Java

2023 – 2024 : les 2

jeroen.devos01@ap.be

AP HOGESCHOOL ANTWERPEN

AP.BE

Lesdoelen

- Wat is Maven
- Generics
- Collections (List)
- Array
- Equals & hashcode
- Comparable
- Sorting
- Loops


Apache
Maven™

Wat is Maven?

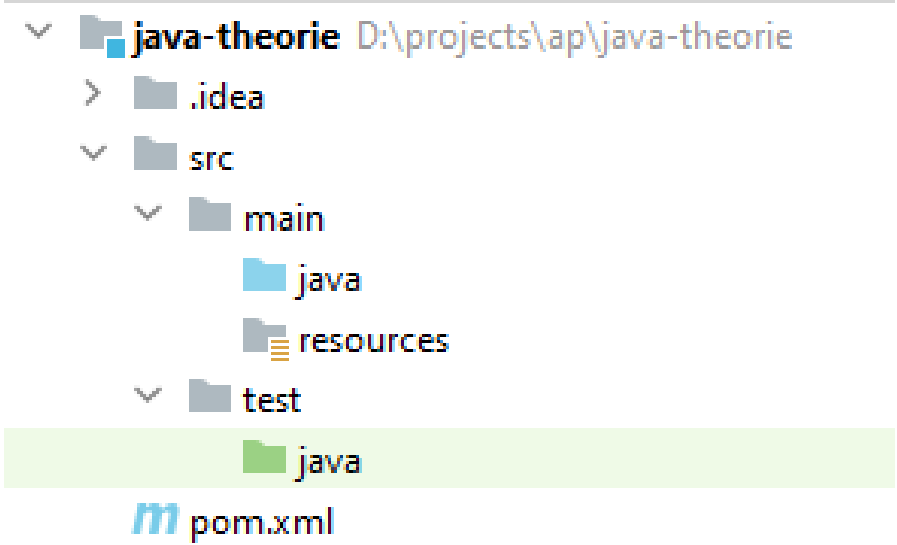
- Tool die het onderhoud van een Java project makkelijker maakt.

- Open Source , onderdeel van Apache

- Belangrijkste onderdelen:
  - ‘dwingt’ een standaard project layout af
  - beschrijft een Java project dmv Project Object Model (POM)

Maven standaard project layout

- src : bevat alle code
- src.main : productie
- src.test : test
- src.main.java : productie **code**
- src.main.resources : bestanden
- src.test.java : test **code**
- pom.xml : project beschrijving



Maven POM

pom.xml (java-theorie)

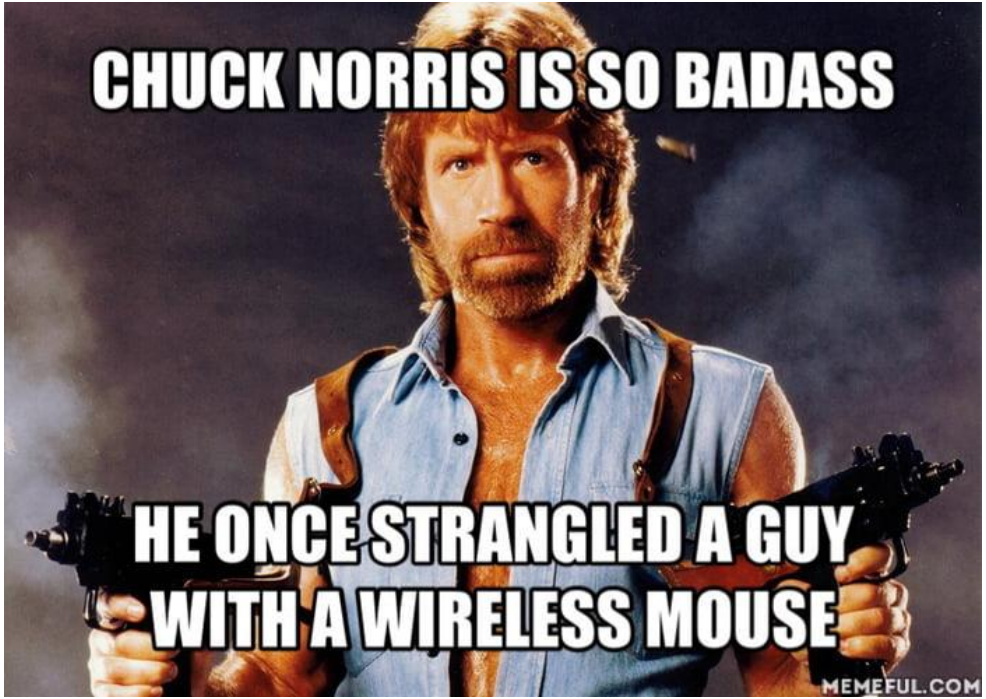
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>be.ap.student</groupId>
    <artifactId>java-theorie</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

</project>
```

Maven POM

- **groupId** : unieke sleutel die de maker van het project identificeert
- **artifactId** : unieke naam van het “artifact” dat door het project gemaakt wordt
- **version** : de versienummer van het artifact
- **properties** : eigenschappen van het project

pom.xml (java-theorie)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http:/
    <modelVersion>4.0.0</modelVersion>

    <groupId>be.ap.student</groupId>
    <artifactId>java-theorie</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncc
    </properties>

</project>
```


CHUCK NORRIS IS SO BADASS
HE ONCE STRANGLED A GUY WITH A WIRELESS MOUSE
MEMEFUL.COM


WHO WINS IN A FIGHT BETWEEN KING KONG AND GODZILLA?
VS


NEITHER, CHUCK NORRIS NEVER LETS HIS PETS FIGHT
imgflip.com

Chuck Norris joke generator

Requirement : elke dag bij opstart een random Chuck Norris joke

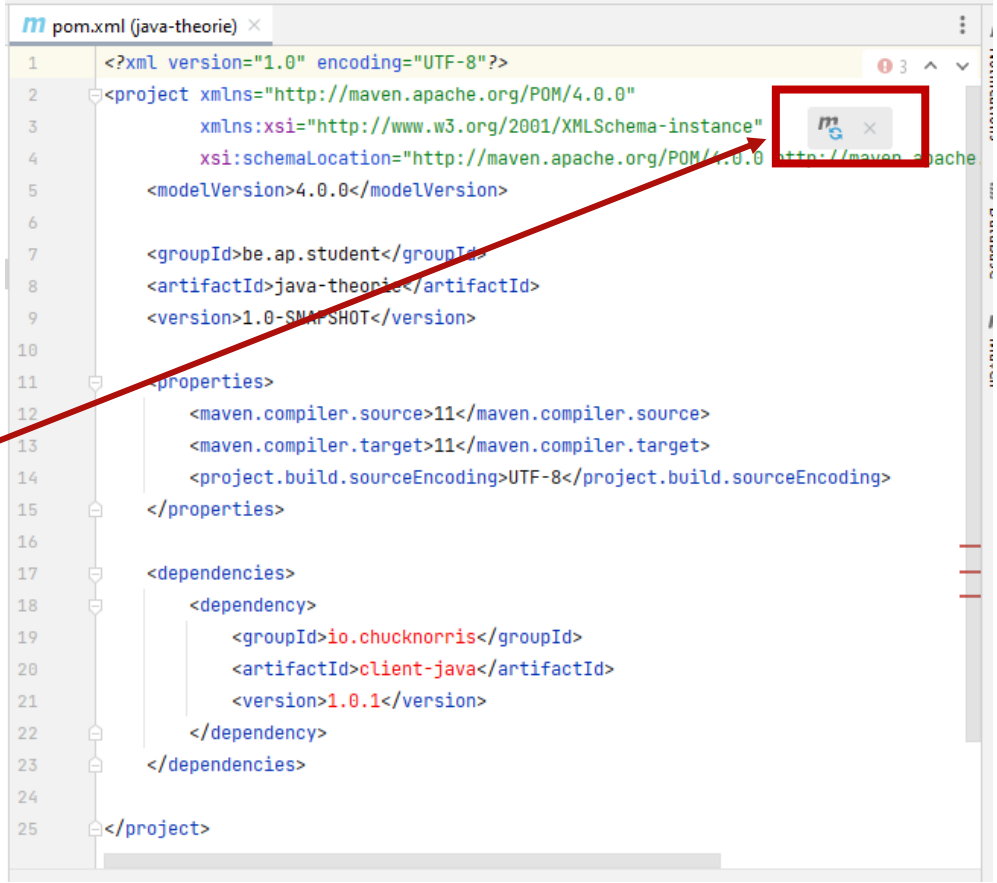
https://api.chucknorris.io/

https://github.com/chucknorris-io/client-java

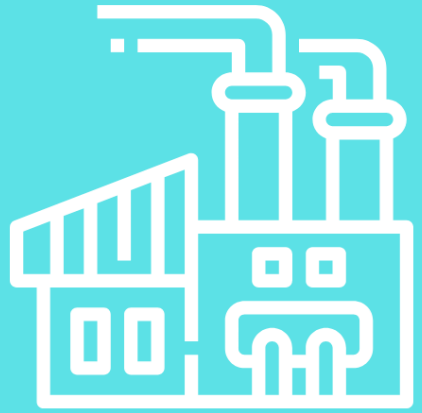
Maven dependencies

- Tag : dependencies
- Weet je nog…
  - groupId
  - artifactId
  - version

- “Load maven changes”



Generics



Java™

robust, type-safe, and reusable

Generics - probleemstelling

```
List list = new ArrayList();
list.add(10);
list.add("10");
```

```
List list = new ArrayList();
list.add(10);
Integer i = list.get(0);
String s = list.get(0);
```

Beide situaties leveren een probleem op:

1. Lijst dwingt geen datatype af, je kan perfect Integer en String in dezelfde lijst steken.
2. Bij opvragen van element uit de lijst is het datatype onbekend, de ontwikkelaar kiest een datatype uit.

Generics lost het op

```
List<Integer> list = new ArrayList<>();
list.add(10);
list.add("10"); //compile error
```

```
List<Integer> list = new ArrayList<>();
list.add(10);
Integer i = list.get(0);
String s = list.get(0); //compile error
```

Door de introductie van generics wordt het datatype afgedwongen waardoor de ontwikkelaar geen fouten meer kan maken.

Zelf generics gebruiken

1. Maak een nieuw class aan en definieer een generic <T>
2. Gebruik in de class T als datatype
3. Bij aanmaak van de class geef je het datatype mee

```
public class Ship<T> {
   private T pilot;
   public void addPilot(T pilot) {
      this.pilot = pilot;
   }
   public void showPilot() {
      System.out.println(pilot);
   }
}

Ship<LukeSkywalker> xwing = new Ship<>();
xwing.addPilot(new LukeSkywalker());
xwing.showPilot();

Ship<DarthVader> tiefighter = new Ship<>();
tiefighter.addPilot(new DarthVader());
tiefighter.showPilot();
```

Zelf generics gebruiken - extends

```
Ship<TheEmpireStrikesBack> wrong = new Ship<>();
```

Hoe dwingen we af dat enkel bepaalde datatypes mogen gebruikt worden?

Zelf generics gebruiken - extends

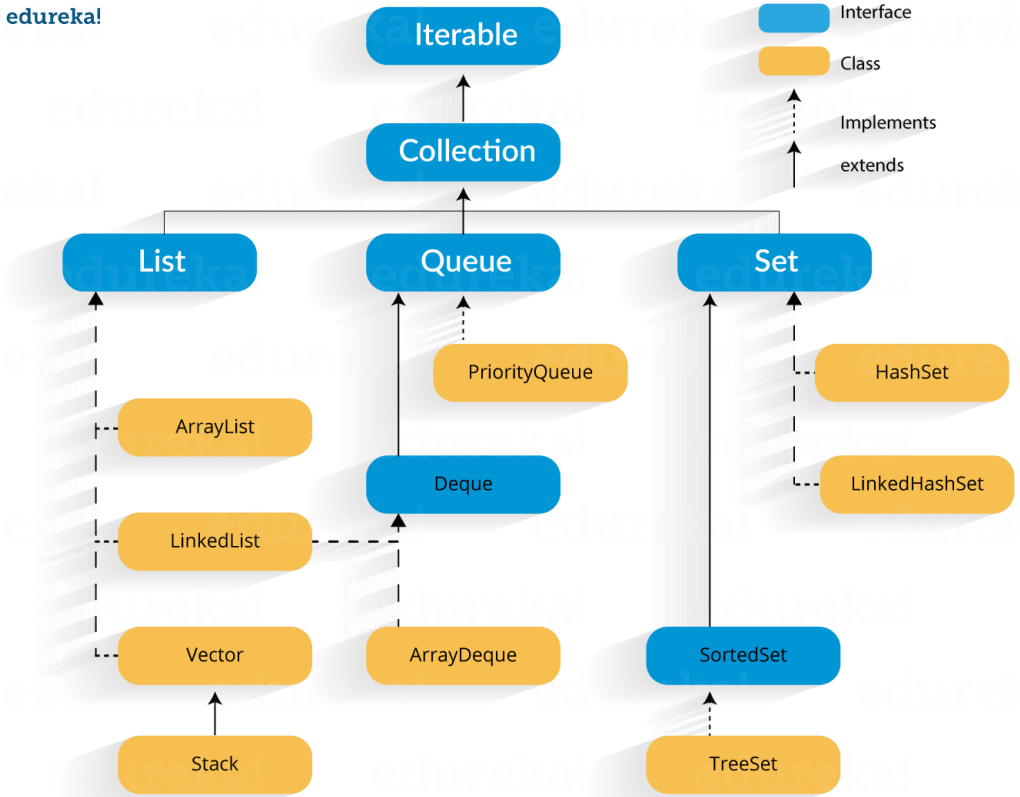
```
Ship<TheEmpireStrikesBack> wrong = new Ship<>();
```

Hoe dwingen we af dat enkel bepaalde datatypes mogen gebruikt worden?

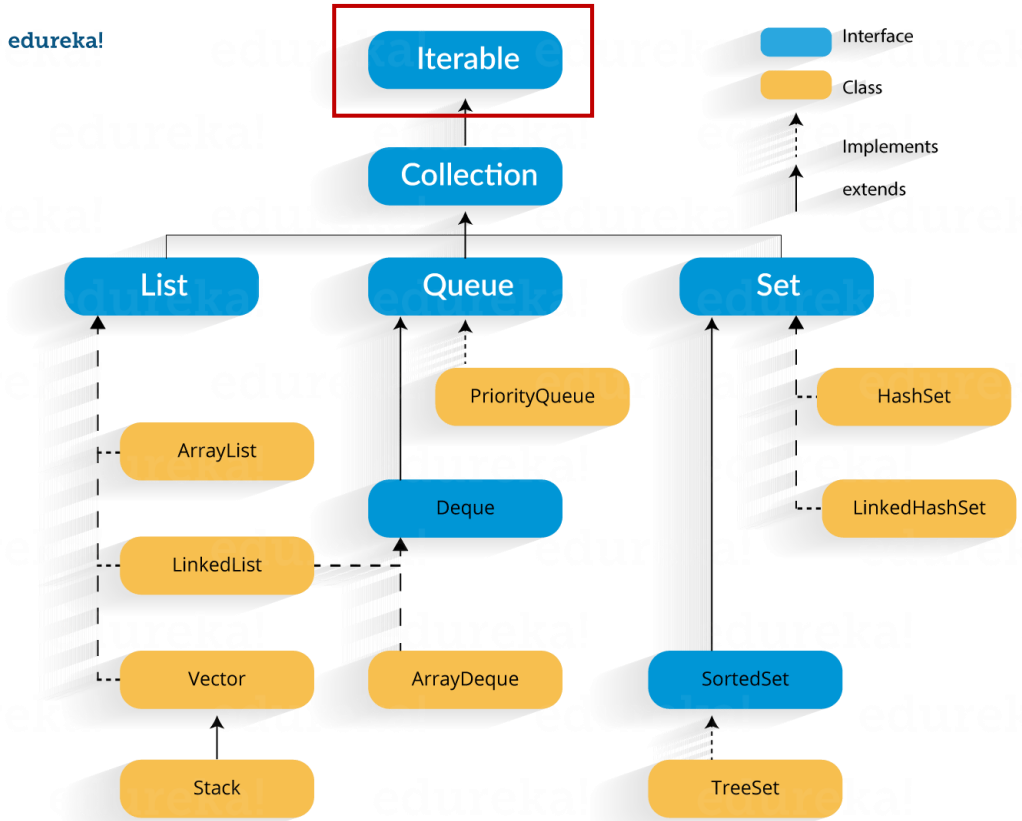
```
public class Ship<T extends StarWarsCharacter> {

}
```

Oplossing : laat het generic datatype overerven van een superclass.

Collections



Collections



Iterable interface

```
public interface Iterable<T>
```

- Basis van alle collections
- Data type via een generic
- Voorziet twee belangrijke methods:
  - Maak een Iterator aan
  - Voer een for each loop uit
- Geen implementatie want is een interface !

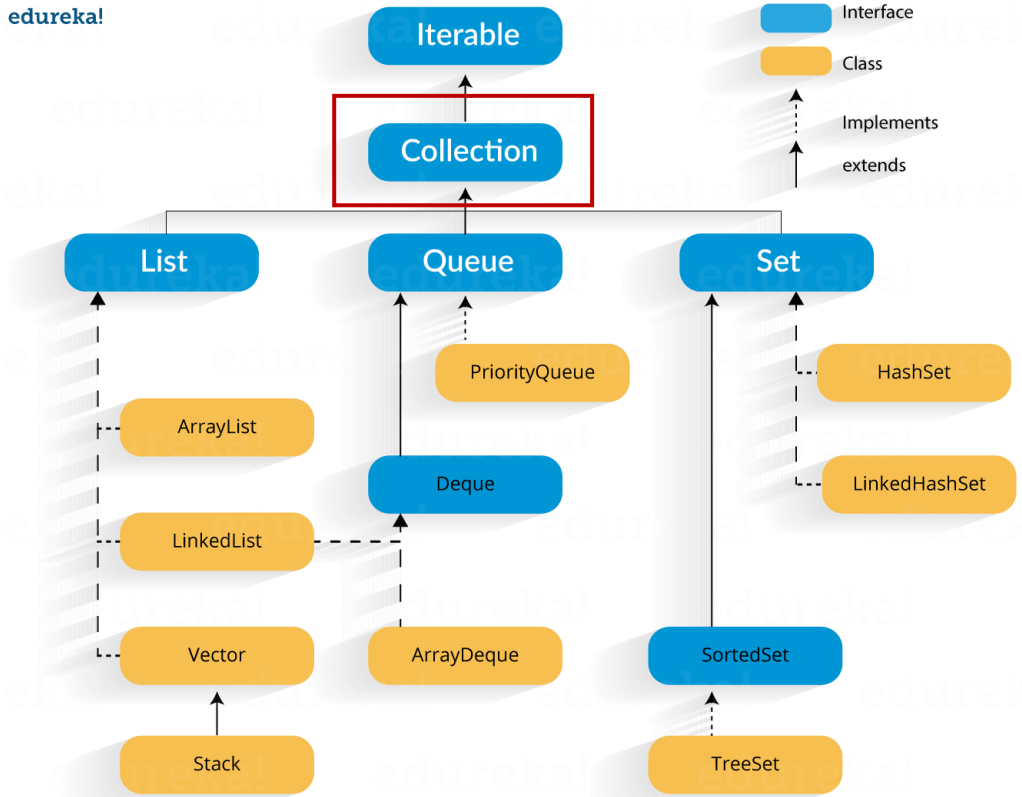
Iterator interface

```
List<String> trooperNames = new ArrayList<>();
trooperNames.add("FN-2198");
trooperNames.add("FN-2199");
trooperNames.add("FN-2200");
```

```
Iterator<String> iterator = trooperNames.iterator();

while(iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

- Interface om de objecten van een lijst één per één op te vragen
- Voorziet o.a. volgende methodes:
  - hasNext() : om te vragen of er nog een element in de lijst is
  - next() : geef het volgende element in de lijst terug
- Elke Collection implementatie voorziet zijn eigen manier om een Iterator aan te maken ! (want Iterator is een interface)

Collections

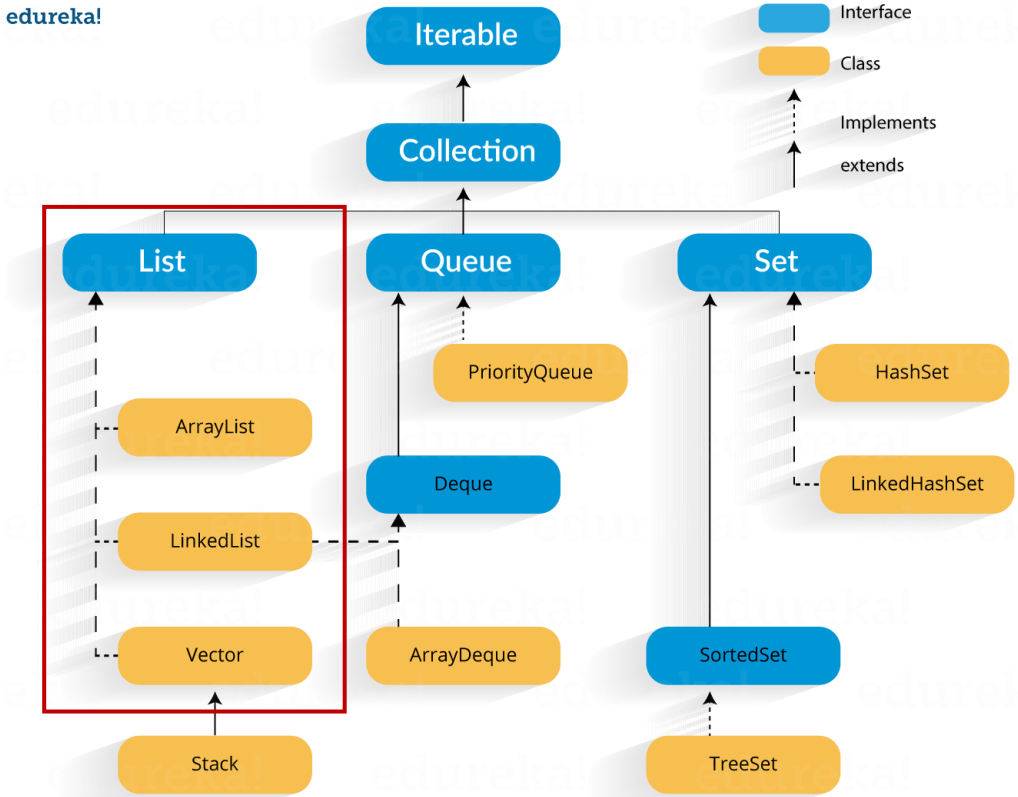


Collection interface

```
public interface Collection<T> extends Iterable<T>
```

- Bouwt verder op Iterable
- Voegt alle basisfunctionaliteiten van lijsten toe:
  - size()
  - isEmpty()
  - add(T)
  - remove(T)
  - …
- Geen implementatie want is een interface !

Collections



List interface

```
public interface List<T> extends Collection<T>
```

- Bouwt verder op Collection
- Voegt **volgorde** en **sortering** toe:
  - get(index) / set(index) / add(index) / remove(index) / indexOf()
  - sort()

- Geen implementatie want is een interface !

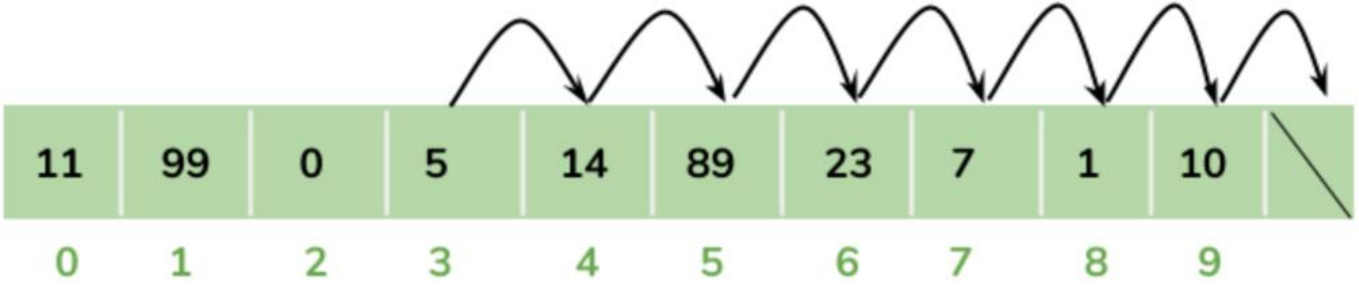
ArrayList class

```
public class ArrayList<T> implements List<T>
```

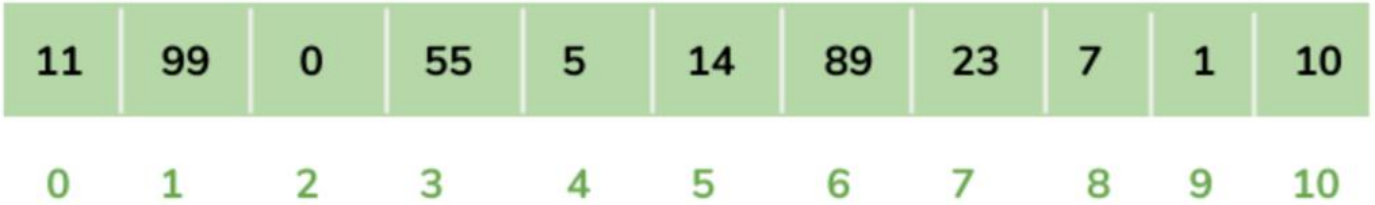
- Implementatie van de List interface
- Dynamisch elementen toevoegen, verwijderen en opvragen
- Duplicaten zijn toegestaan
- Gebaseerd op een Array structuur
- Zonder vaste grootte, de lengte van de Array past zich aan
- Het is mogelijk om bij aanmaak van de ArrayList al een grootte mee te geven, maar deze wordt aangepast tijdens gebruik

ArrayList class

Adding Element in ArrayList at specified position:

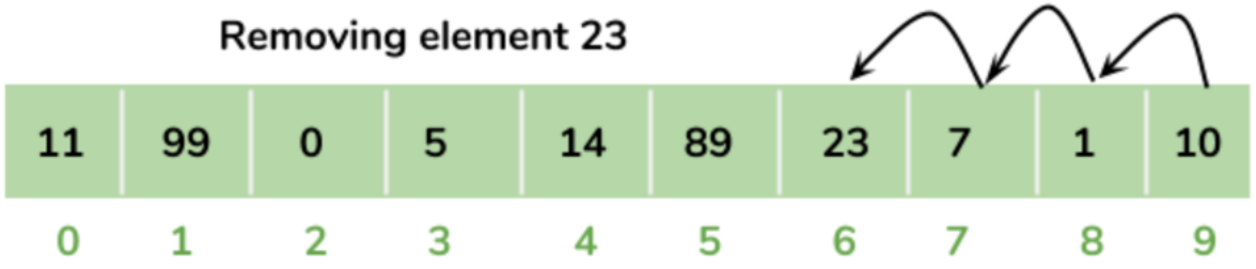


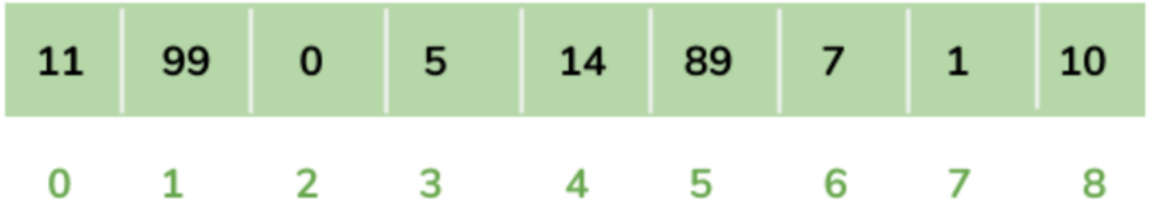
Adding element 55 at fourth position(index 3)



ArrayList class

Removing Element from ArrayList:



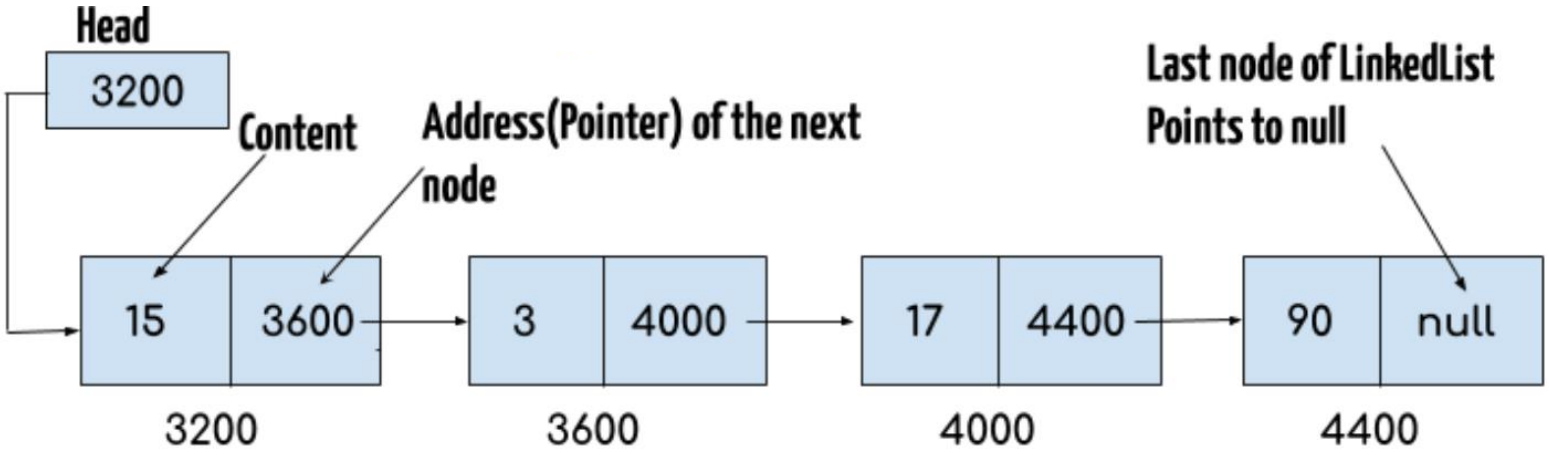


LinkedList class

```
public class LinkedList<T> implements List<T>
```

- Implementatie van de List interface
- Dynamisch elementen toevoegen, verwijderen en opvragen
- Duplicaten zijn toegestaan
- Gebaseerd op een lineaire data structuur met nodes
- Elke node bevat data en verwijzing naar volgende node

LinkedList class



ArrayList vs LinkedList

- Zoeken in de lijst
- Verwijderen in de lijst
- Geheugengebruik van de lijst

ArrayList vs LinkedList

- Zoeken in de lijst
  - ArrayList houdt zelf de index bij.
  - LinkedList moet steeds door heel de lijst gaan.
- Verwijderen in de lijst

- Geheugengebruik van de lijst

ArrayList vs LinkedList

- Zoeken in de lijst
  - ArrayList houdt zelf de index bij.
  - LinkedList moet steeds door heel de lijst gaan.
- Verwijderen in de lijst
  - ArrayList moet alle elementen verschuiven in de array.
  - LinkedList wijzigt de referentie van één element.
- Geheugengebruik van de lijst

ArrayList vs LinkedList

- Zoeken in de lijst
  - ArrayList houdt zelf de index bij.
  - LinkedList moet steeds door heel de lijst gaan.
- Verwijderen in de lijst
  - ArrayList moet alle elementen verschuiven in de array.
  - LinkedList wijzigt de referentie van één element.
- Geheugengebruik van de lijst
  - ArrayList gebruikt één grote brok geheugen, zeer moeilijk voor de garbage collector.
  - LinkedList zijn allemaal kleine blokjes, makkelijk op te kuisen.

De juiste lijst voor de juiste taak

- ArrayList
  - Opslag van data
  - Opvragen van data
- LinkedList
  - Manipulatie van data

- Meest gebruikt : ArrayList omdat het veel efficiënter is voor het opvragen van gegevens, wat je toch meestal doet bij een lijst.

Vector class

```
public class Vector<T> implements List<T>
```

- Implementatie van de List interface
- Dynamisch elementen toevoegen, verwijderen en opvragen
- Duplicaten zijn toegestaan
- Legacy code… restant van vroegere tijden, wordt zelden gebruikt
- Werkt net als ArrayList
- Meer controle over grootte en hoe de Vector groeit

Collections

Array

Array

```
String[] names = new String[] {“Luke Skywalker”, “Darth Vader”};
int[] numbers = new int[6];

System.out.println(names[0]); //Luke Skywalker
System.out.println(numbers.length) //6
```

- Is een verzameling van elementen van hetzelfde data type
- Heeft een index, start index = 0
- Heeft een lengte
- Bij declaratie :
  - de elementen meegeven
  - de grootte meegeven (elke cel opgevuld met default waarde)

Array

```
String[] names = new String[] {“Luke Skywalker”, “Darth Vader”};

List<String> namesAsList = Arrays.asList(names);
```

- Array is de basis van veel List implementaties.
- Wordt in de praktijk minder gebruikt omdat een lijst een duidelijkere interface heeft.
- Conversie van Array -> List : Arrays.asList()

Array - notatie

```
//java style
String[] names = new String[] {“Luke Skywalker”, “Darth Vader”};
//C style
String names[] = new String[] {“Luke Skywalker”, “Darth Vader”};
```

- Opgelet : java style vs C style

Equals en HashCode

equals

```
StormTrooper trooper1 = new StormTrooper(“FN-2198”, RANK.SERGEANT);
StormTrooper trooper2 = new StormTrooper(“FN-2199”, RANK.SOLDIER);
```

Hoe twee objecten vergelijken?

→ trooper1 == trooper2 of trooper1 != trooper2 ???

In Java mag de notatie == en != enkel gebruikt worden voor primitive datatypes en enums !!!

Bij objecten mag je deze notatie niet gebruiken.

equals

```
StormTrooper trooper1 = new StormTrooper(“FN-2199”, RANK.SOLDIER);
StormTrooper trooper2 = new StormTrooper(“FN-2199”, RANK.SOLDIER);

trooper1.equals(trooper2);
```

- method equals : is standaard voorzien op ALLE OBJECTEN

Is trooper1 gelijk aan trooper2 ?

→ NEEN : het zijn twee verschillende objecten

Default implementatie van equals() controleert dat referenties in het geheugen gelijk zijn en niet de inhoud van het object.

@Override equals

```
@Override
public boolean equals(Object o) {
   StormTrooper trooper = (StormTrooper) o;
   return Object.equals(name, trooper.name) && rank == trooper.rank;
}
```

- method equals : eigen implementatie voorzien op je object, je bepaalt dan zelf aan welke voorwaarden een object moet voldoen om ‘gelijk’ te zijn.

Is trooper1 gelijk aan trooper2 ?

→ JA : de naam en rang zijn hetzelfde

Not equals

```
StormTrooper trooper1 = new StormTrooper(“FN-2198”, RANK.SERGEANT);
StormTrooper trooper2 = new StormTrooper(“FN-2199”, RANK.SOLDIER);

!trooper1.equals(trooper2);
```

Plaats de not notatie (!) voor de codelijn

hashCode

De hashCode van een object wordt bepaald door een hashing algoritme.

Deze hashCode wordt o.a. gebruikt om twee objecten te vergelijken in hash structuren zoals bv HashMap.

Regel : als twee objecten gelijk zijn moet hun hashCode ook gelijk zijn.

Default hashCode

```
StormTrooper trooper1 = new StormTrooper(“FN-2199”, RANK.SOLDIER);
StormTrooper trooper2 = new StormTrooper(“FN-2199”, RANK.SOLDIER);

System.out.println(trooper1.hashCode());
System.out.println(trooper2.hashCode());
```

Zijn deze twee hash codes gelijk?

→ NEEN : standaard zal elk object een unieke hash code toegewezen krijgen

De regel van hash codes wordt niet gerespecteerd.

@Override hashCode

```
@Override
public int hashCode() {
   return Objects.hash(name, rank);
}
```

Bij aanmaken van custom equals methode steeds ook een hashCode methode aanmaken !

De generate functies van de meeste IDE’s doen dit automatisch.

Gebruik die generate functies en ga niet zelf je equals() methode schrijven.

hashCode collision

Hash code = Integer → 4.294.967.296 mogelijke getallen.

Kan het zijn dat twee verschillende objecten toch dezelfde hash code hebben?

Birthday Paradox

https://nl.wikipedia.org/wiki/Verjaardagenparadox

Er zijn 365 dagen in een jaar.

Hoe groot moet de groep zijn zodat minstens 2 personen dezelfde verjaardag hebben?

We testen het uit !

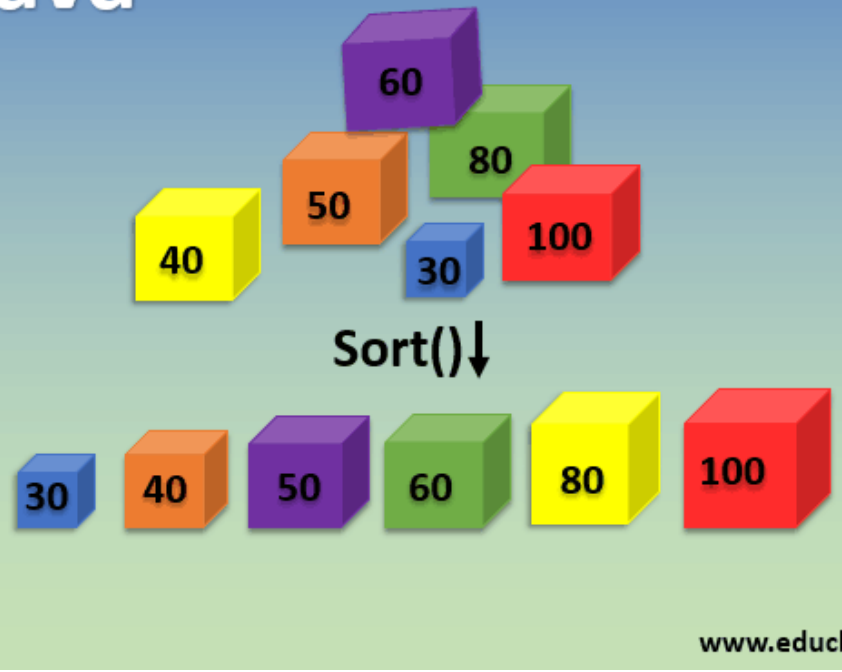
hashCode collision

Vanaf 77.163 objecten is de kans 50% dat er een collision is.

Structuren zoals HashMap kunnen hiermee overweg.

→ GEBRUIK NOOIT DE HASH CODE ALS UNIEKE SLEUTEL

Sorting in Java



Sortering

```
List<String> trooperNames = new ArrayList<>();
trooperNames.add(“FN-2199”);
trooperNames.add(“FN-2198”);
trooperNames.add(“FN-2200”);

Collections.sort(trooperNames);
```

Collections.sort() past standaard sortering toe.

De volgorde binnen de lijst wordt aangepast!

Sortering

```
List<StormTrooper> troopers = new ArrayList<>();
troopers.add(new StormTrooper(“FN-2198”, Rank.SERGEANT);
troopers.add(new StormTrooper(“FN-2199”, Rank.SOLDIER);
troopers.add(new StormTrooper(“FN-2200”, Rank.SOLDIER);

Collections.sort(troopers); //compile error
```

Bij eigen objecten werkt de sortering niet.

Waarom?

Comparable interface

- Om te sorteren moet het algoritme weten wat de sorteerregels zijn.
- Je moet zelf op je eigen objecten de regels bepalen.

- Comparable interface bevat één methode : compareTo

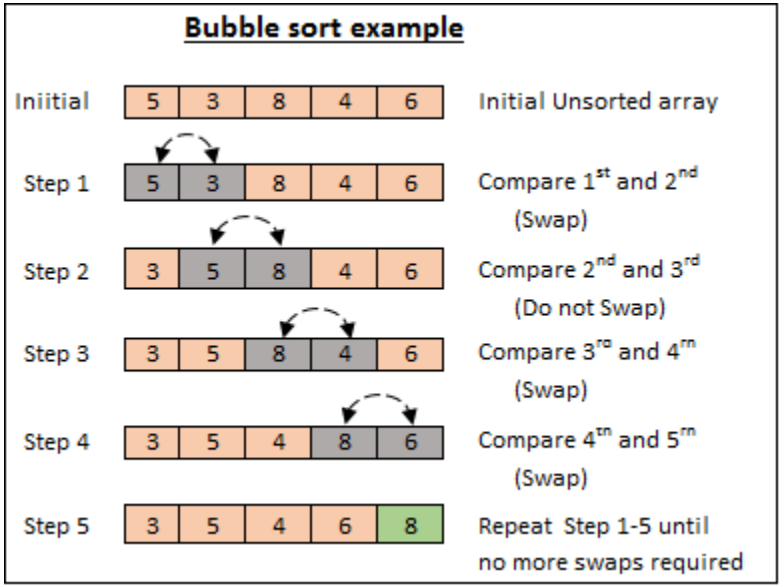
compareTo

```
public int compareTo(StormTrooper other) {
   //return -1, 0 of 1
}
```

compareTo vergelijkt het huidige object met een ander object.

Resultaat:

- this komt voor other : -1
- this gelijk aan other : 0
- **other komt voor this : 1**



Comparable interface - implements

```
public class StormTrooper implements Comparable<StormTrooper> {

   public int compareTo(StormTrooper o) {
      return 0;
   }
}
```

Comparable interface gebruikt een generic om het juiste data type door te geven aan de compareTo method.

Je moet daarna nog zélf beslissen wat de sorteerregels zijn.

Sorteerregel vb 1 : sorteer op naam

```
public int compareTo(StormTrooper o) {
   return this.name.compareTo(o.name);
}
```

Er wordt op de naam gesorteerd.

Gebruik de compareTo van String voor de sortering.

Sorteerregel vb 2 : sorteer op rang

```
public int compareTo(StormTrooper o) {
   if(this.rank == Rank.SERGEANT && o.rank == Rank.TROOPER) {
      return -1;
   } else if(this.rank == Rank.TROOPER && o.rank == Rank.SERGEANT) {
      return 1;
   } else {
      return 0;
   }
}
```

Sergeant staat voor trooper.

We bepalen zelf wanneer het -1, 0 of 1 is.

Sorteerregel vb 3 : sorteer op rang en naam

```
public int compareTo(StormTrooper o) {
   if(this.rank == Rank.SERGEANT && o.rank == Rank.TROOPER) {
      return -1;
   } else if(this.rank == Rank.TROOPER && o.rank == Rank.SERGEANT) {
      return 1;
   } else {
      return this.name.compareTo(o.name);
   }
}
```

Eerst de sortering op rang.

Als het resultaat ‘0’ zou zijn, dan nieuwe sortering op naam.

Alternatieve sorteervolgorde

- Comparable interface beschrijft de default sortering van het object.

- Wat als je een andere sortering nodig hebt dan de default?

→ Comparator interface

Comparator

```
public class SortByNameDescending implements Comparator<StormTrooper> {

   public int compare(StormTrooper o1, StormTrooper o2) {
      return o2.getName().compareTo(o1.getName());
   }

}
```

Comparator interface:

- Generic data type
- methode compare()
  - Heeft twee objecten als input
  - Geeft als output -1, 0 of 1

Comparator - gebruik

```
List<StormTrooper> troopers = new ArrayList<>();
troopers.add(new StormTrooper(“FN-2198”, Rank.SERGEANT);
troopers.add(new StormTrooper(“FN-2199”, Rank.SOLDIER);
troopers.add(new StormTrooper(“FN-2200”, Rank.SOLDIER);

Collections.sort(troopers, new SortByNameDescending());
```

Collections.sort met extra parameter : de comparator class

Collections.sort - alternatief

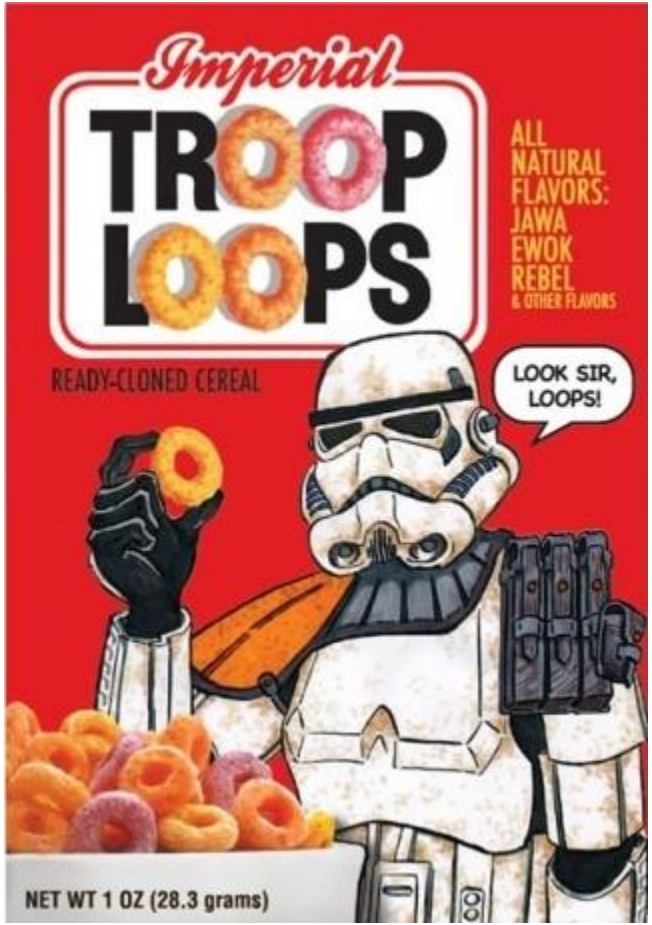
```
Collections.sort(troopers);
troopers.sort(Comparator.naturalOrder());

Collections.sort(troopers, new SortByNameDescending());
troopers.sort(new SortByNameDescending());
```

Collections.sort kan vervangen worden door de .sort methode van de list interface.

Deze sort methode moet een Comparator als input krijgen.

Voor default sortering is dat : Comparator.naturalOrder()

Loops


Imperial
TROOP LOOPS
READY-CLONED CEREAL
ALL NATURAL FLAVORS:
JAWA
EWOK
REBEL
& OTHER FLAVORS
LOOK SIR, LOOPS!
NET WT 1 OZ (28.3 grams)

Soorten loops

Vier soorten :

- **for** loop
- **for each** loop
- **while** loop
- **do while** loop

for loop

```
List<Character> characters = new ArrayList<>();
for(int i = 0 ; i < characters.size() ; i++) {
   //do something with characters.get(i);
}
```

keyword for : definieert de loop

- Initialisatie : int i = 0
- Conditie : i < characters.size()
- Iteratie : i++

Opgelet : puntkomma tussen initialisatie, conditie en iteratie

for each loop

```
List<Character> characters = new ArrayList<>();
for(Character character : characters) {
   //do something with character;
}
```

keyword for : definieert de loop

Verschil met de for loop : geen initialisatie, conditie of iteratie

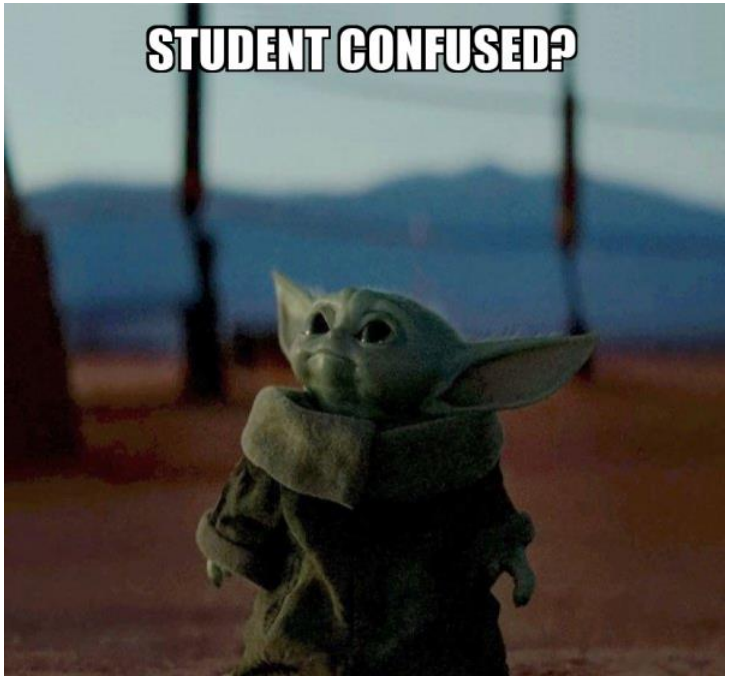
Opgelet : dubbelpunt tussen variabele declaratie en list

“for” of “for each” ?

Vuistregel…

Gebruik “for” als de index van de elementen in de lijst belangrijk is, bv als je elementen moet bewerken of verwijderen.

Gebruik “for each” in alle andere gevallen.



while loop

```
List<Character> characters = new ArrayList<>();
while(characters.size() > 0) {
   //remove an element from list characters
}
```

keyword while : definieert de loop

- Conditie : characters.size() > 0

De while loop blijft lopen tot de conditie niet meer geldig is.

Opgelet : eindeloze lus!!!

do while loop

```
List<Character> characters = new ArrayList<>();
do {
    //remove an element from list characters
} while(characters.size() > 0);
```

Keywords do en while : definieren de loop

- Conditie : characters.size() > 0

De while loop blijft lopen tot de conditie niet meer geldig is.

Wat is het verschil tussen ‘while’ en ‘do while’ ?

break

keyword break : beëindig de loop

Gebruik dit keyword om als een bepaalde conditie in je loop voldoet om deze dan stil te leggen.

Typisch als je een element aan het zoeken bent in je lijst.

```
List<Character> characters = new ArrayList<>();
for(Character character  : characters) {
   System.out.println(character.toString());

   if(character.isForceUser()) {
      System.out.println("Use the Force!");
      break;
   }

}
```

continue

keyword continue : start de volgende iteratie van de loop

Gebruik dit keyword om verdere verwerking in de body te stoppen en naar het volgende element in de lijst te gaan.

```
List<Character> characters = new ArrayList<>();
for(Character character  : characters) {
   if(character.isForceUser()) {
      System.out.println("Use the Force!");
      continue;
   }
   System.out.println(character.toString());
}
```


FOR TODAY
ANYWAY...