

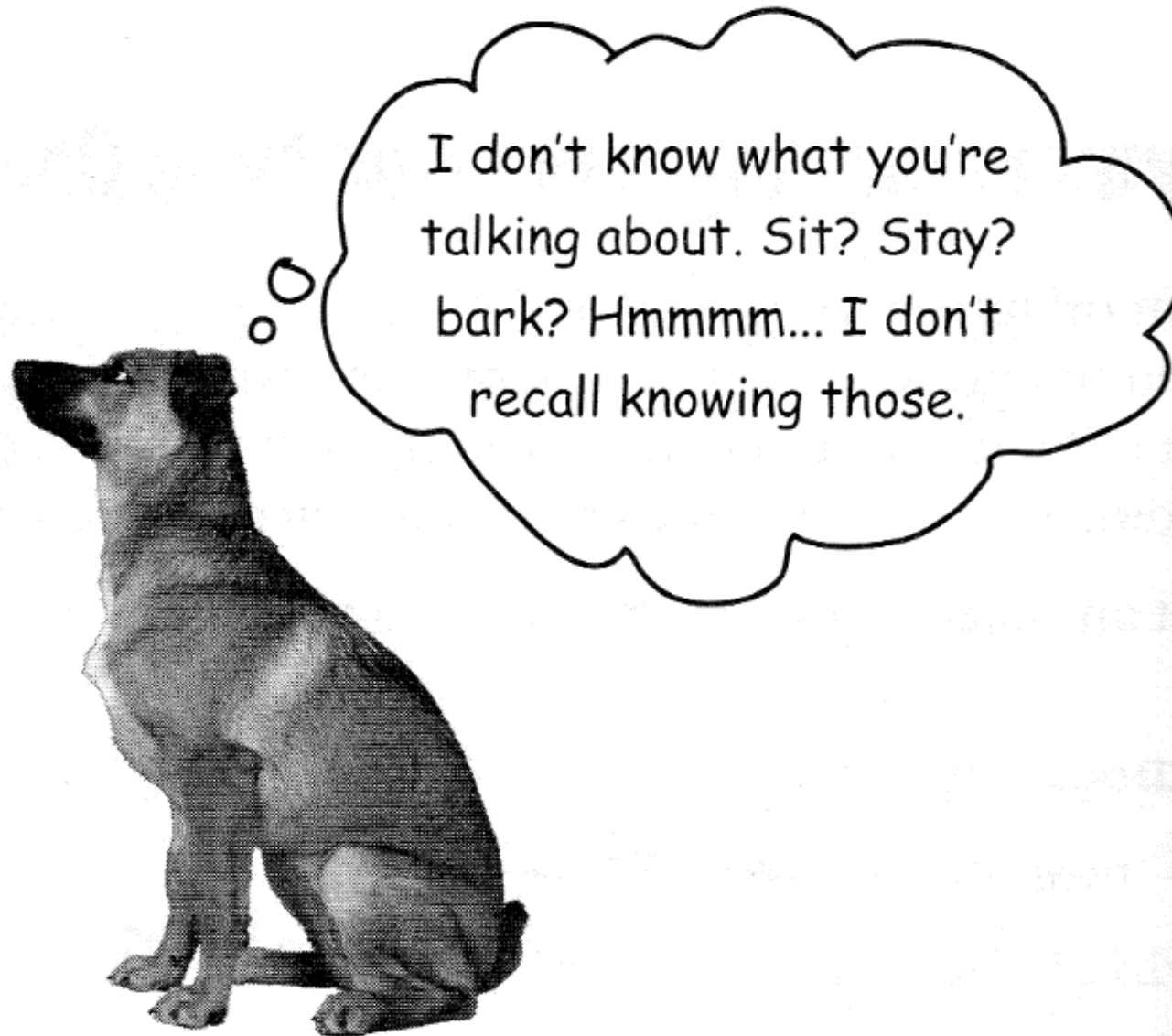


PGR 101 Objektorientert Programmering 2  
Vår 2017

# Forelesning 14.3.17

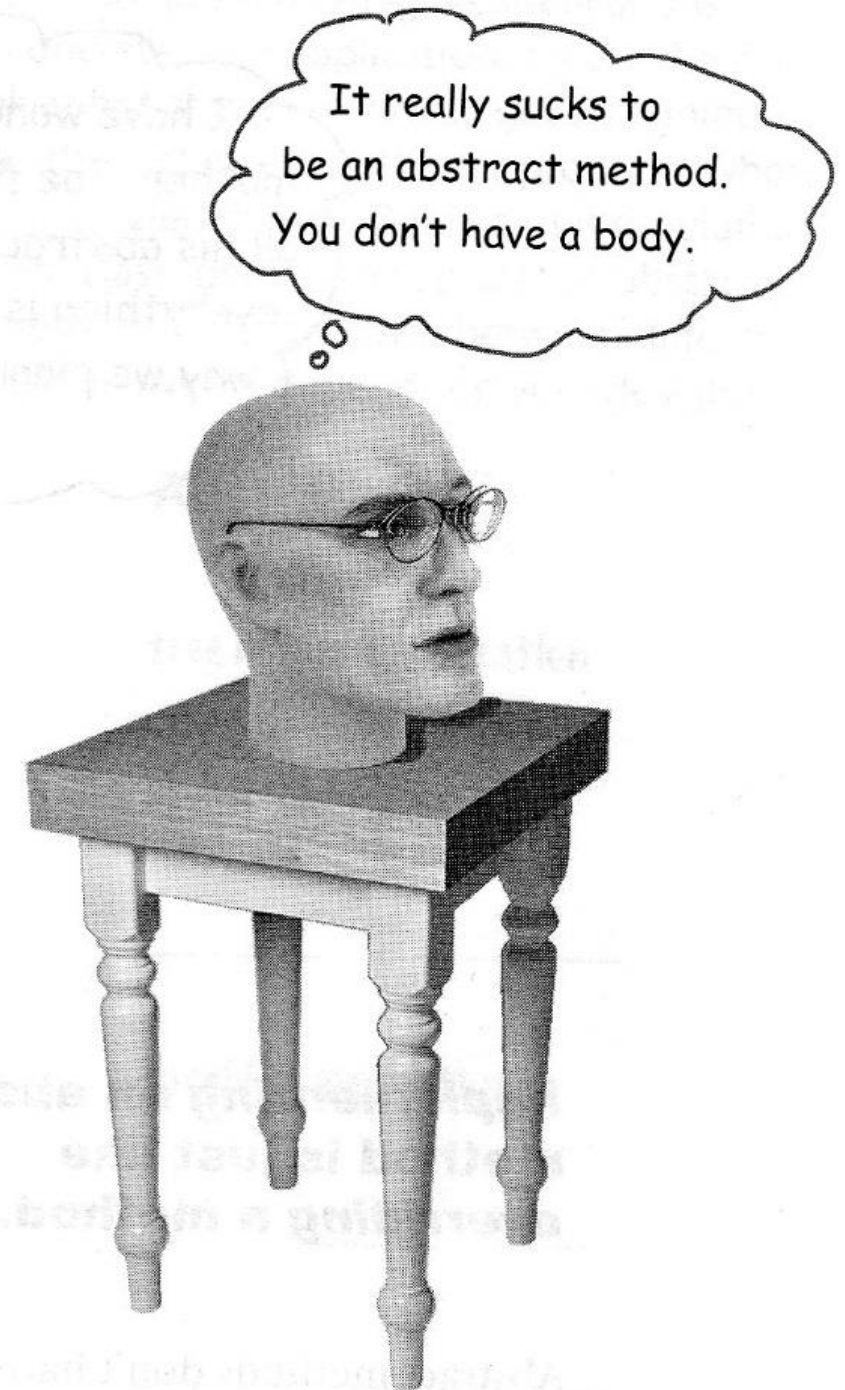
(Stein Marthinsen – [marste@westerdals.no](mailto:marste@westerdals.no))

# Oppvarming



14.05.2017

FOR101 - Objektorientert Programmering - Stein M

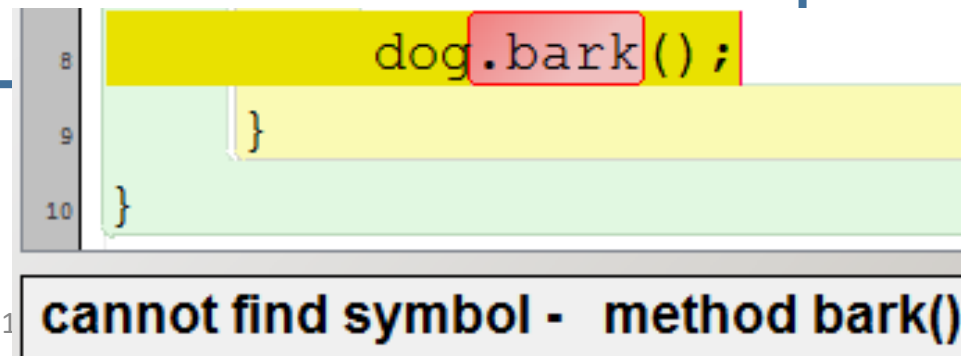
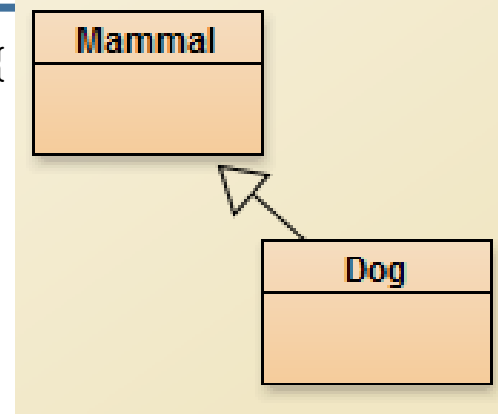


# Oppvarming - hunden

```
public class Mammal {  
    ...  
    public void eat(...) {  
        ...  
    }  
    ...  
}
```

```
public class Client {  
    public void mainMethod() {  
        Dog enHund = new Dog();  
        Mammal dog = enHund;  
        dog.bark();  
    }  
}
```

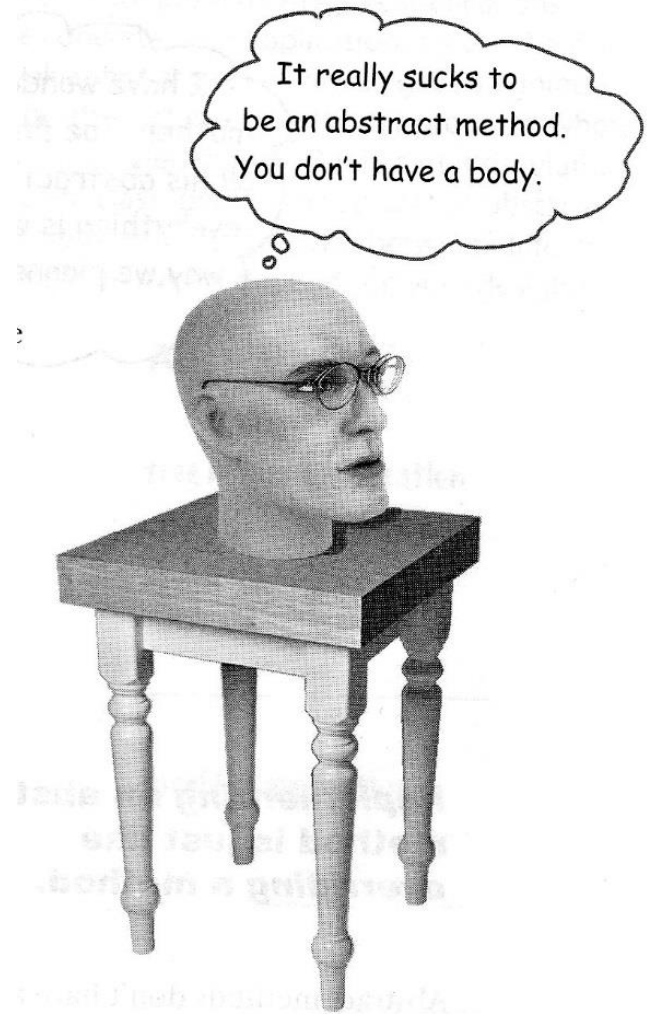
```
public class Dog extends Mammal {  
    ...  
    public void sit(...) {  
        ...  
    }  
    public void stay(...) {  
        ...  
    }  
    public void bark() {  
        System.out.println("Grrrrr VOFF!!");  
    }  
    ...  
}
```



# Oppvarming – "uten kropp"

```
public abstract double getArea();
```

```
public abstract double getPerimeter();
```



# Dagens tema

Oppvarming

Innleveringen

Eksamensoppgave

JAVA API

Interface **Comparable**

Metoden **compareTo**

# Innleveringen

Teksten er endret.

Termometer lagrer maks/min temp som et desimaltall.

# Eksamensoppgave

- a) Hva er *ikke* en superklasse/subklasse-relasjon av det følgende?
1. Software/MS Office
  2. Høgskole/Westerdals
  3. Seilbåt/Slepebåt
  4. Land/Norge
- b) Hva er *riktig* måte å kalle på en superklassekonstruktør?
1. Bruk av `super`, etterfulgt av et sett parenteser.
  2. Bruk av `super`, etterfulgt av et punktum (.).
  3. Bruk av `super`, etterfulgt av et sett parenteser med argumenter for superklassekonstruktøren.
  4. Bruk av `super`, etterfulgt av et punktum og navnet på superklassekonstruktøren.
- c) På hvilken måte kan man i en subklasse få tilgang til private fields i en superklasse?
1. Ved å kalle på private metoder deklart i superklassen
  2. Ved å kalle på public eller protected metoder deklart i superklassen
  3. Ved å bruke dem direkte
  4. Alle alternativene over

# Eksamensoppgave



Følgende klasser er gitt

```
public class Client {  
    public void mainMethod() {  
        B b = new B(1, 2, 3);  
        System.out.println(b);  
    }  
}
```

```
public class B extends A {  
    private int i;  
  
    public B(int a) {  
        setI(a);  
    }  
  
    public int getI() {..  
  
    public void setI(int i) {..  
  
    public String toString() {  
        return "i = " + i;  
    }  
}
```

```
public class A {  
    private int m;  
    private int n;  
  
    public A () {  
        m = -1;  
        n = -2;  
    }  
  
    public A (int m, int n) {  
        setM(m);  
        setN(n);  
    }  
  
    public int getM() {..  
  
    public int getN() {..  
  
    public void setM(int m) {..  
  
    public void setN(int n) {..  
  
    public String toString() {  
        return "m = " + m +  
            ", n = " + n;  
    }  
}
```

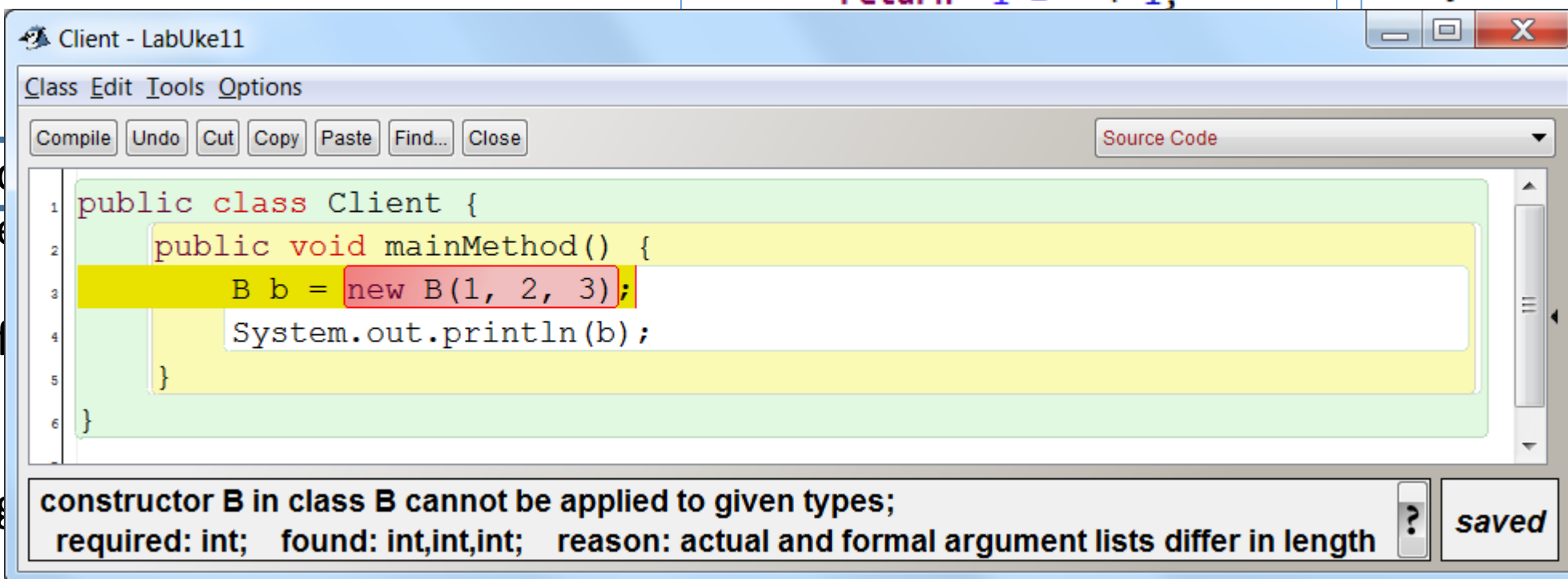
- d) Hvilken feilmelding vil kompilatoren gi?
- e) Utstyr klassen B med to versjoner av en konstruktør som retter opp feilen.
- f) Hva blir resultatet av setningen:  
    System.out.println(b);  
i mainMethod i klassen Client?
- g) Endre toString() i klassen B (på 2 måter) slik at den fungerer som tiltenkt.

Følgende klasser er gitt

```
public class Client {  
    public void mainMethod() {  
        B b = new B(1, 2, 3);  
        System.out.println(b);  
    }  
}
```

```
public class B extends A {  
    private int i;  
  
    public B(int a) {  
        setI(a);  
    }  
  
    public int getI() {..  
  
    public void setI(int i) {..  
  
    public String toString() {  
        return "i = " + i;  
    }  
}
```

```
public class A {  
    private int m;  
    private int n;  
  
    public A () {  
        m = -1;  
        n = -2;  
    }  
  
    public A (int m, int n) {  
        setM(m);  
        setN(n);  
    }  
  
    public int getM() {..  
    public int getN() {..  
  
    void setM(int m) {..  
    void setN(int n) {..  
  
    String toString() {  
        return "m = " + m +  
            ", n = " + n;  
    }  
}
```



Følgende klasser er gitt

```
public class Client {  
    public void mainMethod() {  
        B b = new B(1, 2, 3);  
        System.out.println(b);  
    }  
}
```

```
public class B extends A {  
    private int i;  
  
    public B(int a) {  
        setI(a);  
    }  
  
    public int getI() {..  
  
    public void setI(int i) {..  
  
    public String toString() {  
        return "i = " + i;  
    }  
}
```

```
public class A {  
    private int m;  
    private int n;  
  
    public A () {  
        m = -1;  
        n = -2;  
    }  
  
    public A (int m, int n) {  
        setM(m);  
        setN(n);  
    }  
  
    public int getM() {..  
  
    public int getN() {..  
  
    public void setM(int m) {..  
  
    public void setN(int n) {..  
  
    public String toString() {  
        return "m = " + m +  
            ", n = " + n;  
    }  
}
```

```
public B(int a, int b, int c) {  
    setI(a);  
    setM(b);  
    setN(c);  
}
```

i mainMethod i klassen Client?

g) Endre toString() i klassen B (på 2 måter) slik at den fungerer som tiltenkt.

Følgende klasser er gitt

```
public class Client {  
    public void mainMethod() {  
        B b = new B(1, 2, 3);  
        System.out.println(b);  
    }  
}
```

```
public class B extends A {  
    private int i;  
  
    public B(int a) {  
        setI(a);  
    }  
  
    public int getI() {..  
  
    public void setI(int i) {..  
  
    public String toString() {  
        return "i = " + i;  
    }  
}
```

```
public class A {  
    private int m;  
    private int n;  
  
    public A () {  
        m = -1;  
        n = -2;  
    }  
  
    public A (int m, int n) {  
        setM(m);  
        setN(n);  
    }  
  
    public int getM() {..  
  
    public int getN() {..  
  
    public void setM(int m) {..  
  
    public void setN(int n) {..  
  
    public String toString() {  
        return "m = " + m +  
            ", n = " + n;  
    }  
}
```



BlueJ: Terminal Window

Options

i = 1

i mainMethod i klassen Client?

g) Endre toString() i klassen B (på 2 måter) slik at den fungerer som tiltenkt.

Følgende klasser er gitt

```
public class Client {  
    public void mainMethod() {  
        B b = new B(1, 2, 3);  
        System.out.println(b);  
    }  
}
```

```
public class B extends A {  
    private int i;  
  
    public B(int a) {  
        setI(a);  
    }  
  
    public int getI() {..  
  
    public void setI(int i) {..  
  
    public String toString() {  
        return "i = " + i;  
    }  
}
```

```
public class A {  
    private int m;  
    private int n;  
  
    public A () {  
        m = -1;  
        n = -2;  
    }  
  
    public A (int m, int n) {  
        setM(m);  
        setN(n);  
    }  
  
    public void setM(int m) {..  
  
    public void setN(int n) {..  
  
    public String toString() {  
        return "m = " + m +  
            ", n = " + n;  
    }  
}
```

```
public String toString() {  
    return "i = " + i + ", m = " + getM() + ", n = " + getN();  
}
```

e) Utstyr klassen B med to versjoner av en konstruktør som retter opp feilen.

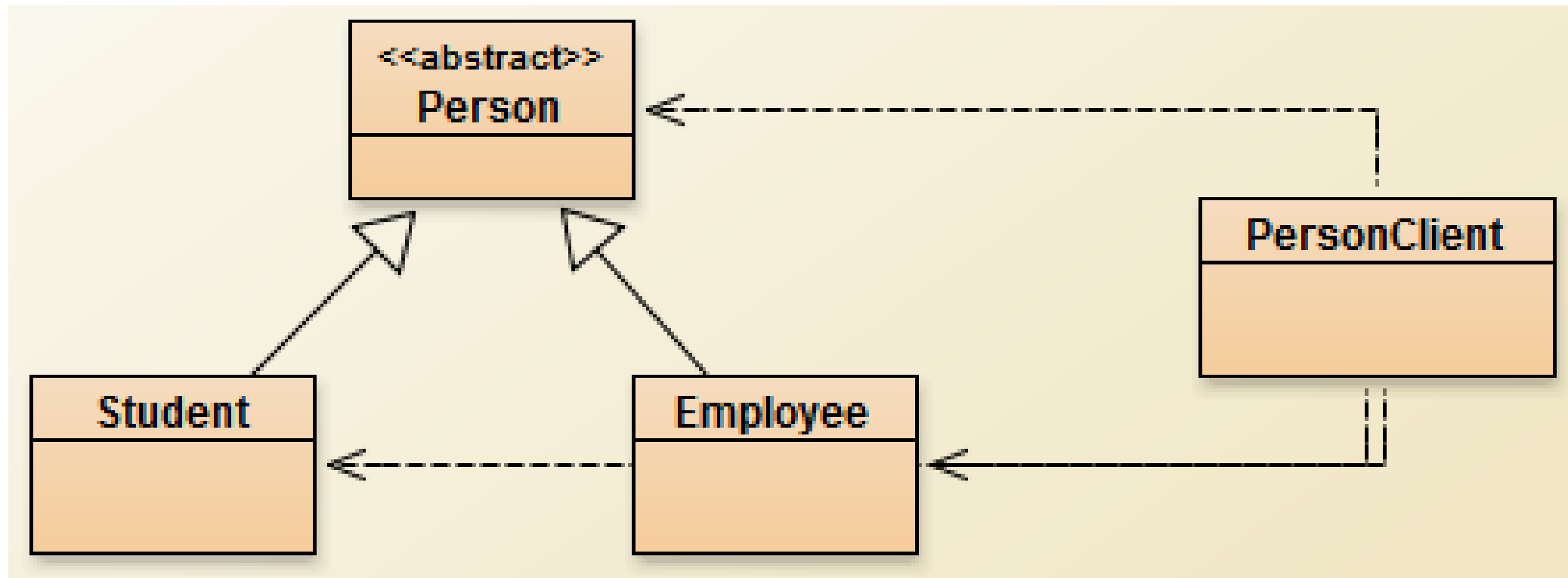
f) Hva blir resultatet av setningen:

```
System.out.println(b);  
i mainMethod i klassen Client?
```

g) Endre toString() i klassen B (på 2 måter) slik at den fungerer som tiltenkt.

```
public void setM(int m) {..  
  
    public void setN(int n) {..  
  
    public String toString() {  
        return "m = " + m +  
            ", n = " + n;  
    }  
}
```

# En samling av objekter



# En samling av objekter

```
import java.util.*;

public class PersonClient {
    public void mainMethod() {
        ArrayList<Person> persons = new ArrayList<Person>();
        persons.add(new Student("2345", "98769876", "Albin", "Albinsen", 20));
        persons.add(new Student("2312", "97939793", "Joabin", "Joabinsen", 25));
        persons.add(new Student("3122", "79867986", "Sylfest", "Sylfestsen", 20));
        persons.add(new Employee("12345", "98709870", "Dragan", "Dragansen", 250000));
        persons.add(new Employee("23451", "87698769", "Petrus", "Petrussen", 300000));
        persons.add(new Employee("34512", "76987698", "Doris", "Dorissen", 450000));

        for (Person p : persons) {
            System.out.println(p);
        }
    }
}
```

# En samling av objekter

```
import java.util.*;

public class PersonClient {
    public void mainMethod() {
        ArrayList<Person> persons = new ArrayList<Person>();
        persons.add(new Student("2345", "98769876", "Albin", "Albinsen", 20));
        persons.add(new Student("2312", "97939793", "Joabin", "Joabinsen", 25));
    }
}
```

(98769876)	Albin Albinsen (2345) 20
(97939793)	Joabin Joabinsen (2312) 25
(79867986)	Sylfest Sylfestsen (3122) 20
(98709870)	Dragan Dragansen(12345) Lønn: 250000
(87698769)	Petrus Petrusen(23451) Lønn: 300000
(76987698)	Doris Dorissen(34512) Lønn: 450000



# Sortering

Hvis en samling skal kunne sorteres, må elementene kunne *sammenlignes*.

Det må være klart hva som er sammenlignings-kriteriet!

Det må kunne fastslås hvordan den innbyrdes relasjonen mellom to elementer er.



# Tenk på dette...

Går det å vise **Person**-samlingen sortert på personnummer?

Hvordan sorterer man en samling Person-objekter?

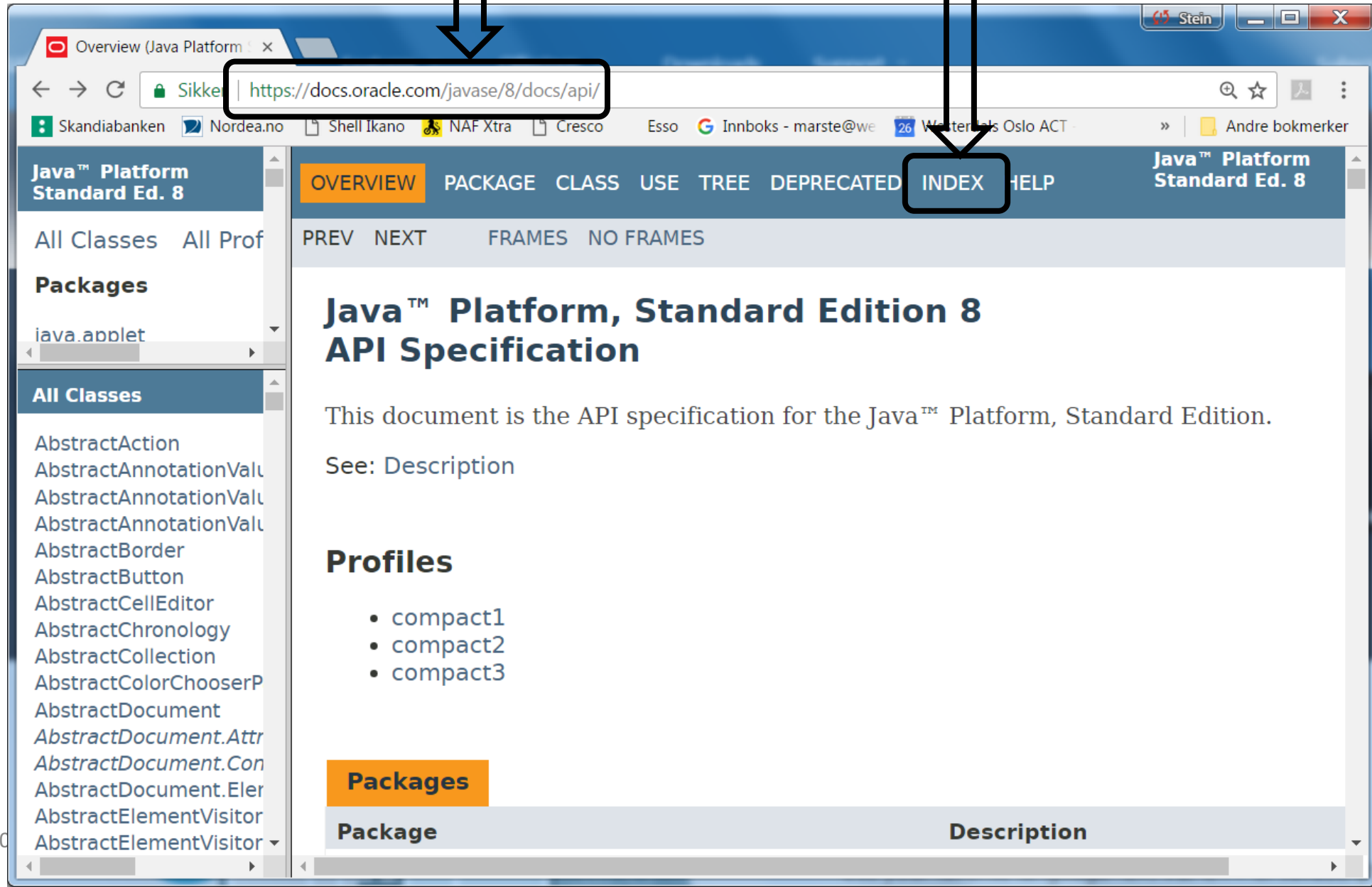
Må vi skrive koden for dette selv?

Finnes det i JAVA-biblioteket en ferdig metode som gjør dette?

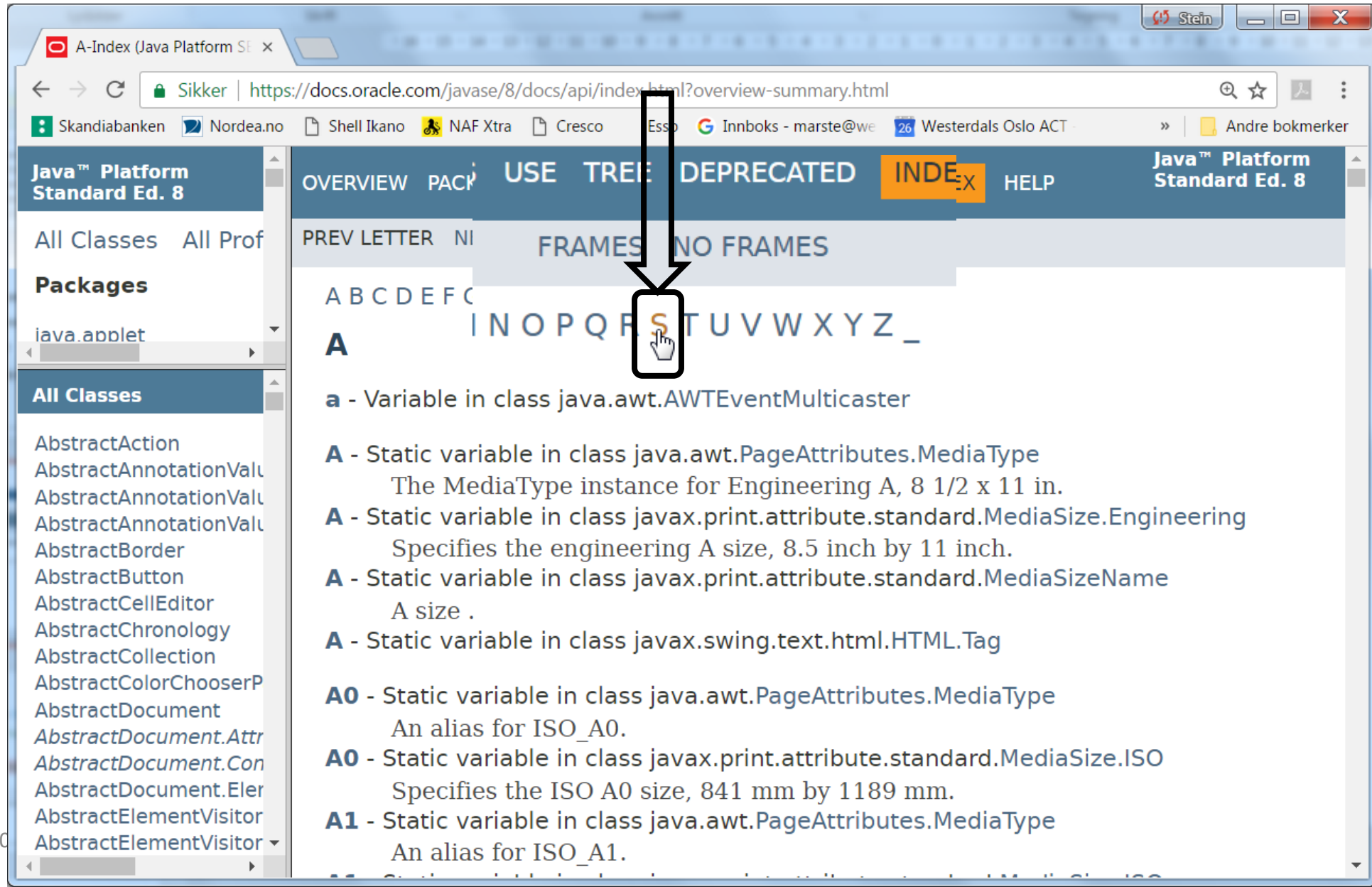
Hvis så – kan vi bruke denne for samlingen av **Person**-objekter?

"Vet" denne metoden hvordan **Person**-objekter skal sorteres?

# Sjekker JAVA API...



# Sjekker JAVA API...



# Sjekker JAVA API...

order, according to the natural ordering of its elements.

**sort(T[], Comparator<? super T>)** - Static method in class java.util.Arrays  
Sorts the specified array of objects according to the order induced by the specified comparator.

**sort(T[], int, int, Comparator<? super T>)** - Static method in class java.util.Arrays  
Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

**sort(List<T>)** - Static method in class java.util.Collections  
Sorts the specified list into ascending order, according to the natural ordering of its elements.

**sort(List<T>, Comparator<? super T>)** - Static method in class java.util.Collections  
Sorts the specified list according to the order induced by the specified comparator.

**sort(Comparator<? super E>)** - Method in class java.util.concurrent.CopyOnWriteArrayList

**sort(Comparator<? super E>)** - Method in interface java.util.List  
Sorts this list according to the order induced by the specified Comparator.

**sort(Comparator<? super E>)** - Method in class java.util.Vector

**sort()** - Method in class javax.swing.DefaultRowSorter  
Sorts and filters the rows in the view based on the sort keys of the columns currently being sorted and the filter, if any, associated with this sorter.



# Sjekker JAVA API...

order, according to the natural ordering of its elements.

**sort(T[], Comparator<? super T>)** - Static method in class java.util.Arrays  
Sorts the specified array of objects according to the order induced by the specified comparator.

**sort(T[], int, int, Comparator<? super T>)** - Static method in class java.util.Arrays  
Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

**sort(List<T>)** - Static method in class java.util.Collections  
Sorts the specified list into ascending order, according to the natural ordering of its elements.

**sort(List<T>, Comparator<? super T>)** - Static method in class java.util.Collections  
Sorts the specified list according to the order induced by the specified comparator.

**sort(Comparator<? super E>)** - Method in class java.util.concurrent.CopyOnWriteArrayList

**sort(Comparator<? super E>)** - Method in interface java.util.List  
Sorts this list according to the order induced by the specified Comparator.

**sort(Comparator<? super E>)** - Method in class java.util.Vector

**sort()** - Method in class javax.swing.DefaultRowSorter  
Sorts and filters the rows in the view based on the sort keys of the columns currently being sorted and the filter, if any, associated with this sorter.

# Sjekker JAVA API...

**sort**

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the **natural ordering** of its elements. **All elements in the list must implement the Comparable interface.** Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

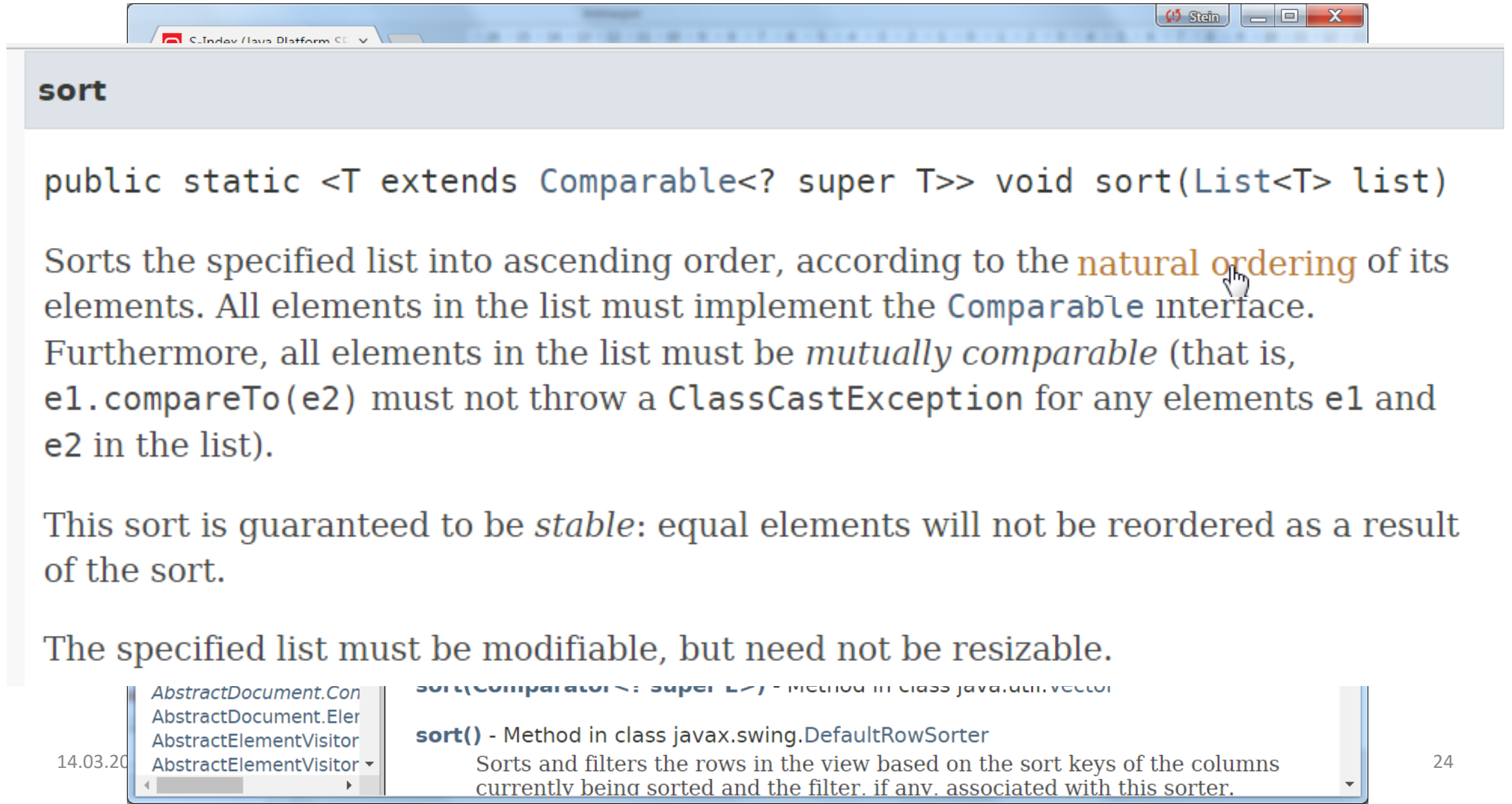
The specified list must be modifiable, but need not be resizable.

AbstractDocument.Com  
AbstractDocument.Eler  
AbstractElementVisitor  
AbstractElementVisitor

14.03.20

**sort()** - Method in class javax.swing.DefaultRowSorter  
Sorts and filters the rows in the view based on the sort keys of the columns currently being sorted and the filter, if any, associated with this sorter.

# Sjekker JAVA API...



The screenshot shows a Java IDE window titled "S-Index / Java Platform SE". The main content area displays the `sort` method signature and its description. The signature is `public static <T extends Comparable<? super T>> void sort(List<T> list)`. The description states: "Sorts the specified list into ascending order, according to the **natural ordering** of its elements. All elements in the list must implement the `Comparable` interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list). This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort. The specified list must be modifiable, but need not be resizable."

At the bottom, there is a scrollable list of search results. The first result is "AbstractDocument.Con", followed by "AbstractDocument.Eler", "AbstractElementVisitor", and "AbstractElementVisitor". The second result is "sort(Comparator<? super T>) - Method in class java.util.Vector". The third result is "sort() - Method in class javax.swing.DefaultRowSorter", which includes a description: "Sorts and filters the rows in the view based on the sort keys of the columns currently being sorted and the filter, if any, associated with this sorter."

14.03.20



# Sjekker JAVA API...

The screenshot shows a web browser window displaying the Java Platform Standard Ed. 8 API documentation for the `Comparable` interface. The browser's address bar shows the URL `https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html`. The page has a dark blue header with navigation links: OVERVIEW, PACKAGE, CLASS (highlighted), USE, TREE, DEPRECATED, INDEX, and HELP. Below the header, there are links for PREVIOUS CLASS, NEXT CLASS, FRAMES, and NO FRAMES. The main content area is titled "Interface Comparable<T>" and includes sections for "Type Parameters:", "All Known Subinterfaces:", and "All Known Implementing Classes:". The "Type Parameters:" section explains that `T` is the type of objects that this object may be compared to. The "All Known Subinterfaces:" section lists `ChronoLocalDate`, `ChronoLocalDateTime<D>`, `Chronology`, `ChronoZonedDateTime<D>`, `Delayed`, `Name`, `Path`, `RunnableScheduledFuture<V>`, and `ScheduledFuture<V>`. The "All Known Implementing Classes:" section lists `AbstractChronology`, `AbstractRegionPainter.PaintContext.CacheMode`, `AccessMode`, `AclEntryFlag`, `AclEntryPermission`, `AclEntryType`, and `...`. On the left side of the browser window, there is a sidebar with a search bar and a list of packages and classes. The sidebar is titled "Java™ Platform Standard Ed. 8" and includes links for "All Classes" and "All Prof". Below these links, there is a section for "Packages" and a section for "All Classes" which lists various classes including `AbstractAction`, `AbstractAnnotationValu`, `AbstractAnnotationValu`, `AbstractAnnotationValu`, `AbstractBorder`, `AbstractButton`, `AbstractCellEditor`, `AbstractChronology`, `AbstractCollection`, `AbstractColorChooserP`, `AbstractDocument`, `AbstractDocument.Attr`, `AbstractDocument.Con`, `AbstractDocument.Eler`, and `AbstractElementVisitor`.

Comparable (Java Platform Standard Ed. 8)

Sikker | <https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html>

Skandiabanken Nordea.no Shell Ikano NAF Xtra Cresco Esso Innboks - marste@we 26 Westerdals Oslo ACT Andre bokmerker

Java™ Platform Standard Ed. 8

OVERVIEW PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3  
java.lang

**Interface Comparable<T>**

**Type Parameters:**  
T - the type of objects that this object may be compared to

**All Known Subinterfaces:**  
`ChronoLocalDate`, `ChronoLocalDateTime<D>`, `Chronology`, `ChronoZonedDateTime<D>`, `Delayed`, `Name`, `Path`, `RunnableScheduledFuture<V>`, `ScheduledFuture<V>`

**All Known Implementing Classes:**  
`AbstractChronology`, `AbstractRegionPainter.PaintContext.CacheMode`, `AccessMode`, `AclEntryFlag`, `AclEntryPermission`, `AclEntryType`, `...`

sort

publ

Sorts

elem

Furth

e1.c

e2 in

This

of the

The s

14.03.20

list)

of its

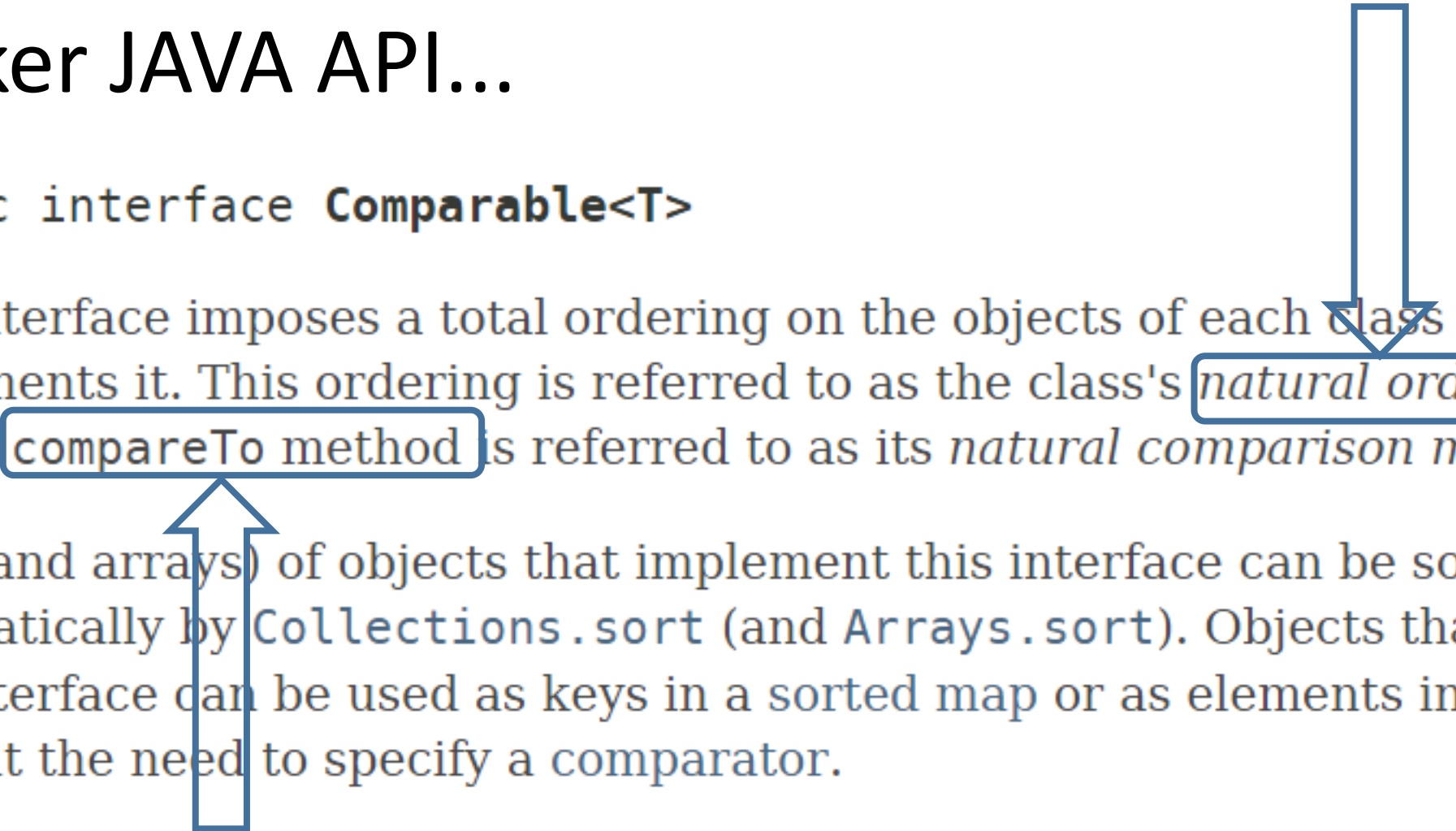
and

result

# Sjekker JAVA API...

```
public interface Comparable<T>
```

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's *compareTo* method is referred to as its *natural comparison method*.



Lists (and arrays) of objects that implement this interface can be sorted automatically by `Collections.sort` (and `Arrays.sort`). Objects that implement this interface can be used as keys in a sorted map or as elements in a sorted set, without the need to specify a comparator.

# Klassen Person

1. Må *implementere* **interface Comparable**.
2. Må (som følge av dette!) ha metoden **compareTo**.

## **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.

# Klassen Person

1. Implementerer **interface Comparable**.

```
public abstract class Person implements Comparable <Person>
```

2. Har metoden **compareTo**.

```
public int compareTo(Person p)
```

# Klassen Person

1. Implementerer **interface Comparable**.

```
public abstract class Person implements Comparable <Person>
```

2. Har metoden **compareTo**.

```
public int compareTo(Person p)
```

Husk fra forrige gang – hvordan **Person** sin **equals** brukte **String** sin **equals**.

# Klassen Person

1. Implementerer **interface Comparable**.

```
public abstract class Person implements Comparable <Person>
```

2. Har metoden **compareTo**.

```
public int compareTo(Person p) {  
    int result = getSocialSecurityNumber().  
        compareTo(p.getSocialSecurityNumber());  
    return result;  
}
```

**String** sin **compareTo**

# En *sortert* samling av objekter

```
import java.util.*;

public class PersonClient {
    public void mainMethod() {
        ArrayList<Person> persons = new ArrayList<Person>();
        persons.add(new Student("2345", "98769876", "Albin", "Albinsen", 20));
        persons.add(new Student("2312", "97939793", "Joabin", "Joabinsen", 25));
        persons.add(new Student("3122", "79867986", "Sylfest", "Sylfestsen", 20));
        persons.add(new Employee("12345", "98709870", "Dragan", "Dragansen", 250000));
        persons.add(new Employee("23451", "87698769", "Petrus", "Petrussen", 300000));
        persons.add(new Employee("34512", "76987698", "Doris", "Dorissen", 450000));

        Collections.sort(persons);

        for (Person p : persons) {
            System.out.println(p);
        }
    }
}
```

# En *sortert* samling av objekter

```
import java.util.*;
```

```
public class PersonClient {
```

```
    public void mainMethod() {
```

```
        ArrayList<Person> persons = new ArrayList<Person>();
```

```
(76987698) Doris Dorissen(34512) Lønn: 450000
```

```
(79867986) Sylfest Sylfestsen (3122) 20
```

```
(87698769) Petrus Petrussen(23451) Lønn: 300000
```

```
(97939793) Joabin Joabinsen (2312) 25
```

```
(98709870) Dragan Dragansen(12345) Lønn: 250000
```

```
(98769876) Albin Albinsen (2345) 20
```

```
    }
```

```
}
```



# interface

En klasse som du har lyst til å bruke på en bestemt måte, må (kanskje) oppfylle bestemte krav.

Et interface kan oppfattes som formulering av slike krav.

Objekter av en klasse oppfyller slike krav ved at *klassen* **"implementerer"** et (eller flere) interface.

```
public abstract class Person implements Comparable <Person>
```

I dette tilfellet vil vi f.eks. kunne sortere en samling **Person**-objekter.

# Oppgaver på øvingen

Jobb med Innlevering 1.

Ekstra: Gjør det mulig å sortere Instrumentene.