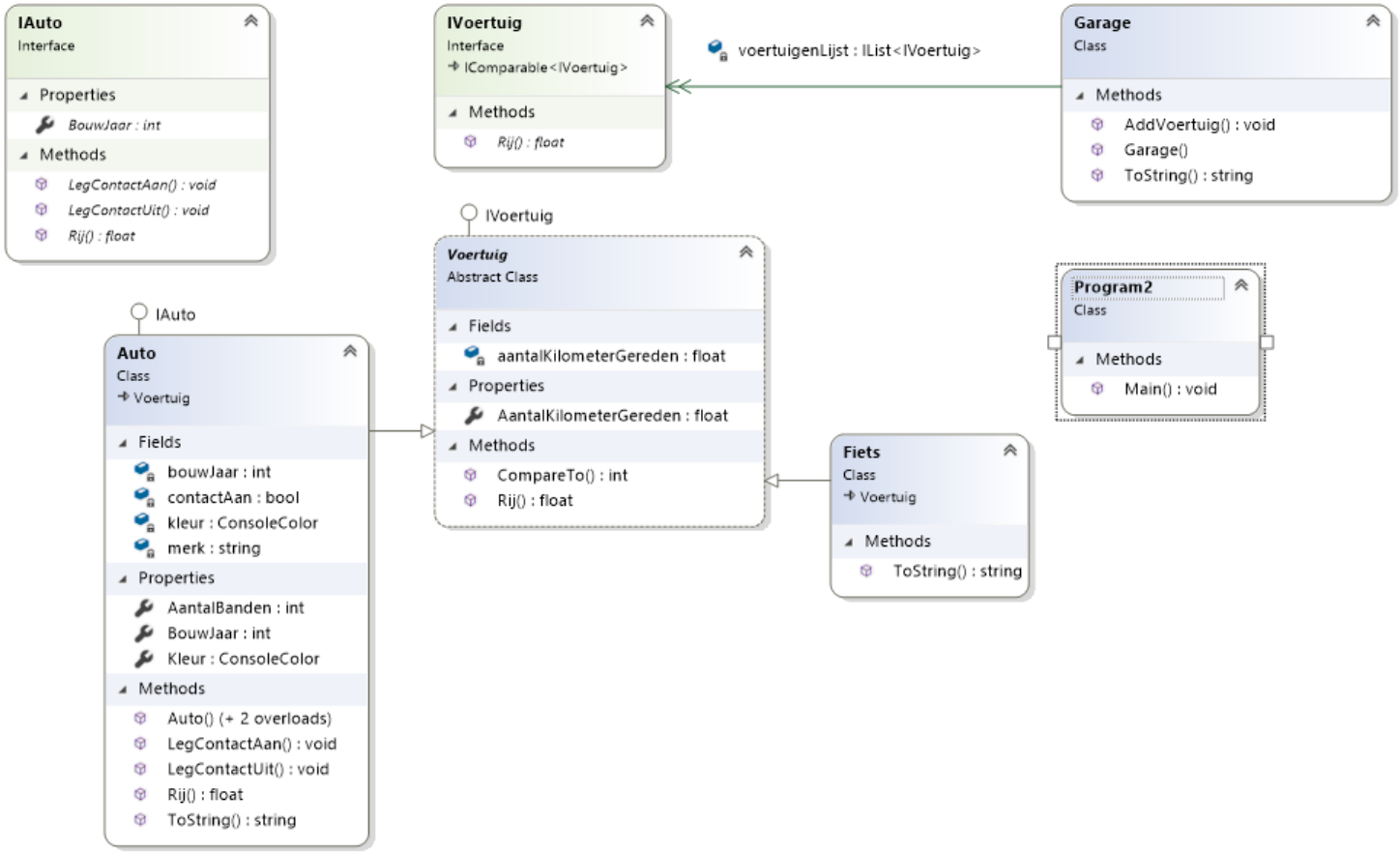
```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(() -> {
        new HogerLagerGUI().setVisible(true);
    });
}
```

`static void` **`invokeLater(Runnable runnable)`**

Causes `runnable` to have its `run` method called in the **`dispatch thread`** of **`the system EventQueue`**.

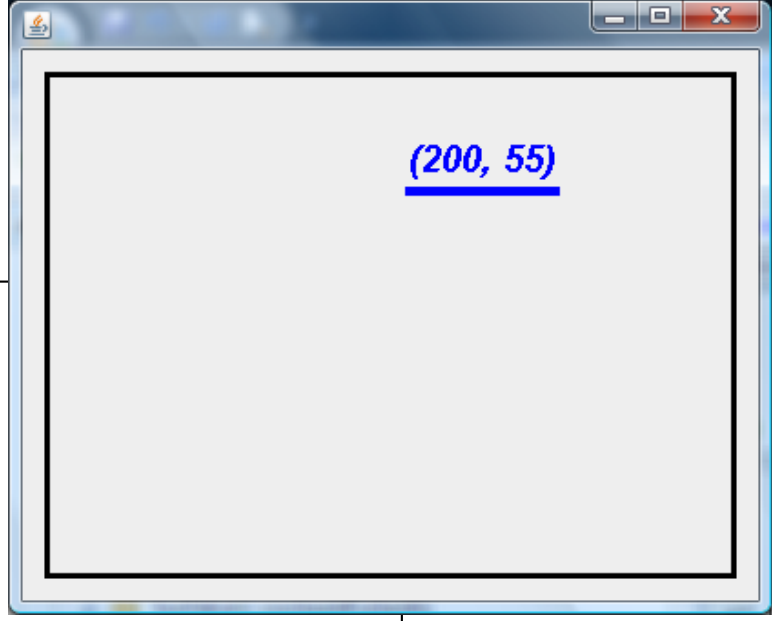
Zelf interfaces schrijven

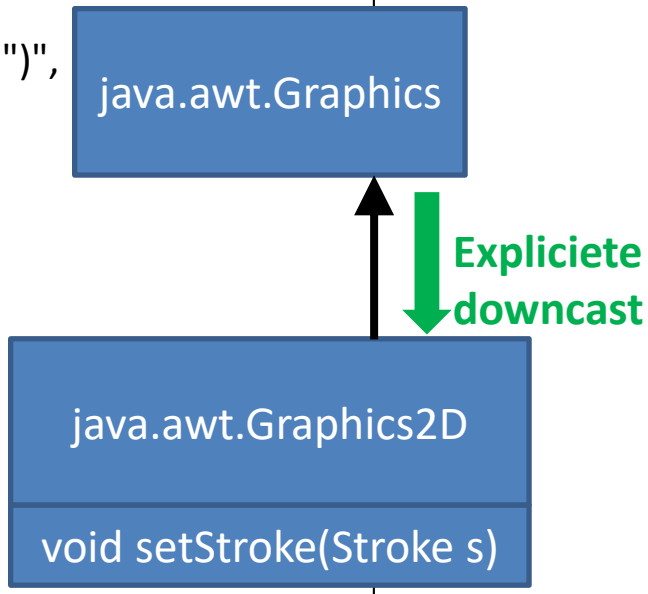
(C# klassendiagramma)



Graphics & Graphics2D

```
import java.awt.*;

public class TekenPanel extends javax.swing.JPanel {
    …
   @Override
   protected void paintComponent(Graphics g) {
      super.paintComponent(g);

      g.setFont(new Font("ARIAL BOLD", Font.ITALIC, 20));
      g.setColor(Color.BLUE);
      g.drawString("(" + this.xPosClicked + ", " + this.yPosClicked + ")",
                              xPosClicked, yPosClicked);

      Graphics2D g2d = (Graphics2D) g; //downcast
      BasicStroke stroke = new BasicStroke(5);
      g2d.setStroke(stroke);
      g2d.setColor(Color.BLUE);
      g2d.drawLine(this.xPosClicked, this.yPosClicked + 10,
                              this.xPosClicked + 80, this.yPosClicked + 10);
   }
}
```





```
BasicStroke()
BasicStroke(float width)
BasicStroke(float width, int cap, int join)
BasicStroke(float width, int cap, int join, float miterlimit)
BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)
```

Imported Items; Press 'Ctrl+SPACE' Again for All Items

java.awt.BasicStroke

public **BasicStroke**(float width)

Constructs a solid `BasicStroke` with the specified line width and with default values for the cap and join styles.

Parameters:

`width` - the width of the `BasicStroke`

Throws:

`IllegalArgumentException` - if `width` is negative

, Punt mp) {

public **BasicStroke**(float width, int cap, int join)

Constructs a solid `BasicStroke` with the specified attributes. The `miterlimit` parameter is unnecessary in cases where the default is allowable or the line joins are not specified as JOIN_MITER.

Parameters:

`width` - the width of the `BasicStroke`
`cap` - the decoration of the ends of a `BasicStroke`
`join` - the decoration applied where path segments meet

Throws:

`IllegalArgumentException` - if `width` is negative
`IllegalArgumentException` - if `cap` is not either CAP_BUTT, CAP_ROUND or CAP_SQUARE

Labo08 & Graphics2D

```
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    g.setColor(this.cirkel.getKleur());

    //https://stackoverflow.com/questions/2839508/java2d-increase-the-line-width
    Graphics2D g2d = (Graphics2D)g;
    g2d.setStroke( new BasicStroke(this.cirkel.getLijndikte()) );

    int straal = this.cirkel.getStraal();
    int x = this.getWidth()/2 - straal;
    int y = this.getHeight()/2 - straal;

    g.drawOval(x, y, 2*straal, 2*straal);
}
```

```
public abstract class Graphics2D
extends Graphics
```

setStroke

```
public abstract void setStroke(Stroke s)
```

Sets the Stroke for the Graphics2D context.

Parameters:

s - the Stroke object to be used to stroke a Shape during the rendering process

See Also:

BasicStroke, getStroke()

```
public class BasicStroke
extends Object
implements Stroke
```

```
public interface Stroke
```

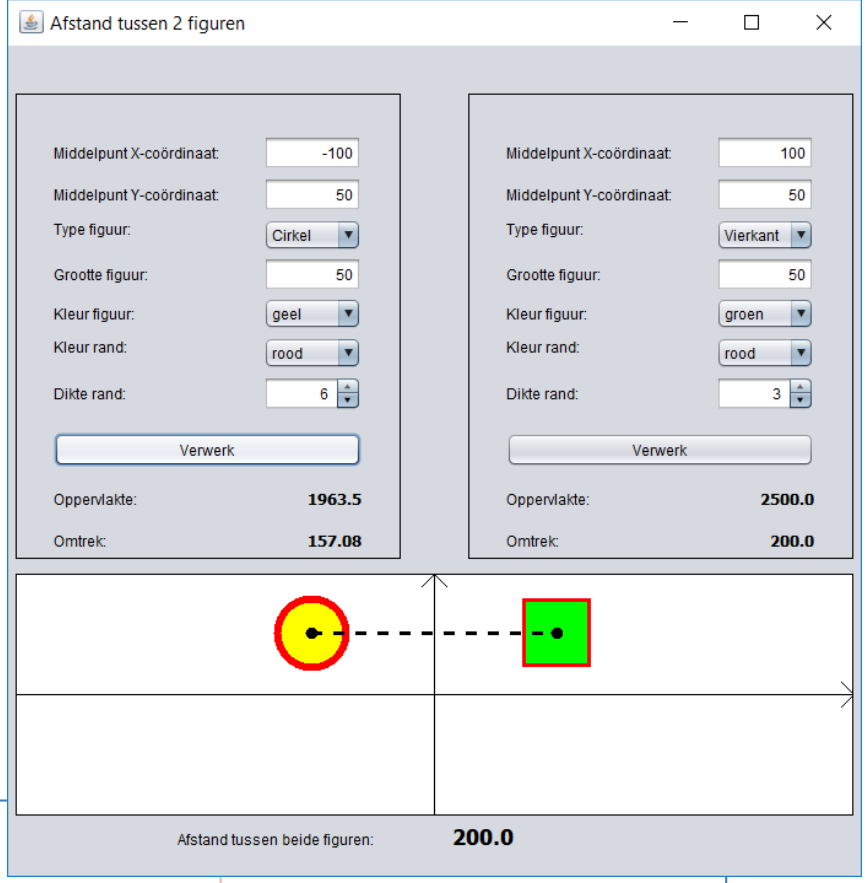
```
public BasicStroke(float width, int cap, int join, float
miterlimit, float[] dash, float dash_phase)
```

Constructs a new `BasicStroke` with the specified attributes.

Parameters:

- `width` - the width of this `BasicStroke`. The width must be greater than or equal to 0.0f. If width is set to 0.0f, the stroke is rendered as the thinnest possible line for the target device and the antialias hint setting.
- `cap` - the decoration of the ends of a `BasicStroke`
- `join` - the decoration applied where path segments meet
- `miterlimit` - the limit to trim the miter join. The miterlimit must be greater than or equal to 1.0f.
- `dash` - the array representing the dashing pattern
- `dash_phase` - the offset to start the dashing pattern

Labo09 & Graphics2D




```java
private void tekenAfstand(Graphics grphcs) {
    if (figuur1 != null && figuur2 != null) {
        grphcs.setColor(Color.black);
        grphcs.fillOval(figuur1.getMiddelpunt().getX() + oorsprong.getX() - 5,
                        -figuur1.getMiddelpunt().getY() + oorsprong.getY() - 5, 10,10);
        grphcs.fillOval(figuur2.getMiddelpunt().getX() + oorsprong.getX() - 5,
                        -figuur2.getMiddelpunt().getY() + oorsprong.getY() - 5, 10,10);

        Graphics2D g2d = (Graphics2D) grphcs;
        Stroke dashed = new BasicStroke(3, BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL, 0, new float[]{9}, 0);
        g2d.setStroke(dashed);
        g2d.drawLine(figuur1.getMiddelpunt().getX() + oorsprong.getX(),
                        -figuur1.getMiddelpunt().getY() + oorsprong.getY(),
                        figuur2.getMiddelpunt().getX() + oorsprong.getX(),
                        -figuur2.getMiddelpunt().getY() + oorsprong.getY());

    }
}
```

Samengevat

De Comparable interface

Array sorteren a.d.h.v. de methode Arrays.sort

ArrayList sorteren a.d.h.v. de methode Collections.sort

⇒ **via implementatie van interface Comparable**

⇒ public int compareTo(Object o)

⇒ **via implementatie van generische interface Comparable<T>**

⇒ public int compareTo(T t)

De Comparator interface

⇒ methode: public int compare(Object o1, Object o2)

De Runnable interface

⇒ methode: public void run()

De Stroke interface

⇒ methode: public Shape createStrokedShape(Shape p)