

Forelæsning Uge 5 – Mandag

- **Sortering ved hjælp af klassen Collections**
 - Ved brug af interfacet Comparable
 - Ved brug af interfacet Comparator
- **findBest som sorteringsproblem**
- **Information om køreprøven**
 - Form
 - Forberedelse



● Sortering via Collections og Comparable

- **Klassen Collections indeholder en række nyttige metoder**
 - Metoderne kan bruges på forskellige typer af **objektsamlinger**
 - Typen af objektsamlingen skal implementere **Collection** interfacet
 - Det er f.eks. tilfældet for ArrayList

```
T min(Collection<T> c)    // Returnerer mindste element
T max(Collection<T> c)    // Returnerer største element

void sort(List<T> l)       // Sorterer listen
void shuffle(List<T> l)    // Blander listen
void reverse(List<T> l)    // Vender listen om

...
...
```

Alle metoderne er
klassemetoder
Collections.metode()

Brug af Collections på ArrayList<String>

```
public class TestDriver {  
    public static void run() {
```

Test klasse med
klassemetode

```
        ArrayList<String> list;  
        list = new ArrayList<>();
```

Lokal variabel, der
initialiseres til at være
en tom arrayliste

```
        list.add("Cecilie");  
        list.add("Erik");  
        list.add("Adam");  
        list.add("Bo");  
        list.add("Dora");
```

```
        print("*****");
```

```
        print("liste: " + list);
```

```
        print("min: " + Collections.min(list));
```

```
        print("max: " + Collections.max(list));
```

```
        Collections.sort(list);
```

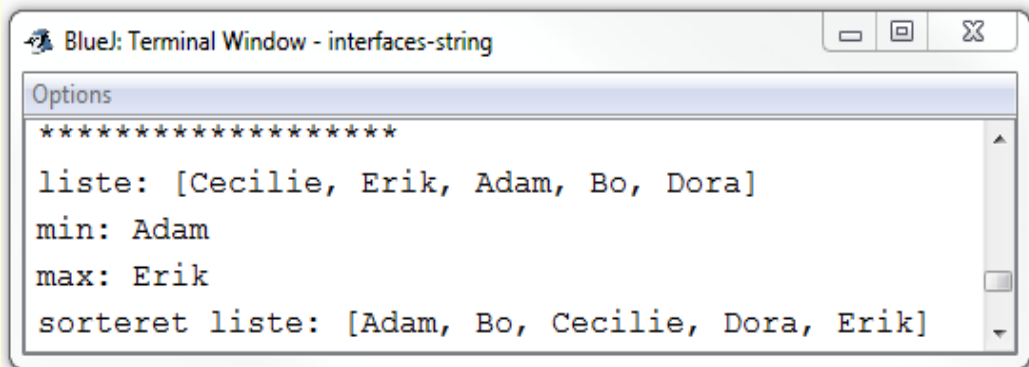
```
        print("sorteret liste: " + list);
```

```
    }  
    private static void print(Object o) {
```

```
        System.out.println(o);
```

Hjælpemetode

print metoden
kalder implicit
toString
metoden



```
Blue: Terminal Window - interfaces-string  
Options  
*****  
liste: [Cecilie, Erik, Adam, Bo, Dora]  
min: Adam  
max: Erik  
sorteret liste: [Adam, Bo, Cecilie, Dora, Erik]
```

Brug af Collections på ArrayList<String>

```
public class TestDriver {  
    public static void run()  
    {  
        ArrayList<String> list = new ArrayList<>();  
        ...  
    }  
}
```

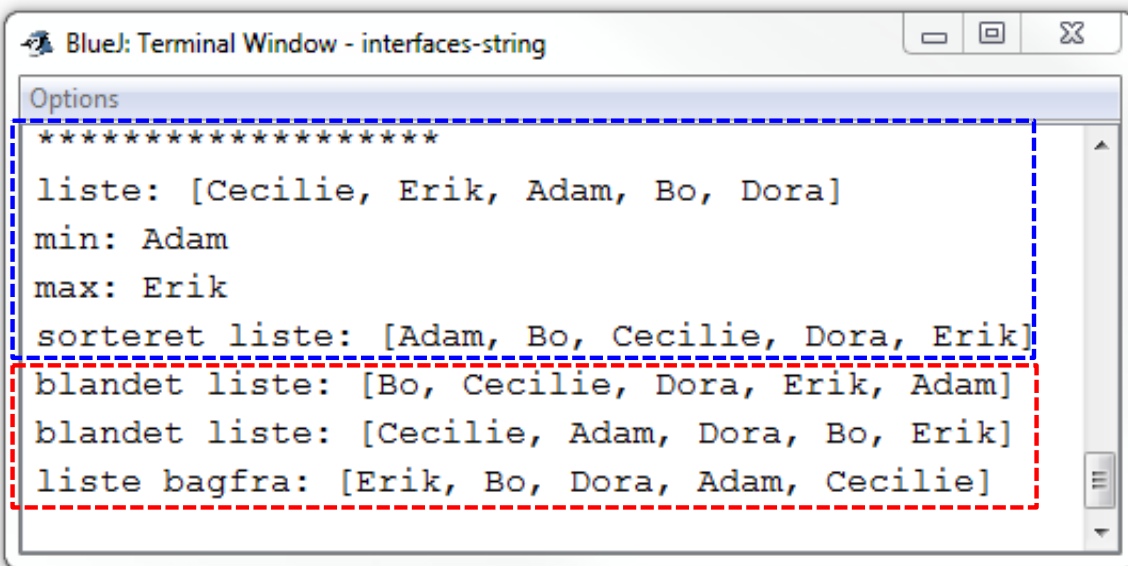
Som
før

```
    print("*****");  
    print("liste: " + list);  
    print("min: " + Collections.min(list));  
    print("max: " + Collections.max(list));  
    Collections.sort(list);  
    print("sorteret liste: " + list);
```

Nyt

```
    Collections.shuffle(list);  
    print("blandet liste: " + list);  
    Collections.shuffle(list);  
    print("blandet liste: " + list);  
    Collections.reverse(list);  
    print("liste bagfra: " + list);
```

```
    }  
    ...  
}
```



```
BlueJ: Terminal Window - interfaces-string  
Options  
*****  
liste: [Cecilie, Erik, Adam, Bo, Dora]  
min: Adam  
max: Erik  
sorteret liste: [Adam, Bo, Cecilie, Dora, Erik]  
blandet liste: [Bo, Cecilie, Dora, Erik, Adam]  
blandet liste: [Cecilie, Adam, Dora, Bo, Erik]  
liste bagfra: [Erik, Bo, Dora, Adam, Cecilie]
```

Brug af Collections på ArrayList<Person>

```
public class TestDriver {  
    public static void run() {
```

← Test klasse med
klassemetode

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

← Erklær og initialiser
lokal variabel, der
er en arrayliste

```
        list.add(new Person("Cecilie", 18));  
        list.add(new Person("Erik", 16));  
        list.add(new Person("Adam", 16));  
        list.add(new Person("Bo", 39));  
        list.add(new Person("Dora", 47));
```

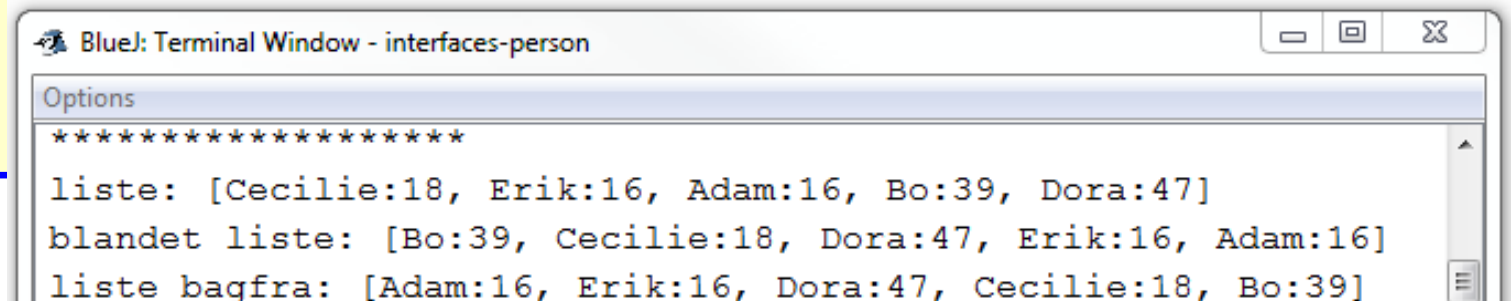
← Tilføj 5
Person
objekter

```
        print("*****");  
        print("liste: " + list);  
        Collections.shuffle(list);  
        print("blandet liste: " + list);  
        Collections.reverse(list);  
        print("liste bagfra: " + list);
```

```
    }
```

```
    ...
```

```
}
```



BlueJ: Terminal Window - interfaces-person

```
Options  
*****  
liste: [Cecilie:18, Erik:16, Adam:16, Bo:39, Dora:47]  
blandet liste: [Bo:39, Cecilie:18, Dora:47, Erik:16, Adam:16]  
liste bagfra: [Adam:16, Erik:16, Dora:47, Cecilie:18, Bo:39]
```

Brug af Collections på ArrayList<Person>

```
public class TestDriver {  
    public static void run() {  
        ArrayList<Person> list = new ArrayList<Person>();  
        list.add(new Person("John", 25));  
        list.add(new Person("Jane", 30));  
        list.add(new Person("Bob", 35));  
        list.add(new Person("Alice", 40));  
        list.add(new Person("Charlie", 45));  
        print("*****");  
        print("liste: " + list);  
        print("min: " + Collections.min(list));  
        print("max: " + Collections.max(list));  
        Collections.sort(list);  
        print("sorteret liste: " + list);  
    }  
    ...  
}
```

no suitable method found for sort(java.util.ArrayList<Person>)
method java.util.Collections.<T>sort(java.util.List<T>) is not applicable
(inference variable T has incompatible bounds
equality constraints: Person
upper bounds: java.lang.Comparable<? super T>)
method
java.util.Collections.<T>sort(java.util.List<T>, java.util.Comparator<? super T>) is not applicable
(cannot infer type-variable(s) T
(actual and formal argument lists differ in length))

Oversætteren kan ikke finde en passende sort metode

- Collections klassen har godt nok to sort metoder
- Den med én parameter kan ikke bruges, fordi Comparable ikke er implementeret for Person klassen
- Den med to parametre kan ikke bruges, fordi kaldet kun har én

Hvad gik galt?

- **Metoderne min, max og sort i Collections kan kun anvendes, hvis elementerne i arraylisten har en **ordning****
 - String klassen har en indbygget ordning (alfabetisk ordning)
 - Derfor kunne vi bruge min, max og sort på ArrayList<String>
- **Person klassen (som vi selv har lavet) har (endnu ikke) en ordning**
 - Derfor kan vi ikke bruge min, max og sort på ArrayList<Person>
 - Men vi kan godt bruge shuffle og reverse, idet disse metoder ikke kræver en ordning

Quiz

Ordning kan defineres via interfacet Comparable

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

Hoved for metode (implementationen mangler)
Implementationen laves i de klasser, der anvender interfacet

Tænk på et interface som en rolle

- Person objekter kan spille rollen Comparable, hvis to ting er opfyldt

Person klassen hoved skal angive, at den vil implementere interfacet

```
public class Person implements Comparable<Person> {  
    ...  
    public int compareTo(Person p) {  
        ...  
    }  
}
```

T = Person

T = Person

Person klassen skal implementere en compareTo metode med den returtype og de parametertyper, der er specificeret i interfacet

Metoden skal sammenligne to objekter af type Person, nemlig this og p og angive deres ordning i returværdien

Det objekt metoden kaldes på

Det objekt parameteren angiver

- $\text{this} < p \Rightarrow$ negativ
- $\text{this} = p \Rightarrow 0$
- $\text{this} > p \Rightarrow$ positiv

Ordningen, som compareTo definerer, kaldes den NATURLIGE ORDNING

Vi kigger nærmere på interfaces i Kap. 12

compareTo kan implementeres på mange måder

- **Vi kan ordne (alfabetisk) efter personens navn**
 - Til dette formål kan vi bruge compareTo metoden fra String klassen

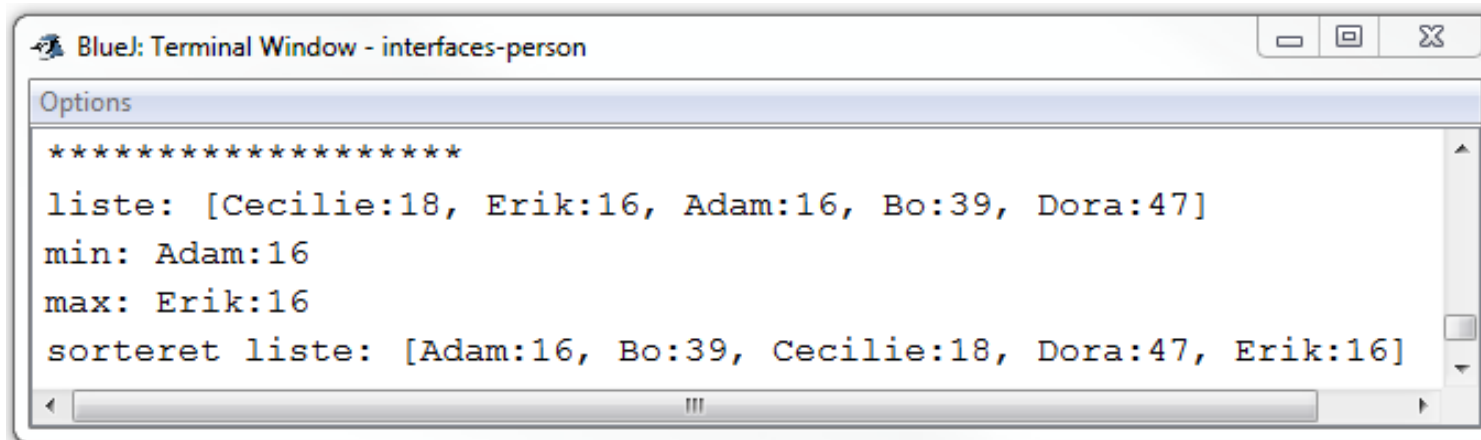
```
public int compareTo(Person p) {  
    return this.name.compareTo(p.name);  
}
```

Personens eget navn
(this kan udelades)

Metode fra
String klassen
(alfabetisk ordning)

Navnet på
personen p

- Bemærk at vi kan referere direkte til den **private** feltvariabel name (uden brug af accessor metode)
- Feltvariablen er privat for **klassen** (ikke privat for objektet)



BlueJ: Terminal Window - interfaces-person

```
Options  
*****  
liste: [Cecilie:18, Erik:16, Adam:16, Bo:39, Dora:47]  
min: Adam:16  
max: Erik:16  
sorteret liste: [Adam:16, Bo:39, Cecilie:18, Dora:47, Erik:16]
```

Vi kan ordne efter personens alder

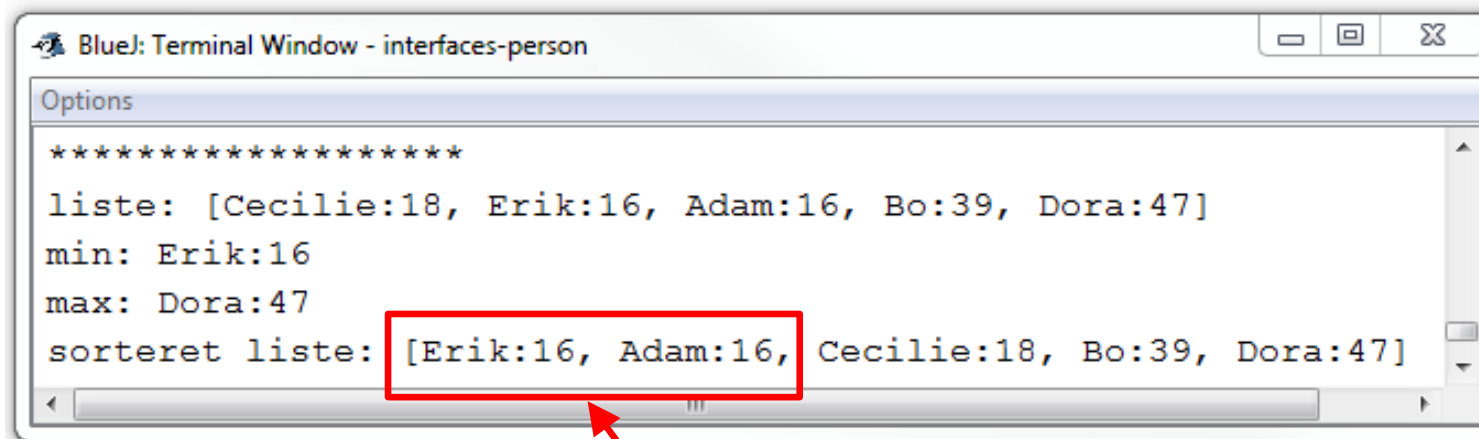
Yngste først

```
public int compareTo(Person p) {  
    if(this.age == p.age) {  
        return 0;  
    }  
    if(this.age < p.age) {  
        return -1;  
    }  
    else {  
        return +1;  
    }  
}
```

- $\text{this} < p \Rightarrow \text{negativ}$
- $\text{this} = p \Rightarrow 0$
- $\text{this} > p \Rightarrow \text{positiv}$

Simplere løsning

```
public int compareTo(Person p) {  
    return this.age - p.age;  
}
```



BlueJ: Terminal Window - interfaces-person

Options

```
*****  
liste: [Cecilie:18, Erik:16, Adam:16, Bo:39, Dora:47]  
min: Erik:16  
max: Dora:47  
sorteret liste: [Erik:16, Adam:16, Cecilie:18, Bo:39, Dora:47]
```

The screenshot shows a terminal window with the output of a sorting operation. The initial list is [Cecilie:18, Erik:16, Adam:16, Bo:39, Dora:47]. The minimum is Erik:16 and the maximum is Dora:47. The sorted list is [Erik:16, Adam:16, Cecilie:18, Bo:39, Dora:47]. A red box highlights the first two elements of the sorted list, [Erik:16, Adam:16], and a red arrow points to this box from the text below.

Hvis to personer har samme alder, er rækkefølgen i listen uændret

Vi kan kombinere de to ordningskriterier

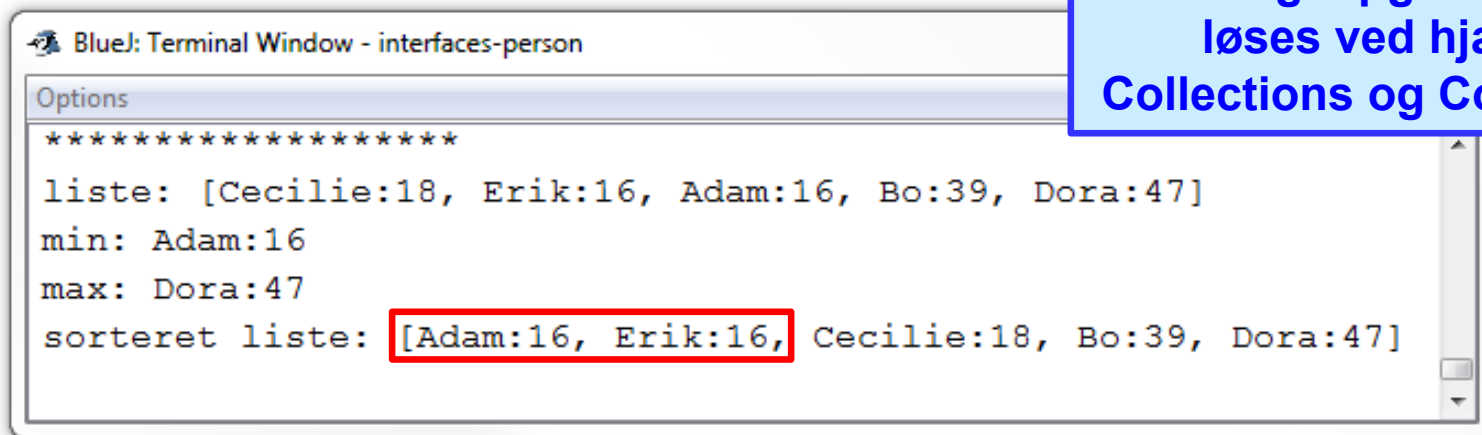
- Vi ordner **primært** efter alder, men hvis to personer er lige gamle ordnes **sekundært** alfabetisk efter navn

```
public int compareTo(Person p) {  
    if( this.age != p.age ) {  
        return this.age - p.age;  
    }  
    // Alderen er identisk  
    return this.name.compareTo(p.name);  
}
```

Hvis alderen er forskellig
ordnes efter alder

Ellers ordnes
alfabetisk efter navn

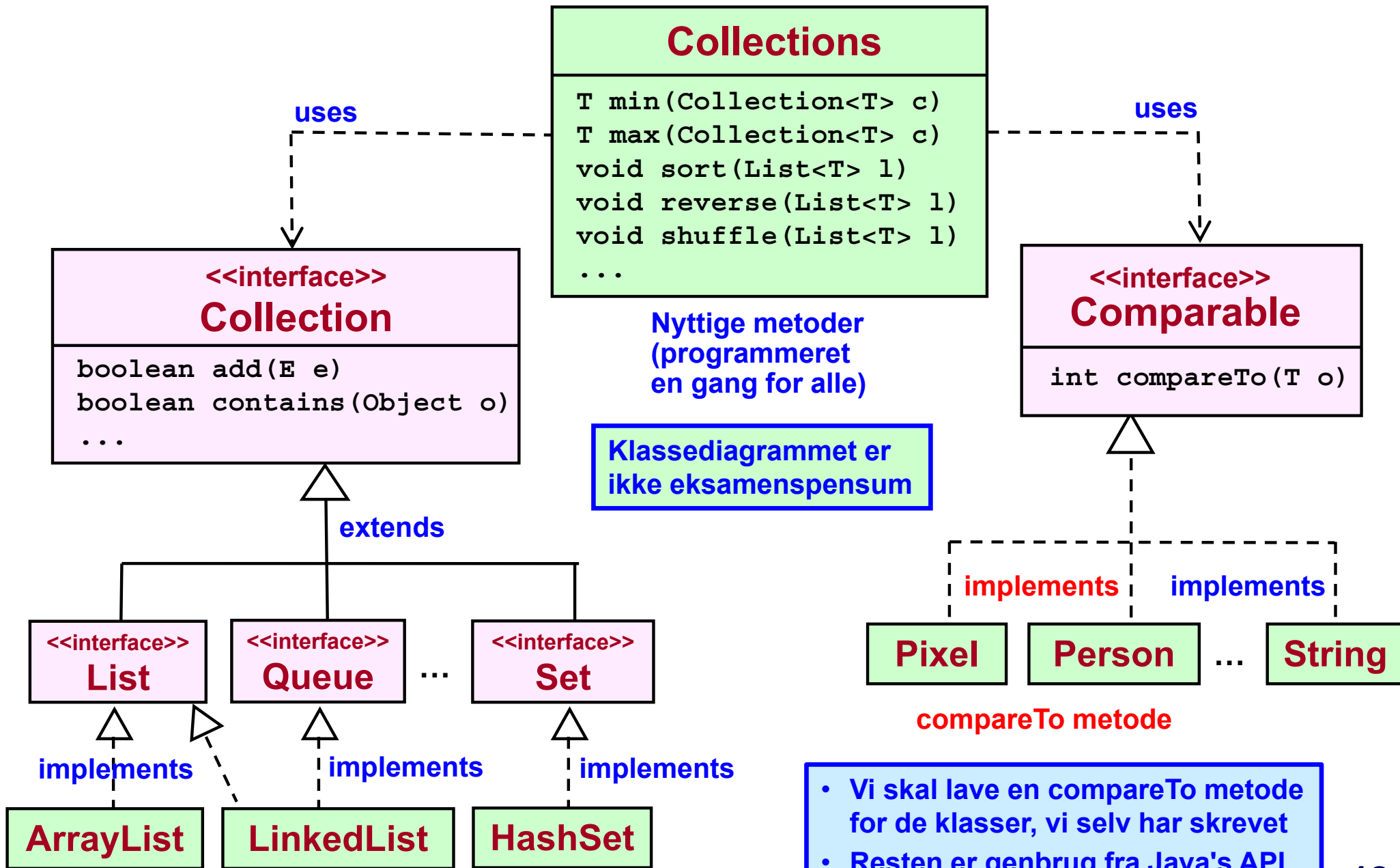
Køreprøven indeholder en
sorteringsopgave, som kan
løses ved hjælp af
Collections og Comparable



Blue: Terminal Window - interfaces-person

```
Options  
*****  
liste: [Cecilie:18, Erik:16, Adam:16, Bo:39, Dora:47]  
min: Adam:16  
max: Dora:47  
sorteret liste: [Adam:16, Erik:16, Cecilie:18, Bo:39, Dora:47]
```

Klassediagram



Ca. 30 forskellige Collection klasser

Hvad gør vi, når vi har brug for flere ordninger?

- **For personer kan vi for eksempel ønske at**
 - ordne efter alder,
 - ordne efter fornavn,
 - ordne efter efternavn,
 - kombinere nogle af ovenstående ordningskriterier
- **Comparable interfacet tillader kun én ordning ad gangen**
 - Specificeret via **compareTo** metoden
- **Comparator interfacet tillader flere ordninger ad gangen**
 - **min**, **max** og **sort** har så en ekstra parameter, der specificerer, hvilken ordning man vil bruge
 - Parameteren skal være et objekt i en klasse, der **implementerer** interfacet **Comparator**
 - Klassen indeholder en **compare** metode, der sammenligner to elementer af den type, som vi ønsker en ordning for

Comparable ligger i
pakken java.lang (som
importeres automatisk)

Collections og Comparator
ligger i pakken java.util (og
skal derfor importeres)

Pause

Brug af Comparator på ArrayList<Person>

```
public class TestDriver {  
    public static void run() {  
        ArrayList<Person> list;  
        list = new ArrayList<>();  
        list.add(new Person("Cecilie", 18));  
        list.add(new Person("Erik", 16));  
        list.add(new Person("Adam", 16));  
        list.add(new Person("Bo", 39));  
        list.add(new Person("Dora", 47));  
        print("*****");  
        print("liste: " + list);  
        print("min: " + Collections.min(list, new ByName()));  
        print("max: " + Collections.max(list, new ByName()));  
        Collections.sort(list, new ByName());  
        print("sorteret liste: " + list);  
    }  
}
```

Collections klassen har to versioner af min, max og sort

- Det ene sæt bruges sammen med Comparable interfacet
- Det andet sæt (som har en ekstra parameter) bruges sammen med Comparator

Parameterværdi

- Anonymt objekt fra klasse, der implementerer Comparator<Person>
- Klassens compare metode bestemmer, hvilken ordning, der anvendes

Ordning efter navn

- Vi laver en helt ny klasse

- Implementerer **Comparator** interfacet og dets **compare** metode
- Metodens to parametre er de to objekter, der skal sammenlignes

- $p1 < p2 \Rightarrow$ negativ
- $p1 = p2 \Rightarrow 0$
- $p1 > p2 \Rightarrow$ positiv

Ny klasse

```
import java.util.Comparator;
public class ByName implements Comparator<Person> {
    public int compare(Person p1, Person p2) {
        return p1.getName().compareTo(p2.getName());
    }
}
```

p1's navn
(tekststreng)

Metode fra
String klassen
(alfabetisk ordning)

p2's navn
(tekststreng)

- Nu må vi bruge en accessor metode for at få fat i den **private** feltvariabel **name**
- **compare** metoden ligger **ikke** i **Person** klassen, som **compareTo** gjorde

Ordning efter alder (med yngste først)

```
import java.util.Comparator;
public class ByAge implements Comparator<Person> {
    public int compare(Person p1, Person p2) {
        return p1.getAge() - p2.getAge();
    }
}
```

p1's alder

Subtraktion
(af heltal)

p2's alder

- $p1 < p2 \Rightarrow$ negativ
- $p1 = p2 \Rightarrow 0$
- $p1 > p2 \Rightarrow$ positiv

- Vi bruger en accessor metode for at få fat i den **private** feltvariabel **age**
- **compare** metoden ligger **ikke** i **Person** klassen, som **compareTo** gjorde

Ordning efter alder og navn

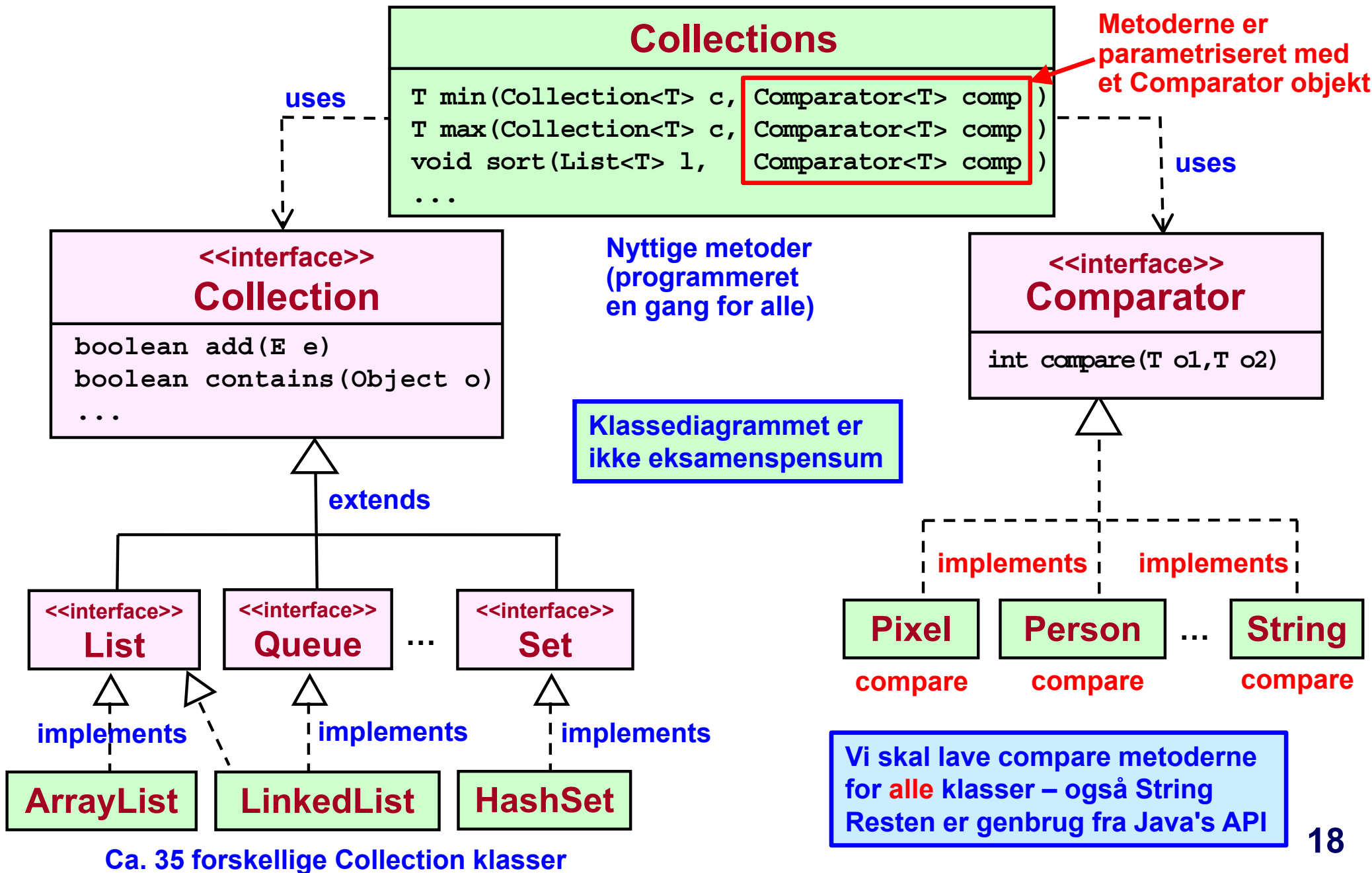
```
import java.util.Comparator;
public class ByAgeName implements Comparator<Person> {
    public int compare(Person p1, Person p2) {
        if (p1.getAge() != p2.getAge()) {
            return p1.getAge() - p2.getAge();
        }
        // Alderen er identisk
        return p1.getName().compareTo(p2.getName());
    }
}
```

```
public int compare(Person p1, Person p2) {
    if (p1.getAge() == p2.getAge()) {
        return p1.getName().compareTo(p2.getName());
    }
    else {
        return p1.getAge() - p2.getAge();
    }
}
```

Nogle negerer testet og skriver compare (og compareTo) som vist her

- Det gør det vanskeligere at generalisere til mere end to ordningskriterier
- Placerer det primære kriterie sidst (hvilket er mindre logisk)

Klassediagram for brug af Comparator



Comparable eller Comparator?

Comparable

```
public int compareTo(Person p) {  
    return this.age - p.age;  
}
```

```
public Person findOldestPerson() {  
    return Collections.max(persons);  
}
```

Simpel

- **compareTo** metoden defineres i **Person** klassen, som implementerer interfacet **Comparable**
- Man kan kun have en ordning ad gangen (naturlige ordning)

Comparator

```
public class ByAge implements Comparator<Person> {  
    public int compare (Person p1, Person p2) {  
        return p1.getAge() - p2.getAge();  
    }  
}
```

```
public Person findOldestPerson() {  
    return Collections.max(persons, new ByAge());  
}
```

Efter alder

```
public Person findFirstPerson() {  
    return Collections.min(persons, new ByName());  
}
```

Efter navn

Mere kompleks

- **compare** metoden defineres i en **ny klasse**, som implementerer interfacet **Comparator**
- **min**, **max** og **sort** metoderne har en **ekstra parameter**
- Dermed er det muligt at bruge **flere** ordninger samtidigt

Når vi ser på funktionel programmering, vil vi se, at man via Comparator kan definere en ordning, uden selv at skrive en compare (eller compareTo metode)

I køreprøven er det tilstrækkeligt at bruge Comparable

Quiz

● Algoritmeskabelonen findBest

- Gennem søger arraylisten **LIST** med elementer af typen **TYPE** og returnerer det **bedste** af de elementer, der opfylder **TEST**

```
public TYPE findBest(PARAM) {  
    TYPE result = null; ← Lokal variabel som indeholder den hidtil bedste  
    for(TYPE elem : LISTE) {  
        if(TEST(elem, PARAM) ) {  
            if(result == null || BEST(elem, result, PARAM) ) {  
                result = elem;  
            }  
        }  
    }  
    return result;  
}
```

Test for om elem er bedre end result
Husk at result == null skal stå til venstre for ||, idet vi ellers vil få en NullPointerException

- Hvis flere elementer er lige gode, returneres det først fundne
- Hvis ingen elementer opfylder **TEST**, returneres **null**
- Hvis man undlader **TEST** (og fjerner den yderste if sætning), finder man det **BEDSTE** af alle elementer i **LIST**

findBest kan også løses ved at sortere

- **Den ældste kvinde i en liste af personer kan findes på følgende måde**
 - Person objekter ordnes efter alder (ved hjælp af **compareTo** i Comparable eller **compare** i Comparator)
 - Brug **findAll** til at finde en delliste med alle kvinder
 - Brug **max** metoden til at finde den ældste kvinde i dellisten (hvis dellisten er tom returneres null)
- **Alternativt kan man erstatte de sidste to skridt med**
 - Brug **sort** metoden til at sortere Person listen efter alder (ældste først)
 - Brug **findOne** til at finde den første kvinde
- **Man kan også lave en ordning som først ordner efter køn (mænd før kvinder) og dernæst efter alder (yngste først)**
 - Brug **max** metoden til at finde den ældste kvinde i dellisten (hvis listen er tom eller det maksimale element er en mand returneres null)
- **Hvilken af de tre fremgangsmåder er bedst og hvorfor?**
 - Nr. 1 og 3 er mest effektive
 - Nr. 2 sorterer hele listen, hvilket er langt dyrere end blot at finde det maksimale element (som kan gøres i ét gennemløb)
 - findBest klarer også opgaven i ét gennemløb

● Information om køreprøven

- Køreprøven afvikles **torsdag den 9. oktober**
- Finder sted i **Institut for Datalogis studiecafé**, der ligger i Stueetagen af Vannevar Bush bygningen (bygning 5343 i IT-Parken, Åbogade 34)
- Det præcise tidspunkt for hvert øvelseshold er publiceret i en "Vigtig meddelelse" på Brightspace
- Hvis det er nødvendigt, kan du flytte til et andet prøvetidspunkt, hvis du via mail beder Kurt Jensen herom senest tre dage før prøven
- Pointene fra køreprøven tæller med til mundtlig eksamen
- Inden køreprøven skal du have afleveret **alle** afleveringsopgaver fra Uge 1-6 (inklusive quizzene) – hvis du mangler opgaver, fratrækkes 1,0 point for hver manglende opgave/quiz
- Køreprøven er en **30 minutters praktisk prøve** (uden forberedelsestid)
- Den afvikles i hold på **15-25 personer** (svarende til et øvelseshold)
- Du skal **medbringe en bærbar computer** og har selv ansvar for, at den fungerer tilfredsstillende og har **netadgang**, således at du kan tilgå Javas klassebibliotek og aflevere på Brightspace

Tjekpunkter

- Køreprøven har 12 spørgsmål, som **skal** løses i rækkefølge (hvis man f.eks. springer spørgsmål 7 over, får man intet for de efterfølgende)
- Undervejs er der seks **tjekpunkter**. Ved hvert af disse **skal** du tilkalde en **instruktor** (og være klar til at demonstrere din kode)
- Det er vigtigt, at du husker at få din kode **godkendt af en instruktor** – hver gang du passerer et **tjekpunkt**
- På den måde undgår du at forsætte uden at det, som du har lavet, er korrekt
- Derudover får vi **registreret**, at du har klaret tjekpunktet.
- Efter køreprøven ser vi kun på din kode, hvis der opstår tvivlsspørgsmål
- Lav dit program så **letlæseligt** og **velstruktureret** som muligt (og overhold Java style guiden)
- Ved køreprøven behøver du **ikke** at bruge tid på at skrive **kommentarer**
- Vi anbefaler dog, at du indsætter **forklarende tekst** i dine udskrifter, så du (og instruktorerne) kan se, hvad det er, du forsøger at skrive ud

Tilladt / forbudt

- Spørgsmål 1-10 skal løses ved hjælp af **imperativ programmering**. Man må altså ikke bruge streams og lambda'er (som introduceres i næste forelæsning)
- Spørgsmål 11-12 skal løses ved hjælp af **funktionel programmering**. De to metoder man skal skrive og afteste kan implementeres ved hjælp af de **funktionelle** algoritmeskabeloner (som introduceres i næste forelæsning)
- Eneste tilladte hjælpemidler er JavaDoc for **Javas klassebibliotek** (API) samt **BlueJ editoren** (eller en anden Java editor)
- Man må ikke **auto-generere** kode for konstruktører, accessor metoder, import sætninger og lignende (men man må godt auto-extend variabel- og metodenavne)
- Det er ikke tilladt at benytte bogen eller at tilgå andet materiale, herunder slides, noter og gamle BlueJ projekter
- Bliver man taget i dette, **bortvises** man fra prøven (og får 0 point)
- Det er normalt ikke tilladt at benytte **høretelefoner**
- Man må gerne bruge ørepropper, og ved prøvens start kan man bede om at blive placeret i et roligt hjørne af lokalet
- Personer med specielle handicaps kan søge om tilladelse til at bruge høretelefoner ved at sende en mail til Kurt Jensen senest 1 uge inden køreprøven

Andre ting

- Til stede ved prøven vil være **forelæseren** og et **antal instruktører**
- Det er **tilladt at kommunikere med disse personer** (opklarende spørgsmål, hjælp til at komme videre, etc.)
- Det er **ikke tilladt** at kommunikere med de **øvrige eksaminander**
- Ved prøvens afslutning afleveres din besvarelse på samme måde som ved de obligatoriske afleveringer i løbet af kurset, dvs. via Brightspace

Forlænget tid

- I tilfælde af ordblindhed, autisme, ADHD, og lignende har man mulighed for at få **forlænget eksamenstid**
- Det gælder også ved køreprøven, hvor man så typisk får 35 minutter i stedet for 30 min
- Ansøgning om forlænget tid (inklusiv fornøden dokumentation) sendes til Kurt Jensen med mail senest 1 uge inden køreprøven

Resultat + praktiske ting

- Man får **2 point** for **hvert tjekpunkt**, dvs. at fuld besvarelse giver 12 point
- Man kan **ikke** dumpe køreprøven, men man kan i teorien godt få 0 point (hvilket er ekstremt sjældent)
- Pointene tæller med ved fastlæggelsen af den endelige karakter for kurset
- Hvis du på grund af sygdom (eller andet) ikke kan deltage den 9. oktober, kan du ved at sende en mail til Kurt Jensen komme til en **ny køreprøve** umiddelbart efter efterårsferien
- Kom i **god tid** – senest 15 minutter før start
- I bliver lukket ind i lokalet ca. 10 minutter før start
- Husk at medbringe dit **studiekort** (eller billedlegitimation + en seddel med dit fulde navn og studienummer)

Forberedelse til køreprøven

- **Se videoer (meget vigtigt)**
 - **Imperativ** løsning af fire køreprøvesæt findes under uge 4-5 på ugeoversigten (Phone, Pirate, Car og Turtle)
 - **Funktionel** løsning af et køreprøvesæt findes under uge 6 på ugeoversigten (Penguin)
 - Husk at det ikke er nok at se videoerne. Du skal bagefter **selv** prøve at **løse opgaverne**
- **Løs tidligere køreprøvesæt**
 - Et stort udvalg (ca. 40 stk.) findes på Brightspace siden "Køreprøvesæt fra tidligere år" under "Øvelser"
 - Husk at du kan bruge testserveren til at kontrollere din besvarelse (hvilket du **skal** gøre for de sæt, der afleveres i uge 5-6)
- **Tag tid, så du kan se, hvor lang tid du er om at løse et køreprøvesæt**
 - Det er ikke unormalt, at det i begyndelsen tager halvanden time at løse et køreprøvesæt – men øvelse gør mester
- **Deltag i prøveeksamen ved den første øvelsesgang i uge 7**

● Afleveringsopgaver i uge 5 og 6

- **Tre køreprøvesæt i uge 5 (med 10 opgaver i hver)**
 - Imperativ programmering
- **Fire køreprøvesæt i uge 6 (med 12 opgaver i hver)**
 - Imperativ og funktionel programmering
- **Alle køreprøvesæt løses og afleveres individuelt**
 - Undervejs må I gerne snakke med jeres makker og hjælpe hinanden
 - Når I begge har løst en opgave, kan I gennemgå hinandens løsninger og diskutere, hvordan de kan forbedres
 - Derefter forbedrer I jeres egen løsning og afleverer den
- **Husk at kurset har nul-tolerance overfor plagiering**
 - Man må ikke **kopiere** hinandens kode
 - Hvis I bliver taget i plagiering, kommer I først til eksamen næste år
- **Husk at teste køreprøvesættene på testserveren før de afleveres**
 - Ellers får I automatisk genaflevering

● Opsummering

- **Sortering ved hjælp af klassen Collections**
 - Ordning fastlægges ved hjælp af interfacet Comparable (Naturlige ordning)
 - Ordning fastlægges af interfacet Comparator (mulighed for flere ordninger)
- **findBest som sorteringsproblem**
- **Information om køreprøven**
 - Form
 - Forberedelse

Køreprøven indeholder en sorteringsopgave, som kan løses ved hjælp af Collections og Comparable

Det var alt for nu.....

... spørgsmål

