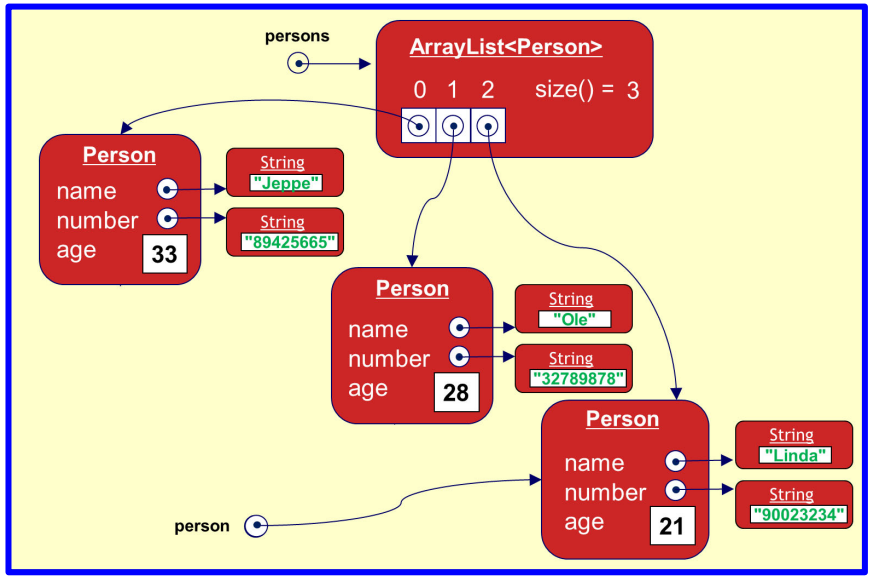
Forelæsning Uge 3 – Mandag

- **ArrayList klassen**
  - Gør det let at lave en objektsamling (collection) med et variabelt antal elementer
  - Der er mange andre slags objektsamlinger (se Collection interfacet i JavaDoc)
- **MusicOrganizer projektet**
  - Eksempel på brug af ArrayList
- **Javas for-each løkke**
  - Alternativ til for, while og do-while løkkerne
  - Velegnet til gennemløb af arraylister (og andre collections)
- **Java API**
  - Grænsefladen til Javas klassebibliotek
- **Afleveringsopgaver i uge 3**

- **På Projekt Euler, CodingBats og Kattis findes en masse opgaver, hvor I kan øve jer i Java programmering, hvis I har tid tilovers**
- **Links under Uge 3 på Ugeoversigten**

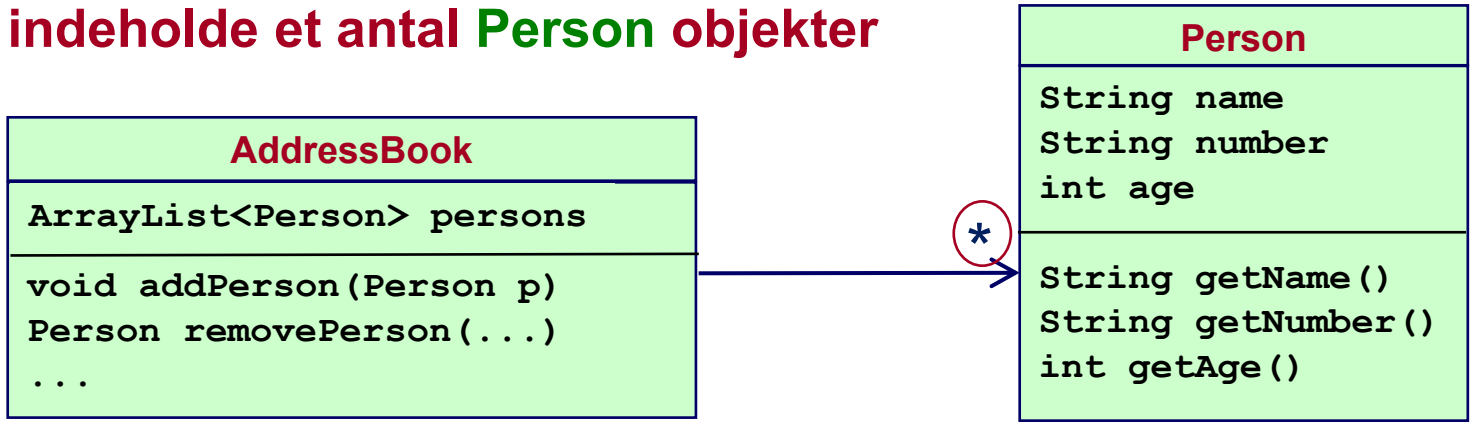


Collections – Samlinger af objekter

- **Objektreferencer**
  - For at "holde fast" i et objekt skal der bruges en objekt-reference (en variabel)
  - I Raflebæger 1 og 2 bruger vi to feltvariabler **d1** og **d2** til at ”holde fast” i hvert sit **Die** objekt.
  - Hvis man har 10 terninger i raflebægeret, skal man have 10 feltvariabler
  - For at "holde fast" i 1000 objekter skal der bruges 1000 objekt-referencer
- **Collections (objektsamlinger)**
  - En særlig slags objekter, der kan opbevare (referencer til) objekter
  - Klassen **ArrayList** kan bruges til at skabe en liste (ordnet sekvens) af objekter
  - Et **ArrayList** objekt kan f.eks. have referencer til
    - et antal **Person** objekter, eller
    - et antal **String** objekter, eller
    - et antal **Die** objekter, eller
    - et antal heltal
  - Når vi erklærer en arrayliste, specificerer vi, hvilken slags objekter, den skal kunne indeholde (pege på)

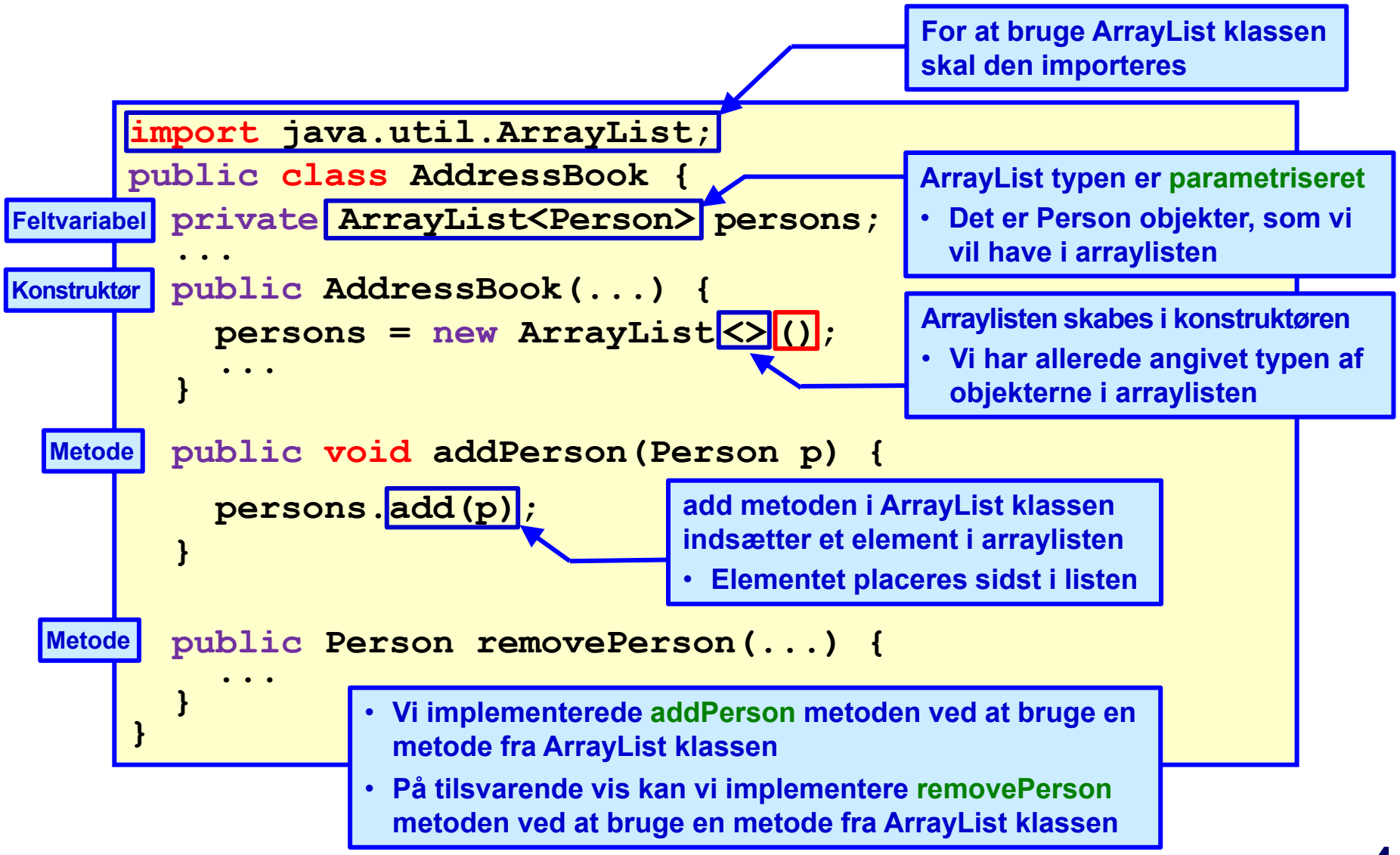
Klassediagram for adressebog

- **Vi vil lave en adressebog, der kan indeholde et antal Person objekter**



- **Hvordan realiseres den én-til-mange relationen, som stjernen angiver?**
  - Der skal kunne være et **ubegrænset** antal personer i adressebogen
  - Vi ved ikke på forhånd, hvor mange, der bliver tilføjet
  - Hvordan kan feltvariablen **persons** have referencer til alle **Person** objekterne?
- **Det kan vi gøre ved hjælp af en arrayliste**
  - Arraylister minder på mange måder om arrays, som nogle af jer kender fra andre programmeringssprog
  - Syntaksen er anderledes
  - Man behøver ikke at tænke på, hvor mange elementer der skal være i listen

Implementation af adressebogen



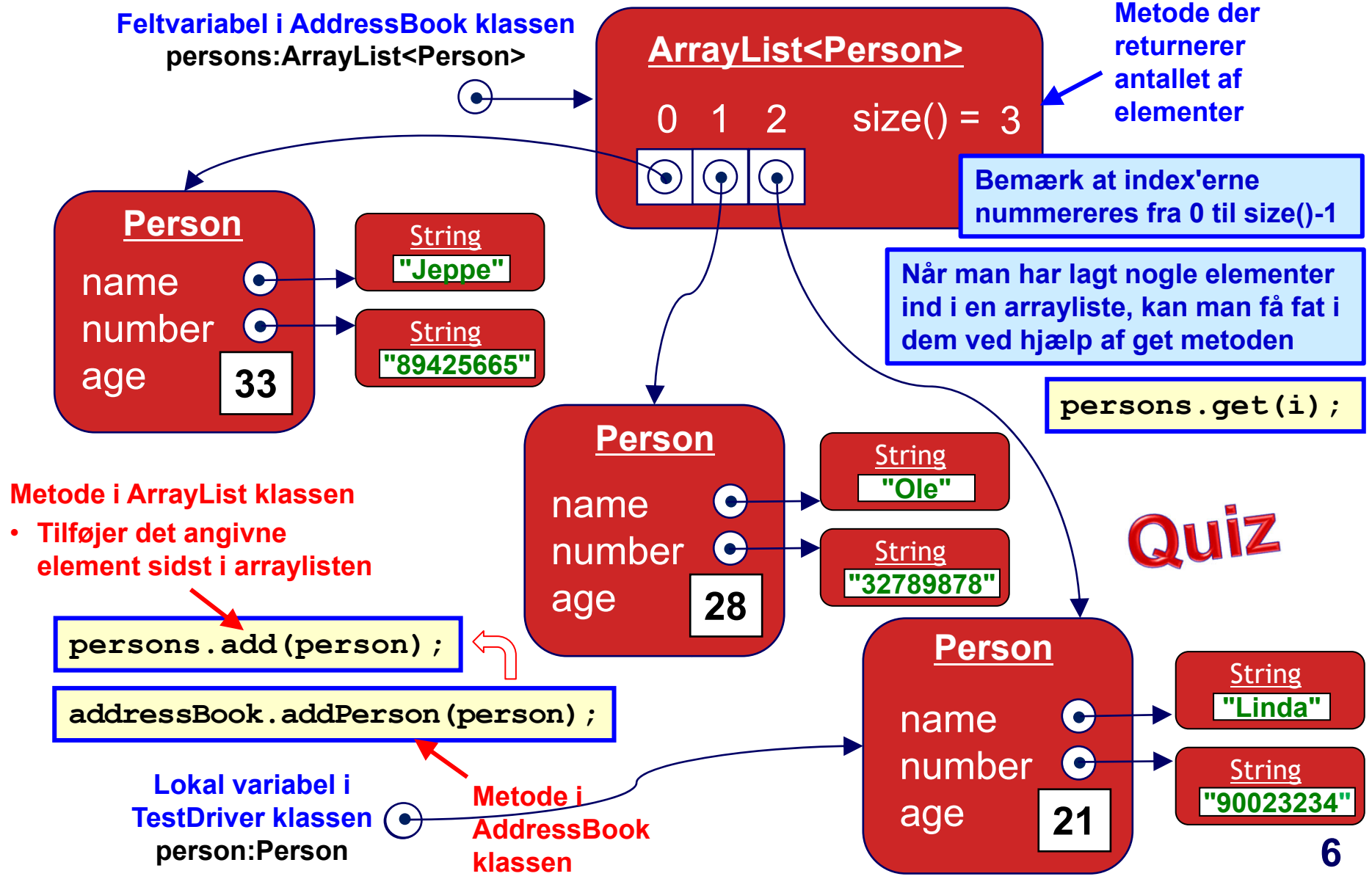
TestDriver klasse

```
import java.util.ArrayList;
public class TestDriver {
  public static void runTest() {
    AddressBook addressBook = new AddressBook(...);
    Person person;

    person = new Person("Jeppe", "89425665", 33);
    addressBook.addPerson(person);

    person = new Person("Ole", "32789878", 28);
    addressBook.addPerson(person);

    person = new Person("Linda", "90023234", 21);
    addressBook.addPerson(person);
  }
}
```

Lokale variabler

Metode i AddressBook klassen
- Kalder add metoden i ArrayList klassen

Objektdiagram for adressebogen



Realisering af *en-til-mange* relation – Java

- **For at realisere en *en-til-mange* relation i koden skal man gøre 3 ting**

IM

1. **Importere** ArrayList klassen (fra java.util pakken)

```
import java.util.ArrayList;
```

ER

2. **Erklære** en feltvariabel af typen ArrayList<…>

```
private ArrayList<Person> persons;
```

CO

3. Initialisere feltvariablen (gøres normalt i **konstruktøren**)

```
public AddressBook() {
  persons = new ArrayList<>();
}
```

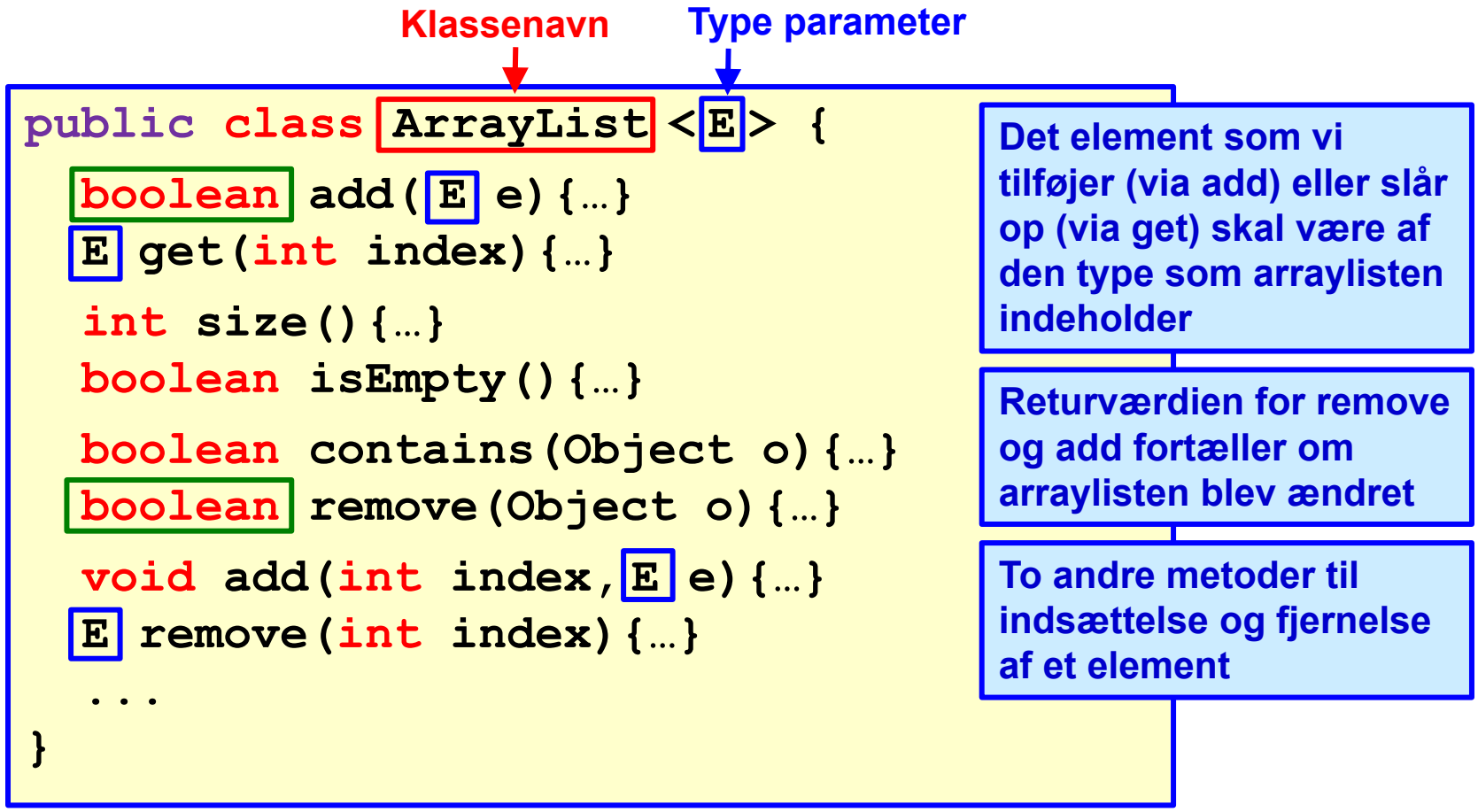
Vi behøver ikke at gentage type parameteren til ArrayList

Husk de runde parenteser
- **Kald af konstruktør**
- **Ellers syntaksfejl**

IMERCO

ArrayList er en parametriseret type

- Dokumentationen for ArrayList klassen fortæller, at der bl.a. er nedenstående metoder



Flere detaljer: se JavaDoc... Link

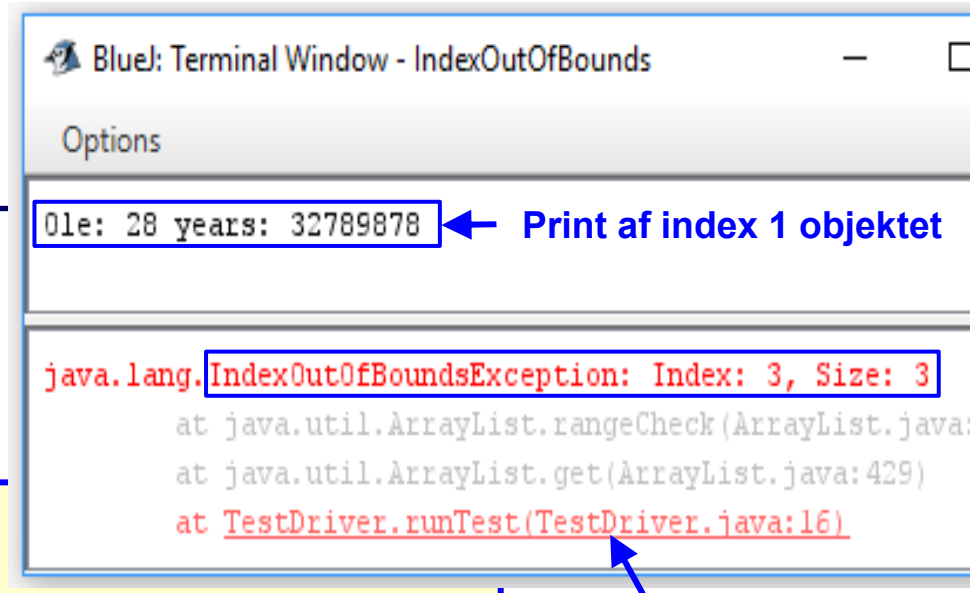
Arrayliste med heltal

- **Parameteren til ArrayList skal være en objekt type**
  - Det betyder, at man ikke kan skrive ArrayList<int>
  - I stedet skal man skrive ArrayList<Integer>
- **Integer er en objekt type med de "samme værdier" som den primitive type int**
  - Integer er en **wrapper klasse** for int (wrapper = indpakning)
  - Integer værdier konverteres automatisk til int værdier (og omvendt), når der er behov for det
- **Eksempel**

```
private int i;
private ArrayList<Integer> list;
...
list.add(i);      // int → Integer
i = list.get(3); // Integer → int
```

Runtime exceptions

- **Husk at indices begynder ved 0 og slutter ved size()-1**




```
import java.util.ArrayList;
public class TestDriver {
  public static void runTest(){
    ArrayList<Person> list = new ArrayList<>();
    list.add(new Person("Jeppe", "89425665", 33));
    list.add(new Person("Ole", "32789878", 28));
    list.add(new Person("Linda", "90023234", 21));
    System.out.println(list.get(1));
    System.out.println(list.get(3));
    System.out.println(list.get(2));
  }
}
```

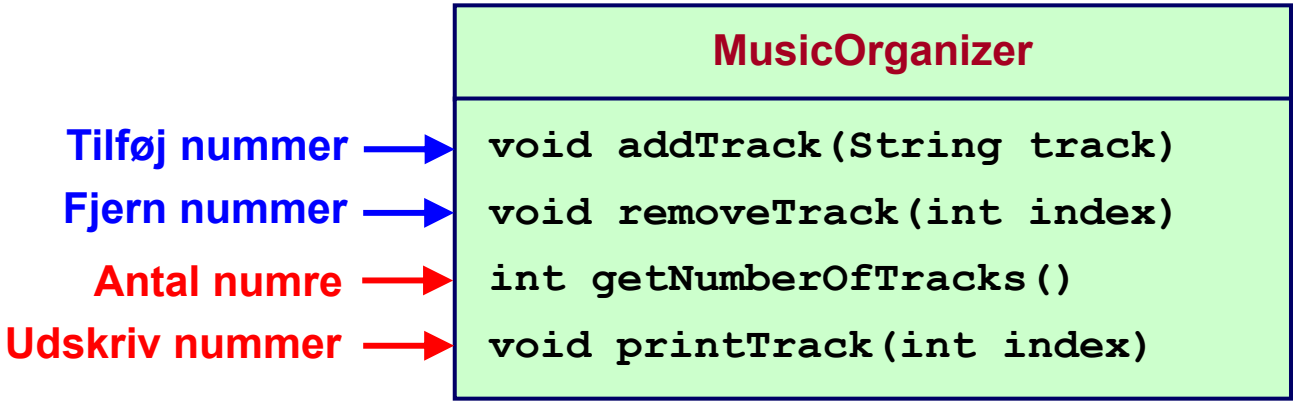
Pause

get metoden i ArrayList klassen returnerer det element, som arraylisten har på det specificerede index

Der er mange andre typer exceptions
- NullPointerException
- ArithmeticException: / by zero

MusicOrganizer – brug af ArrayList

- **Vi vil lave en klasse som kan holde styr på vores musiknumre**
  - Klassen minder lidt om musikafspilleren fra BlueJ bogens kapitel 4, men den gør nogle lidt andre ting (kartotek over musik – ingen aktiv afspilning)
  - I første version repræsenteres hvert musiknummer ved hjælp af en tekststreng (String)
  - Senere skal vi indføre en Track klasse til at repræsentere musiknumre



Oprettelse af arrayliste

1. **Importere**
   ArrayList klassen

IM

2. **Erklære**
   en feltvariabel af type ArrayList<…>

ER

3. Initialisere feltvariablen i **konstruktøren**

CO

```
import java.util.ArrayList;
/**
 * A class to hold a number of tracks.
 */
public class MusicOrganizer {
  // An ArrayList for storing the tracks.
  private ArrayList<String> tracks;

  /**
   * Creates a MusicOrganizer object.
   */
  public MusicOrganizer() {
    tracks = new ArrayList<>();
  }
  ...
  // Methods omitted.
  ...
}
```

IMERCO reglen

Tilføjelse og fjernelse af musiknumre

```
/**
 * Adds a track to the collection.
 * @param track   Track to be added.
 */
public void addTrack(String track) {
   tracks.add(track);
}
```

Metode i ArrayList klassen

- Tilføjer det angivne element sidst i arraylisten

Arraylisten klassen stiller forskellige metoder til rådighed

- Ved at benytte dem, sparer vi en masse arbejde

```
/**
 * Removes a track from the collection.
 * @param index   Index of the track to be removed.
 */
public void removeTrack(int index) {
   if(0 <= index && index < tracks.size()) {
      tracks.remove(index);
   }
}
```

Metode i ArrayList klassen

- Elementet på det angivne index fjernes fra arraylisten
- Efterfølgende elementer forskydes et "hak" mod venstre (til foregående index)

Inden vi kalder remove metoden, tester vi, om indexet er i brug (så vi undgår at få en IndexOutOfBoundsException)

Hvad bør vi gøre hvis index'et er ulovligt?

- I vores eksempel gør vi ingenting
- Det ville være pænere, at rapportere en fejl via en fejlmeddelelse på terminalen (eller ved at rejse en exception)

Antal numre og udskrivning

```
/**
 * Returns the number of tracks in the collection.
 * @return   Number of tracks in the collection.
 */
public int getNumberOfTracks() {
  return tracks.size();
}
```

Metode i ArrayList klassen

- Returnerer størrelsen af arraylisten

Arraylisten klassen stiller forskellige metoder til rådighed

- Ved at benytte dem sparer vi en masse arbejde

```
/**
 * Prints a track from the collection.
 * @param index   Index of the track to be printed.
 */
public void printTrack(int index) {
  if(0 <= index && index < tracks.size()) {
    System.out.println(tracks.get(index));
  }
}
```

Udskrift

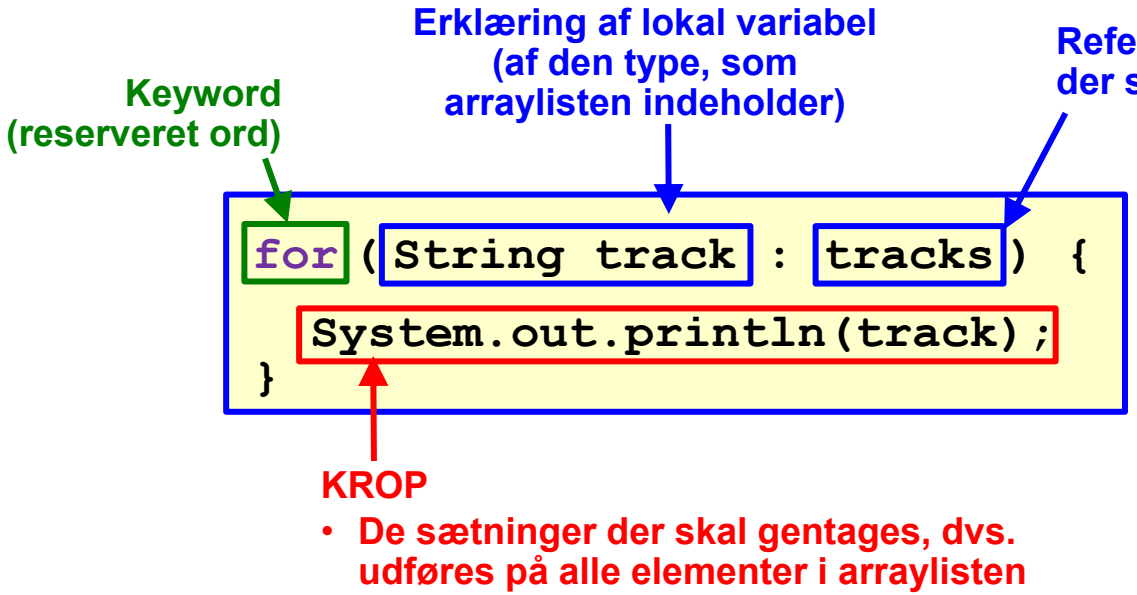
Metode i ArrayList klassen

- Returnerer elementet på det angivne index

Inden vi kalder get metoden tester vi om index'et er i brug (så vi undgår at få en IndexOutOfBoundsException)

Javas for-each løkke (udvidet for løkke)

- Bruges til at gennemløbe alle elementer i en Arrayliste og gøre "et eller andet" ved dem
- Kan bruges på alle collections



- Den lokale variabel sættes (efter tur) til at pege på de enkelte objekter i arraylisten (startende ved index 0 og sluttende ved index size()-1)
- For hvert objekt udføres de sætninger, der er indeholdt i kroppen
- I vores eksempel udskrives objekterne på terminalen ved hjælp af println metoden

Find gennemsnitsalder i adressebog

```
/**
 * Returns the average age of the
 * persons in the address book.
 */
public double averageAge() {
   double sum = 0;
   for(Person person : persons) {
      sum += person.getAge();
   }
   return sum / persons.size();
}
```

Den lokale variabel sum erklæres til at være en double (for at undgå afrunding ved division)

- person er en lokal variabel af type Person (kontrolvariabel)
- persons er den arrayliste som vi vil gennemløbe

Angiver at værdien af udtrykket på højresiden (personens alder) lægges til variablen på venstresiden

Udskrift af arrayliste

- **Alle klasser er subklasser af klassen Object**
  - Indeholder en metode som returnerer en tekstrepræsentation af det pågældende objekt

```
String toString(){…}
```

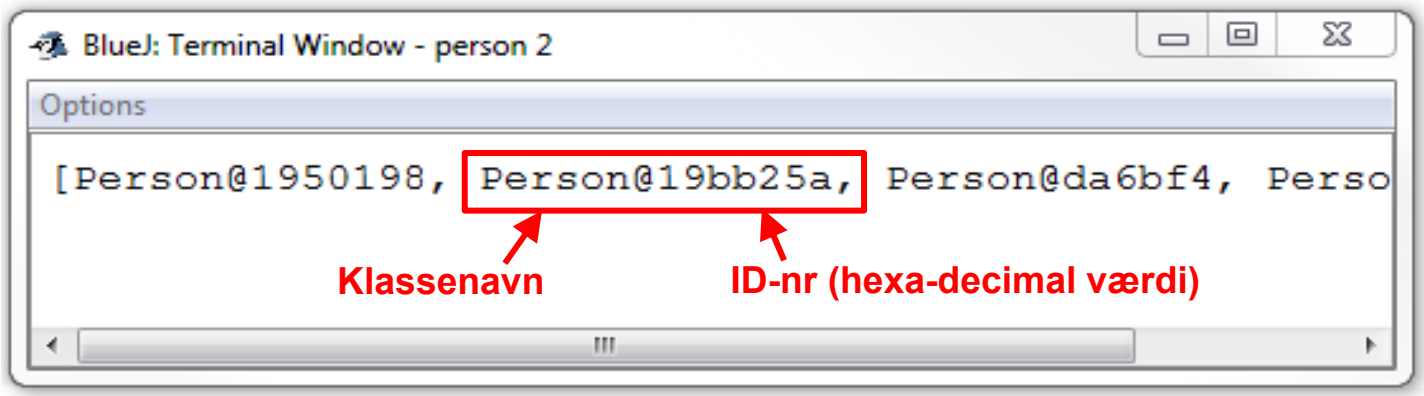
- **Et vilkårligt objekt o kan udskrives ved hjælp af sætningen**

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

  - **println** metoden kalder automatisk **toString** metoden (medmindre argumentet **o** allerede er af typen **String**)

- **Arraylisten persons kan udskrives ved hjælp af sætningen**

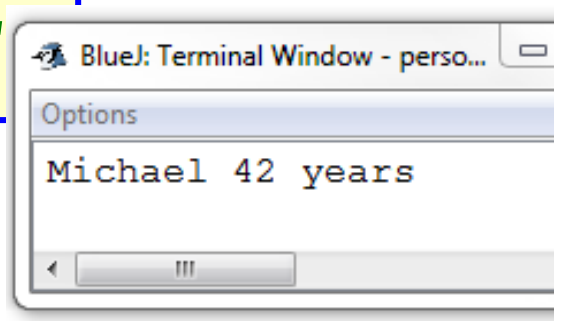
```
System.out.println(persons);
```



Pænere udskrift af arrayliste

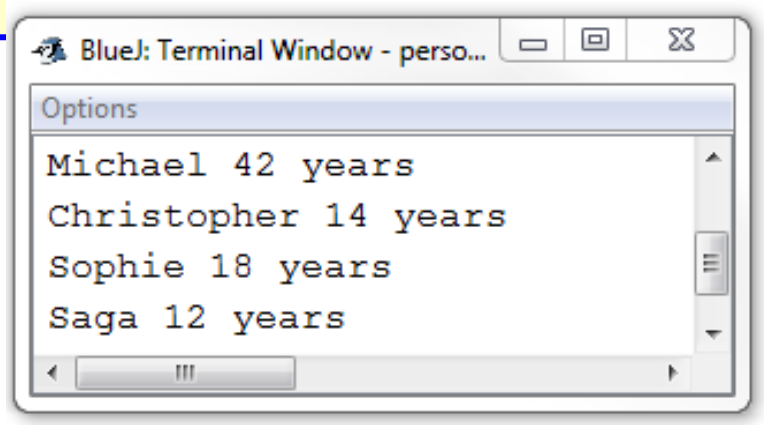
- **I Person klassen defineres en toString metoden, der returnere noget meningsfyldt (i stedet for klassenavn og hexa-decimalt ID-nr)**
  - Den nye toString metode **overskriver** (erstatter) toString fra Object klassen

```
public String toString() {
   return name + " " + age + " years";
}
```



- **Ved hjælp af en for-each løkke udskrives hvert Person objekt på en separat linje**

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



Quiz

Java API (Java's klassebibliotek)

- **Hvis vi vil have mere info om ArrayList klassen, kan vi konsultere Java API'en som beskriver grænsefladen til Java's klassebibliotek**
  - Oversigt over alle klasser (og interfaces) i Java Library
  - API = Application Programming Interface Link

API er en **softwaregrænseflade**, der tillader et stykke software at interagere med andet software.

Et typisk eksempel er at applikationer "taler" med styresystemet for at åbne en fil, hvorefter styresystemet på programmets vegne indlæser filen fra en harddisk eller lignende.

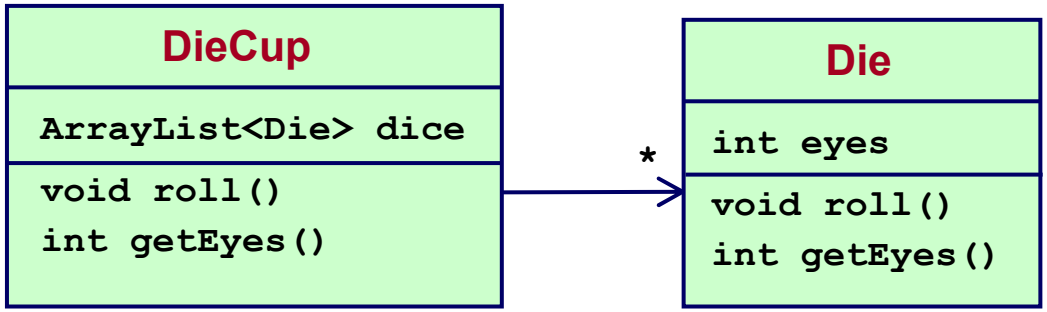
I en nøddeskal kan et API betegnes som en måde at **tilbyde tjenester**, herunder data, fra et system til et andet system.

Demo

- **Math og String klassen i pakken java.lang**
- **ArrayList og Random klassen i pakken java.util**

Afleveringsopgave: Raflebæger 3 (DieCup 3)

- **I skal endnu en gang arbejde videre med jeres raflebæger**
- **I skal først lave et raflebæger, som kan indeholde et vilkårligt (positivt) antal terninger (som alle har 6 sider)**
  - I skal "huske" terningerne ved hjælp af en feltvariabel, der er en arrayliste (som jo kan indeholde et vilkårligt antal objekter)



  - Derudover skal I ændre konstruktøren for DieCup klasen, så den får en parameter, der angiver antallet af terninger

```
// skaber raflebæger med n terninger
DieCup(int n){…}
```

  - Endelig skal I tilpasse metoderne i TestDriver klassen, således at de kan anvendes til raflebægre af ovenstående slags

Raflebæger 3 (DieCup 3) – fortsat

- **Dernæst skal I lave et raflebæger, som kan indeholde et vilkårligt (positivt) antal terninger, som hver har et vilkårligt antal sider (≥ 2)**
  - Bemærk at vi nu kan have et raflebæger, hvori vi har terninger med forskellige antal sider
  - For at håndtere dette, skal I ændre konstruktøren for DieCup klassen, så antallet af terninger og antallet af deres sider kan specificeres
  - Dette kan gøres ved hjælp af en arrayliste af heltal

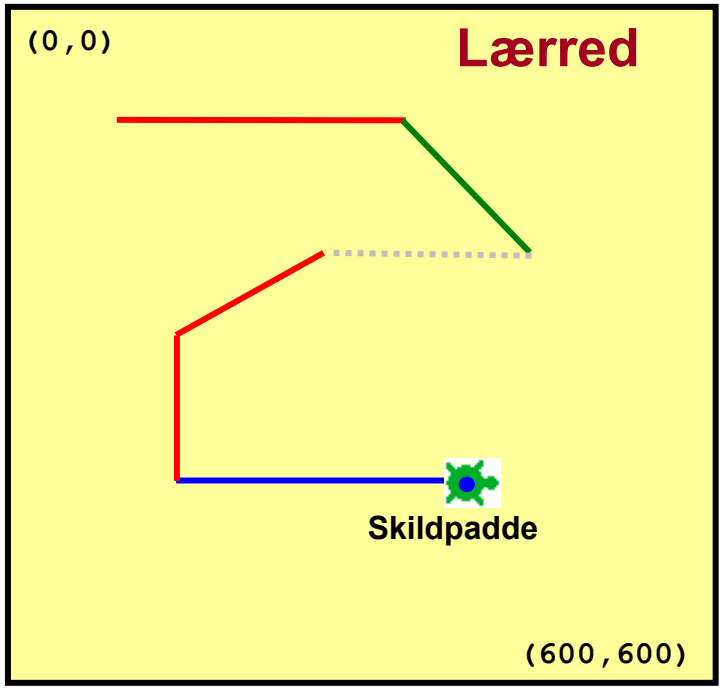
```
// Skaber raflebæger med de specificerede terninger
DieCup(ArrayList<Integer> dice){…}
```

```
[6,8,5,6]
```

  - Endelig skal I tilpasse metoderne i TestDriver klassen, således at de kan anvendes til raflebægre af ovenstående slags

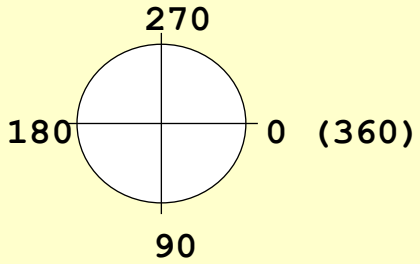
Afleveringsopgave: Skildpadde 1 (Turtle 1)

- Vi vender tilbage til skildpadden fra en tidligere forelæsning



Skildpaddens tilstand

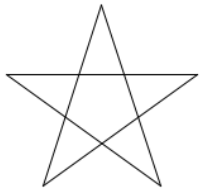
- Position: (x,y)
- Vinkel:

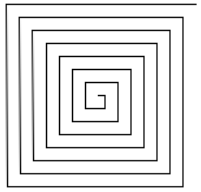


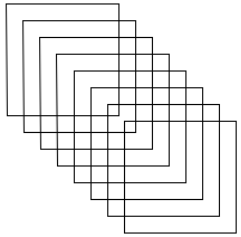
- Farve::

- Pen status: up/down

- I skal lave metoder til at tegne en række forskellige figurer, bl.a. disse:







Brug af testserveren – uddrag af logfil

8.9 kl. 14:30		Raflebæger 2 (DC2)	✓	link
8.9 kl. 13:54		Raflebæger 2 (DC2)	✓	link
8.9 kl. 13:51		Raflebæger 2 (DC2)	✓	link
8.9 kl. 13:44	4 kørsler på 10 min	Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:43		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:41		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:32		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:24		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:24	6 kørsler på 15 min	Raflebæger 1 (DC1)	⛔	link
8.9 kl. 13:18		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:17		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 13:17		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 12:09		Raflebæger 2 (DC2)	✓	link
8.9 kl. 12:08		Raflebæger 2 (DC2)	✓	link
8.9 kl. 12:08		Raflebæger 2 (DC2)	✓	link
8.9 kl. 12:08	7 kørsler på 3 min	Raflebæger 2 (DC2)	✓	link
8.9 kl. 12:07		Raflebæger 2 (DC2)	✓	link
8.9 kl. 12:06		Raflebæger 1 (DC1)	⛔	link
8.9 kl. 12:06		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:55		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:54	4 kørsler på 2 min	Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:54		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:53		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:34		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:33		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:32		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:32		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:28		Raflebæger 2 (DC2)	✓	link
8.9 kl. 11:25		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:24		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:20		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:19	6 kørsler på 12 min	Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:17		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:17		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:12		Raflebæger 2 (DC2)	⛔	link
8.9 kl. 11:06		Raflebæger 2 (DC2)	⛔	link

36 kørsler på 3 1/2 time

Brug af testserveren (fortsat)

- **Det giver ikke mening at lave masser af testkørsler inden for ganske få minutter**
  - Man kan jo slet ikke nå at studere testrapporterne, før næste testkørsel sættes i gang
  - Endsige tænke sig om og finde ud af, hvad der er galt, og hvad der bør rettes inden næste testkørsel
- **Ved at arbejde på ovenstående måde opnår man kun**
  - at spilde sin egen tid (ved at vente på masser af testrapporter)
  - at genere sine medstuderende, idet testserveren belastes helt unødvendigt, hvilket (for de mere komplekse afleveringsopgaver) kan give nedbrud og forøget ventetid
- **Antal testkørsler**
  - Indtil nu er der (i løbet af de første to uger) lavet ca. 1.500 testkørsler, hvilket giver et gennemsnit på 10 per student (eller 20 per par)
  - Men testkørslerne er meget ujævnt fordelt – nogle studerende har allerede mere end 50 testkørsler Link

Opsummering

- **ArrayList (eksempel på en Collection type)**
  - Kan bruges til at realisere én-til-mange relationer
  - Har et variabelt (ubegrænset) antal elementer
- **MusicOrganizer projektet**
  - Eksempel på brug af ArrayList
- **Javas for-each løkke**
  - Alternativ til for, while og do-while løkkerne
  - Velgenet til gennemløb af arraylister (og andre collections)

```
for(Person person : persons) {...}
```

- **Java API (grænsefladen til Javas klassebibliotek)**
- **Afleveringsopgaver i uge 3**

Hvor kan du få hjælp?

- **På siden https://studerende.au.dk/styrkditstudieliv/hjaelp/ finder du en oversigt over, hvor du som studerende har mulighed for at få støtte**
  - Linket kan også findes på kursets Brightspace side under ”Info om kurset”
- **Eksempler på hvad du kan få hjælp til og hvornår**
  - Har du en psykisk lidelse, en opmærksomhedsforstyrrelse, et fysisk handicap eller læse-, skrive-, regnevanskeligheder, kan du få hjælp hos Specialpædagogisk Støtte. Skriv til sps@au.dk
  - Har du psykiske udfordringer, oplever du eksamensangst, ensomhed eller symptomer på stress, kan du få hjælp hos studenterrådgivningen. Ring på 70 26 75 00 alle hverdage mellem kl. 9 - 12
  - Har du brug for hjælp til at forbedre din generelle trivsel som studerende, kan du få hjælp hos studie-og trivselsvejlederne. Skriv til Studievejledning.nat-tech@au.dk

Programmeringspar

- **Deltag i arbejdet i dit programmeringspar**
  - Pas på med, at du ikke bare lader makkeren lave hovedparten af arbejdet i jeres programmeringspar
  - Det er jo nemt og bekvemt, men det får du ingen programmeringsrutine af
  - Så går det galt, når du i uge 5-7 skal til helt alene at løse køreprøvesættene
- **Vi ser hvert år studerende stoppe her**
  - Jeg er overbevist om, at det for de fleste skyldes, at de har været ”sleeping partners” de første fire uger
  - Så lad være med det
- **Der er også nogle par, der deler afleveringsopgaverne imellem sig, således at de laver halvdelen hver**
  - Det er en rigtig dårlig idé
  - Man sparer noget tid, men får kun den halve programmeringserfaring
  - Det gør, at man får det svært, når man kommer til køreprøvesættene og de lidt mere komplicerede opgaver

Programmeringspar (fortsat)

- **Har du mistet din makker eller er din makker inaktiv?**
  - Hvis du mister din makker eller hvis makkeren bliver inaktiv, bør du hurtigst muligt snakke med din instruktor om problemet
  - I nogle tilfælde kan instruktoren finde en anden makker til dig
  - Ofte vil den bedste løsning dog være at forsætte alene i uge 3 og 4, hvor opgaverne stadig er forholdsvis små og overkommelige
  - I uge 5 og 6 er afleveringerne individuelle (køreprøvesæt)
  - Efter efterårsferien reviderer vi parrene – i den udstrækning, der er behov for det

Det var alt for nu….. … spørgsmål