Java

2023 – 2024 : les 3

jeroen.devos01@ap.be

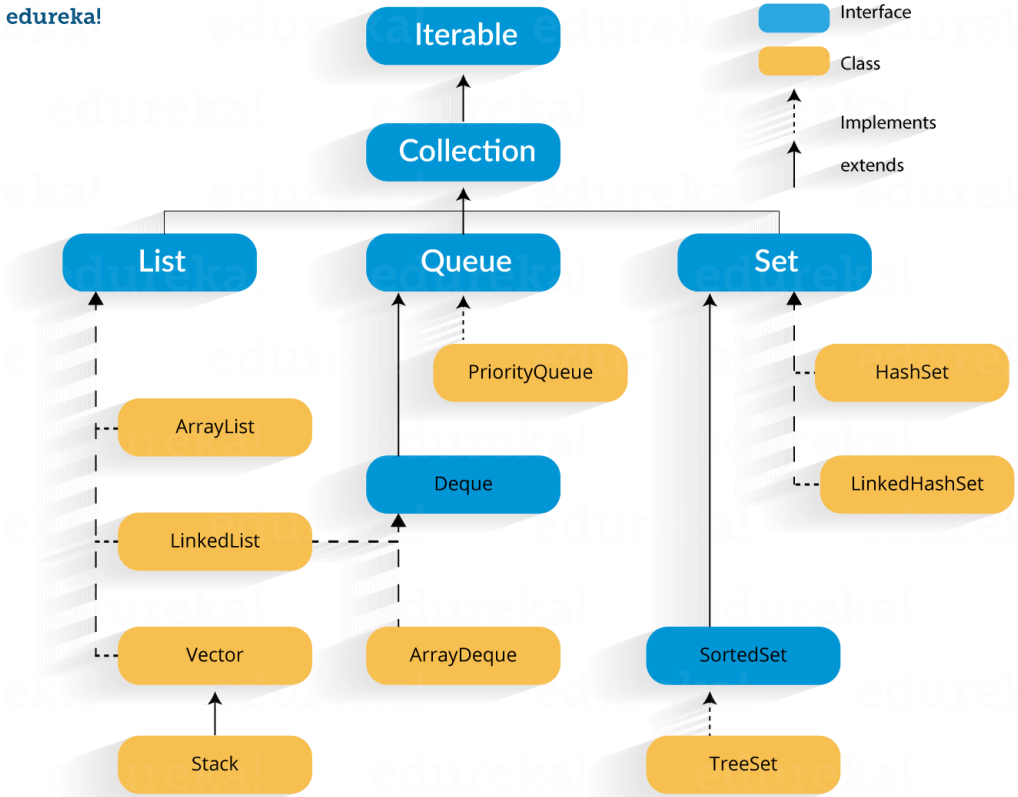
AP HOGESCHOOL ANTWERPEN

AP.BE

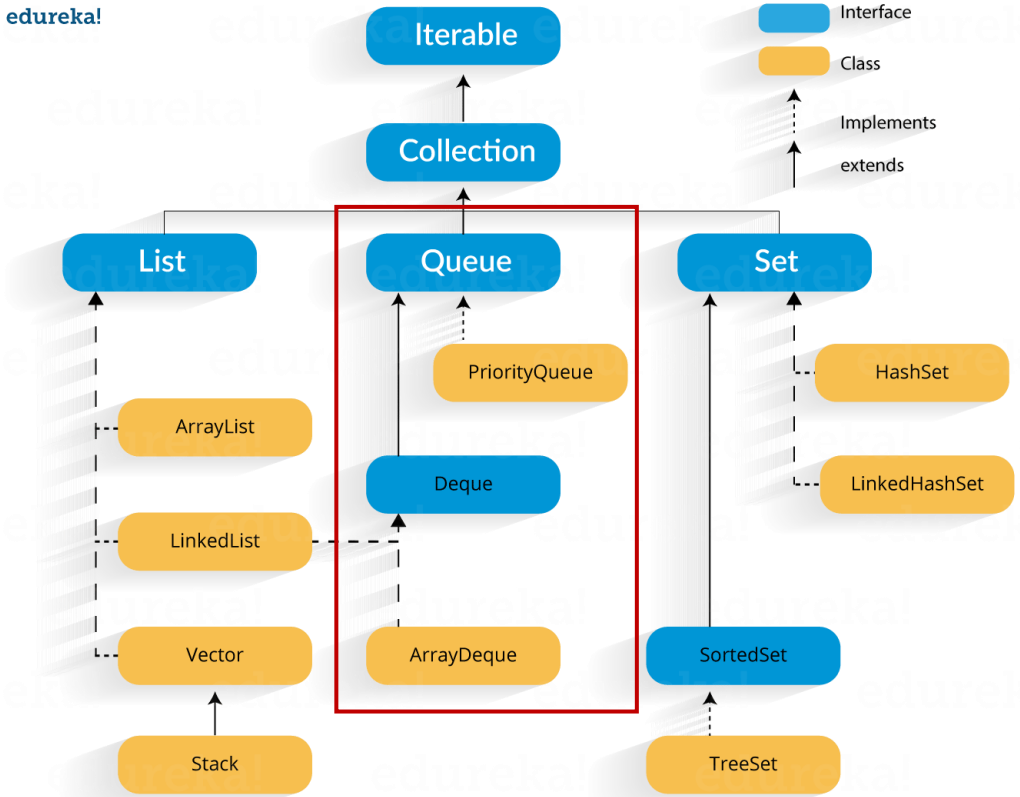
Lesdoelen

- Collections
- Map
- Recursie en iteratie
- Werken met console

Collections



Collections



Queue interface

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

- Bouwt verder op Collection
- Elementen worden in een volgorde gezet bepaald door de implementatie
  - Volgens een sortering
  - FIFO / LIFO
- Voorziet methods: offer() / poll() / peek()

- Geen implementatie want is een interface !

PriorityQueue class

```
public class PriorityQueue<T> implements Queue<T>
```

- Implementatie van de Queue interface
- Elementen worden gesorteerd bij toevoegen aan de queue
  → naturalOrdering of eigen comparator
- Duplicaten zijn toegestaan
- Gebaseerd op een Array structuur
- Zonder vaste grootte, de lengte van de Queue past zich aan

- **Opgelet : alle elementen moeten sorteerbaar zijn (Comparable)**

PriorityQueue class

```
Queue<StarWarsMovie> movies = new PriorityQueue<>();
movies.offer(new StarWarsMovie(1, “The Phantom Menace”));
movies.offer(new StarWarsMovie(2, “Attack of the Clones”));

//movies default sortering op episode
System.out.println(movies.poll()); // “The Phantom Menace”
```

```
Queue<StarWarsMovie> movies = new PriorityQueue<>(new SortByName());
movies.offer(new StarWarsMovie(1, “The Phantom Menace”));
movies.offer(new StarWarsMovie(2, “Attack of the Clones”));

//movies sortering op naam
System.out.println(movies.poll()); // “Attack of the Clones”
```

Deque interface

```
public interface Deque<T> extends Queue<T>
```

- “double ended queue”
- Twee mogelijkheden : FIFO (first in first out) en LIFO (last in first out)
  - offerFirst() / offerLast()
  - pollFirst() / pollLast()
  - peekFirst() / peekLast()

- Geen implementatie want is een interface !

ArrayDeque class

```
public class ArrayDeque<T> implements Deque<T>
```

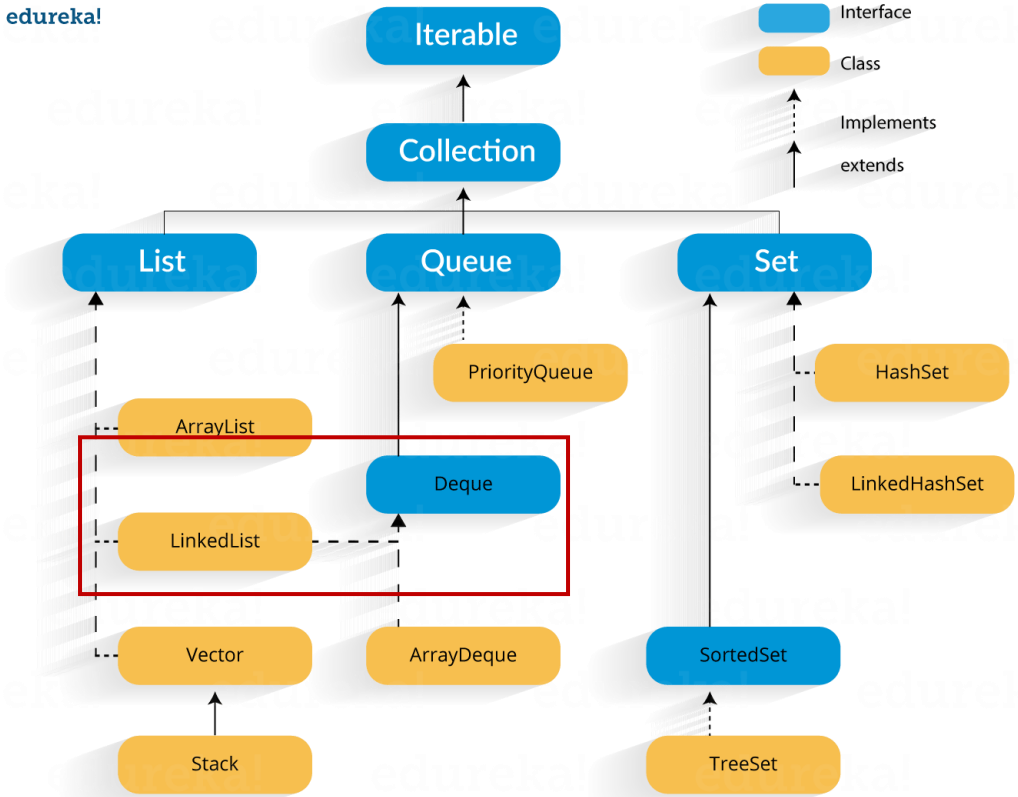
- Implementatie van de Deque interface
- Elementen worden achteraan of vooraan toegevoegd
- Duplicaten zijn toegestaan
- Gebaseerd op een Array structuur
- Zonder vaste grootte, de lengte van de Deque past zich aan
- Default gedrag : FIFO

ArrayDeque class

```
Deque<StarWarsMovie> moviesToSee = new ArrayDeque<>();
moviesToSee.offer(new StarWarsMovie(1, “The Phantom Menace”));
moviesToSee.offer(new StarWarsMovie(2, “Attack of the Clones”));

System.out.println(moviesToSee.poll()); // “The Phantom Menace”
```

- Default gedrag is FIFO:
  - offer() → offerLast()
  - poll() → pollFirst()
  - peek() → peekFirst()

Collections



LinkedList implements Deque

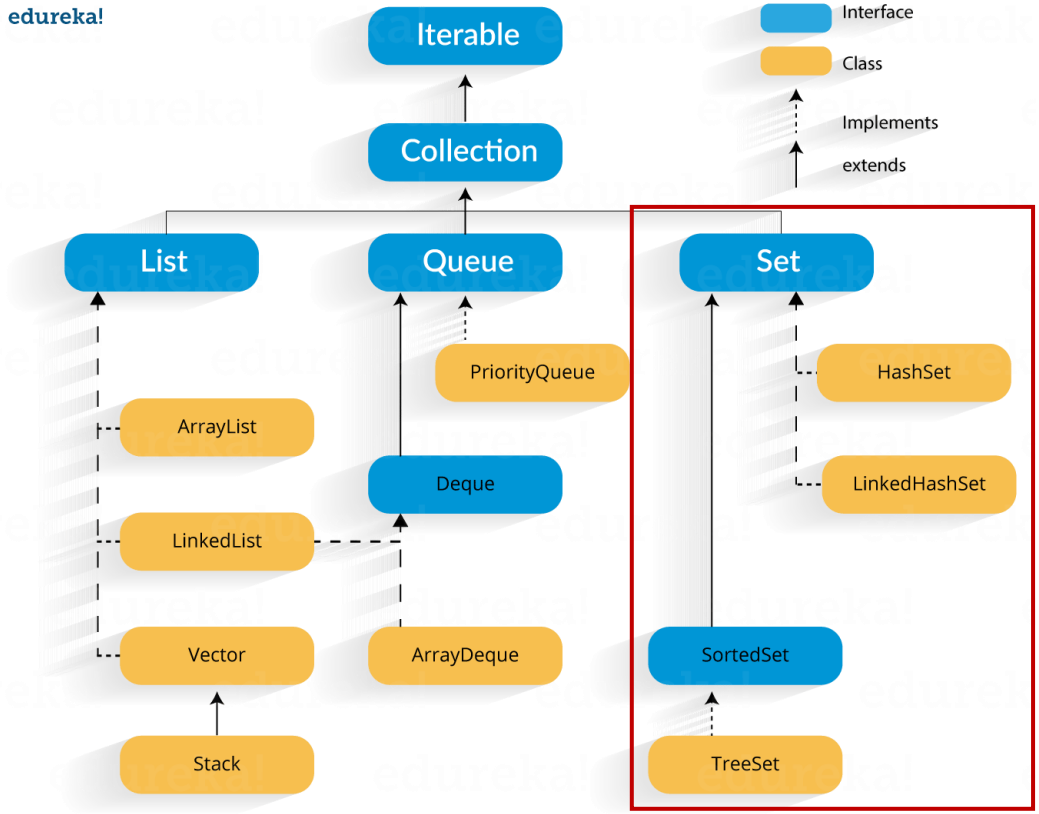
- **LinkedList is ook een Deque implementatie !**

- ArrayDeque vs LinkedList ?

    - LinkedList beter voor geheugen
    - ArrayDeque in meeste gevallen sneller

(= zelfde als bij ArrayList)

Collections



Set interface

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

- Bouwt verder op Collection
- Duplicaten zijn niet toegestaan!
  → Gebruikt hiervoor de equals() methode

- Geen implementatie want is een interface !

HashSet class

```
public class HashSet<T> implements Set<T>
```

- Implementatie van de Set interface
- Elementen worden in een Hash tabel gestoken
- Gebaseerd op een Map structuur
- Geen volgorde in de elementen

- **Opgelet : alle elementen moeten een hashcode hebben**

HashSet class

```
Set<StarWarsMovie> movies = new HashSet<>();
movies.add(new StarWarsMovie(1, “The Phantom Menace”));
movies.add(new StarWarsMovie(2, “Attack of the Clones”));

for (StarWarsMovie movie : movies) {
   System.out.println(movie);
}
```

- Géén “get” method want geen volgorde of sortering
- Enkel opvraagbaar via loop / iterator

LinkedHashSet class

```
public class LinkedHashSet<T> implements Set<T>
```

- Implementatie van de Set interface
- Elementen worden in een Hash tabel gestoken
- Gebaseerd op een Map structuur
- Behoudt de volgorde in de elementen dmv een LinkedList

- **Opgelet : alle elementen moeten een hashcode hebben**

LinkedHashSet class

```
Set<StarWarsMovie> movies = new LinkedHashSet<>();
movies.add(new StarWarsMovie(1, “The Phantom Menace”));
movies.add(new StarWarsMovie(2, “Attack of the Clones”));

for (StarWarsMovie movie : movies) {
   System.out.println(movie);
}
```

- Idem HashSet : geen get en enkel opvraagbaar via loop / iterator
- In de loop is de volgorde van toevoegen behouden

SortedSet interface

```
public interface SortedSet<T> extends Set<T>
```

- Bouwt verder op Set
- Duplicaten zijn niet toegestaan!
- Elementen worden in een volgorde gezet bepaald door de implementatie
  - first() en last()

- Geen implementatie want is een interface !

TreeSet class

```
public class TreeSet<T> implements SortedSet<T>
```

- Implementatie van de SortedSet interface
- Elementen worden in een Hash tabel gestoken
- Gebaseerd op een Map structuur
- Elementen worden gesorteerd bij toevoegen aan de Set
  → naturalOrdering of eigen comparator

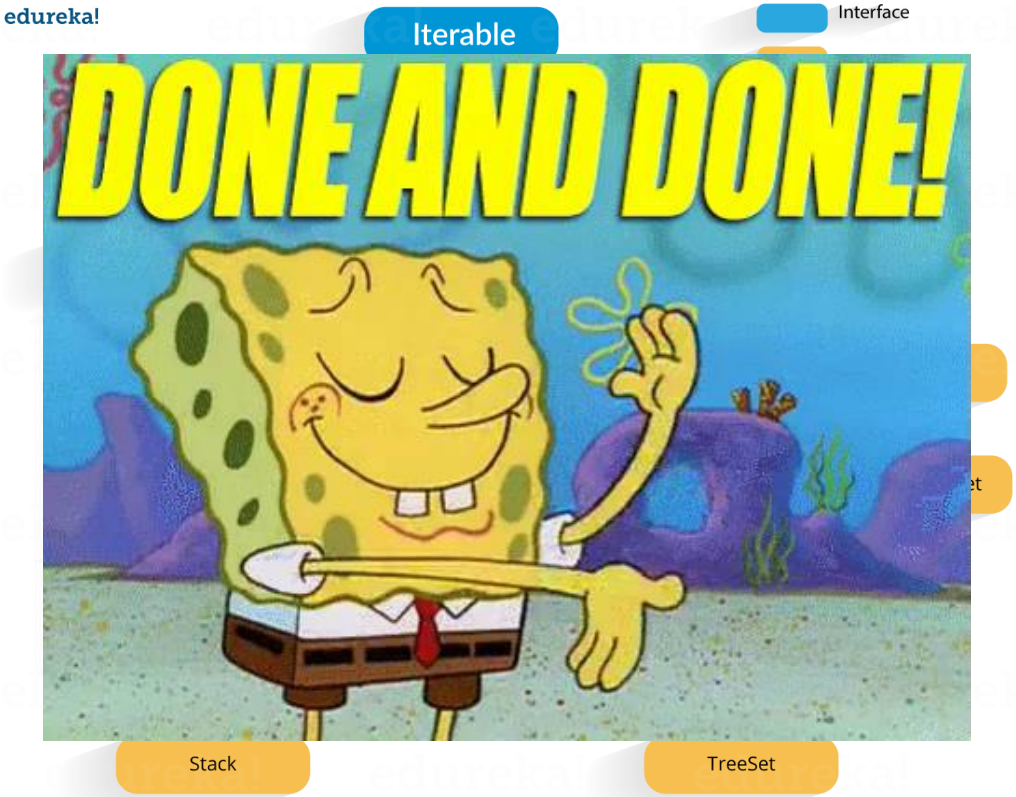
- **Opgelet : alle elementen moeten een hashcode && compareTo hebben**

TreeSet class

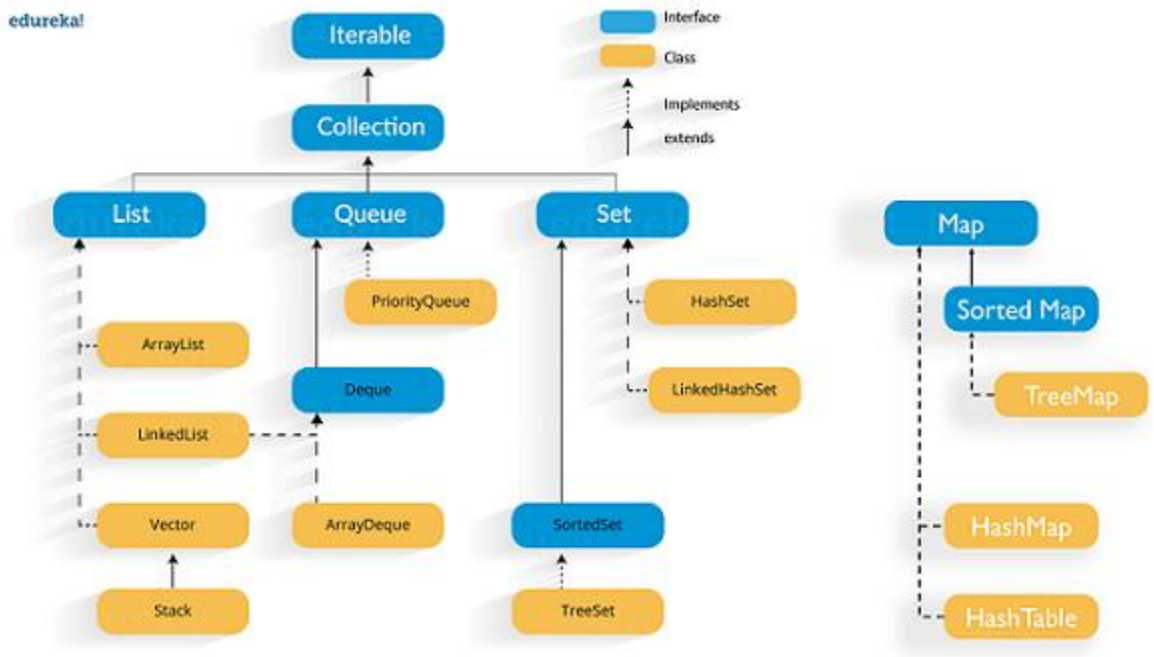
```
SortedSet<StarWarsMovie> movies = new TreeSet<>();
movies.offer(new StarWarsMovie(1, “The Phantom Menace”));
movies.offer(new StarWarsMovie(2, “Attack of the Clones”));

//movies default sortering op episode
System.out.println(movies.first()); // “The Phantom Menace”
```

```
SortedSet<StarWarsMovie> movies = new TreeSet<>(new SortByName());
movies.offer(new StarWarsMovie(1, “The Phantom Menace”));
movies.offer(new StarWarsMovie(2, “Attack of the Clones”));

//movies sortering op naam
System.out.println(movies.first()); // “Attack of the Clones”
```

Collections



Map


edureka!
Iterable
Collection
List
Queue
Set
Map
Interface
Class
Implements
extends
ArrayList
LinkedList
Vector
Stack
PriorityQueue
Deque
ArrayDeque
HashSet
LinkedHashSet
SortedSet
TreeSet
Sorted Map
TreeMap
HashMap
HashTable

Map interface

```
public interface Map<K, V>
```

- Geen collection
- Key – Value store
- Generics K(ey) en V(alue)
- Key is altijd uniek (denk aan **equals**)
- Value kan meermaals voorkomen

- Geen implementatie want is een interface !

Map interface

```
public interface Map<K, V>
```

- put(key, value)
- get(key) -> value
- remove(key)
- keySet() -> Set<K>
- values() -> Collection<V>

HashMap class

```
public class HashMap<K, V> implements Map<K, V>
```

- Implementatie van de Map interface
- Keys worden via hashing vergeleken
- Geen volgorde in de elementen

- **Opgelet : alle keys moeten een hashcode hebben**

HashMap class

```
StarWarsMovie episode4 = new StarWarsMovie(4, “A New Hope”);
StarWarsMovie episode5 = new StarWarsMovie(5, “The Empire Strikes Back”);

Map<StarWarsMovie, StarWarsCharacter> movieCharacterMap = new HashMap<>();
movieCharacterMap.add(episode4, new LukeSkywalker());
movieCharacterMap.add(episode5, new DarthVader());

System.out.println(movieCharacterMap.get(episode4)); //LukeSkywalker
System.out.println(movieCharacterMap.get(episode5)); //DarthVader

//loop over all keys – volgorde niet gekend
for (StarWarsMovie movie : movieCharacterMap.keySet()) {
   System.out.println(movieCharacterMap.get(movie));
}
```

Hashtable class

```
public class Hashtable<K, V> implements Map<K, V>
```

- Implementatie van de Map interface
- Legacy code … wordt niet meer gebruikt
- Alle functionaliteiten zitten in HashMap

SortedMap interface

```
public interface SortedMap<K, V> extends Map<K, V>
```

- Bouwt verder op Map
- Keys worden in een volgorde gezet bepaald door de implementatie
  - firstKey() en lastKey()

- Geen implementatie want is een interface !

TreeMap class

```
public class TreeMap<K, V> implements SortedMap<K, V>
```

- Implementatie van de SortedMap interface
- Keys worden gesorteerd bij toevoegen aan de Map
  → naturalOrdering of eigen comparator

- **Opgelet : alle elementen moeten een hashcode && compareTo hebben (en ook equals voor de Map zelf)**

TreeMap class

```
StarWarsMovie episode4 = new StarWarsMovie(4, “A New Hope”);
StarWarsMovie episode5 = new StarWarsMovie(5, “The Empire Strikes Back”);

SortedMap<StarWarsMovie, StarWarsCharacter> movieCharacterMap = new TreeMap<>();
movieCharacterMap.add(episode4, new LukeSkywalker());
movieCharacterMap.add(episode5, new DarthVader());

//loop over all keys – volgorde = natural ordening van StarWarsMovie
for (StarWarsMovie movie : movieCharacterMap.keySet()) {
   System.out.println(movieCharacterMap.get(movie));
}

//voor sortering op naam
SortedMap<StarWarsMovie, StarWarsCharacter> map = new TreeMap<>(new SortByName());
```


ARE WE THERE YET?
imgflip.com

Advanced mapping

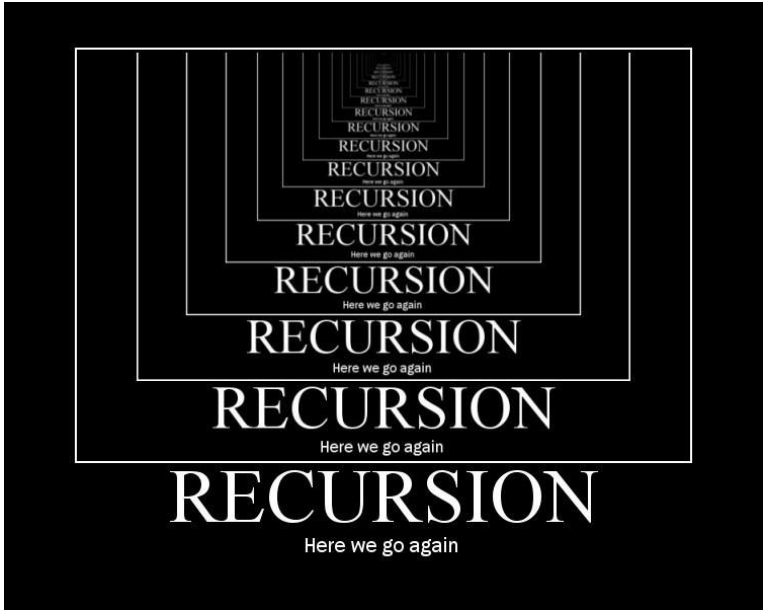
```
StarWarsMovie episode4 = new StarWarsMovie(4, “A New Hope”);

Map<StarWarsMovie, List<StarWarsCharacter>> movieCharacters = new HashMap<>();
movieCharacters.add(episode4, new ArrayList<>());

movieCharacters.get(episode4).add(new LukeSkywalker());
movieCharacters.get(episode4).add(new DarthVader());
```

1. Value = List interface
2. Put lege lijst in Map
3. Get lijst van Map en add element aan lijst

Recursie

- Recursie is een programmeertechniek waarin een methode zichzelf aanroept.

- Elke recursie moet een niet recursief gedeelte bevatten die ervoor zorgt dat de recursie ooit stopt.



Recursie voorbeeld : faculteit

- De faculteit van een natuurlijk getal n, genoteerd als n!, is het product van de getallen 1 tot en met n.

  https://nl.wikipedia.org/wiki/Faculteit_(wiskunde)

- n! = 1 * 2 * 3 * …. * n
- 1! = 1 * 1 = 1
- 5! = 1 * 2 * 3 * 4 * 5 = 120

- Hoe programmeren? → recursief!
- n! = n * (n – 1)! als n > 1

Recursie voorbeeld : faculteit

```
public int calculateFactorial(int number) {
   if(number <= 1) {
      return 1;
   } else {
      return number * calculateFactorial(number – 1);
   }
}
```

- Recursief gedeelte : roept zichzelf terug aan
- Niet recursief gedeelte : zorgt ervoor dat de recursie stopt

Recursie vs iteratie

- Recursie is niet altijd de meest efficiënte manier om een probleem op te lossen!

- Bij elke recursie wordt opnieuw een methode aangeroepen en nieuwe variabelen aangemaakt : geheugenverbruik!

Recursie vs iteratie voorbeeld

- Bereken de som van alle getallen van 1 tot n

- som(n) = n + … + 3 + 2 + 1
- som(4) = 4 + 3 + 2 + 1

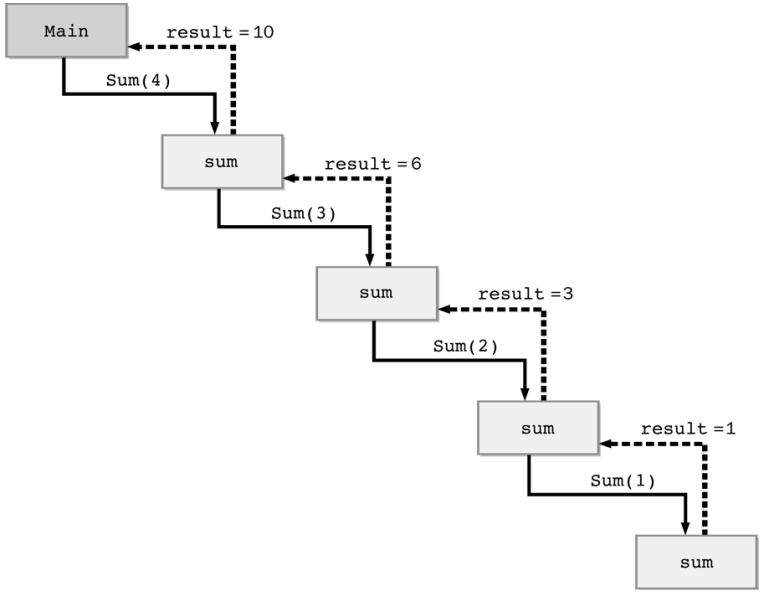


FIGURE 11.3 Herative Computation of 1 . . . N

Recursie vs iteratie voorbeeld

```
//recursie
public int calculateSum(int number) {
   if(number <= 1) {
      return number;
   } else {
      return number + calculateSum(number – 1);
   }
}
```

```
//iteratie
public int calculateSum(int number) {
   int result = 0;
   for(int i = 1; i <= number; i++) {
      result = result + i;
   }
   return result;
}
```

- Resultaat v/d test : recursief is trager dan iteratief

Recursie voorbeeld : torens van Hanoi

- Alle schijven van de eerste stok naar de derde stok verplaatsen
  - Per beweging mag maar 1 schijf verplaatst worden
  - Een grotere schijf mag nooit op een kleinere schijf liggen
  - Alle schijven moeten op een stok zitten

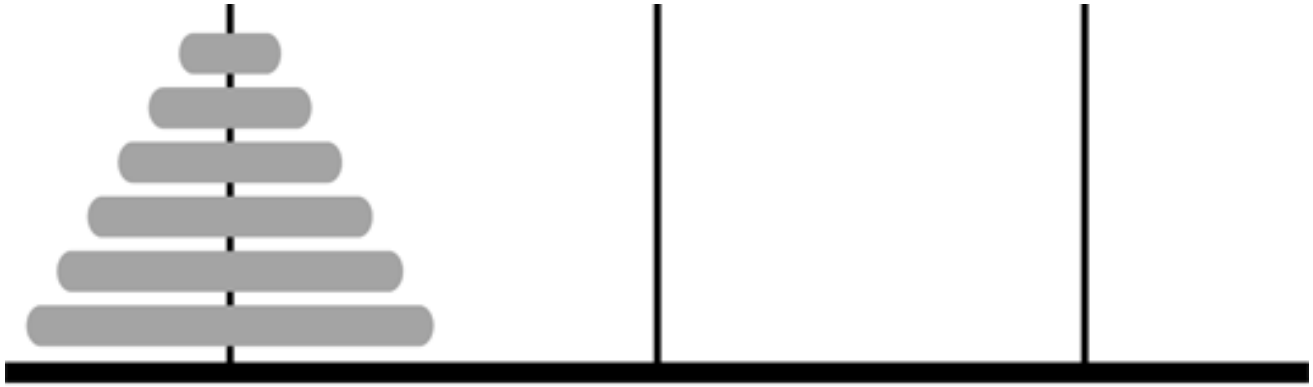


FIGURE 11.5 The Towers of Hanoi puzzle

Torens van Hanoi oplossing

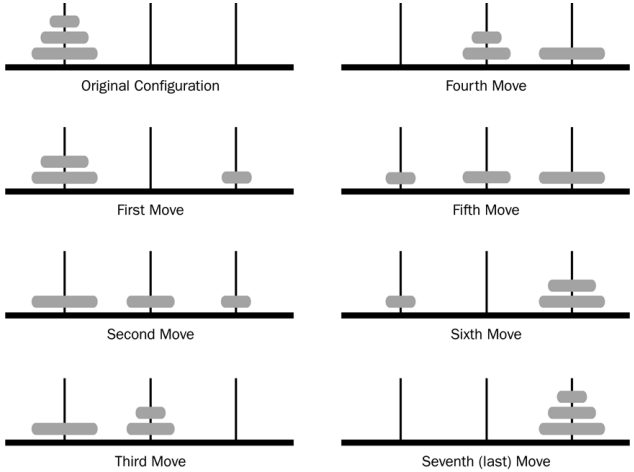
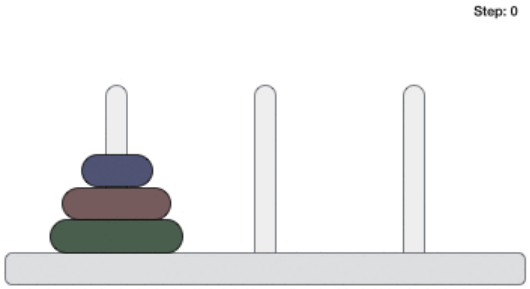


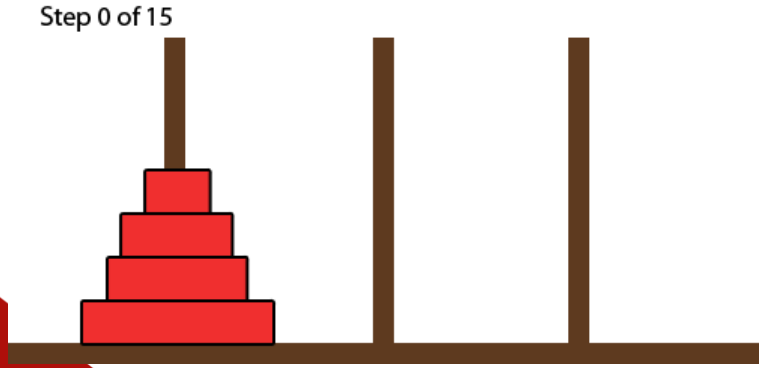
FIGURE 11.6 A solution to the three-disk Towers of Hanoi puzzle

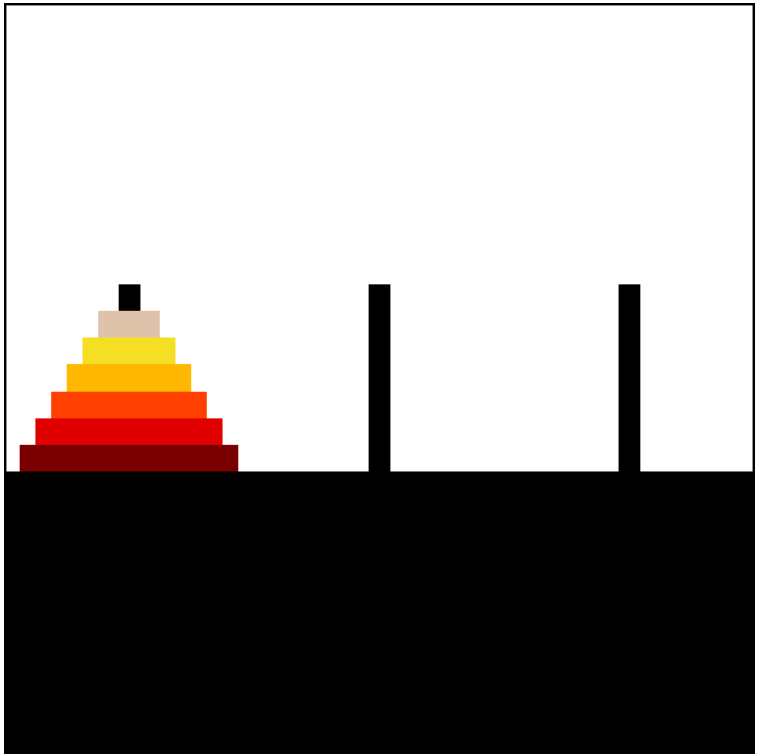
```
public void doTower(int disk, String t1, String t2, String t3) {
   if(disk == 1) {
      System.out.println("Ring " + disk + " van " + t1 + " naar " + t3);
   } else {
      doTower(disk-1, t1, t3, t2);
      System.out.println("Ring " + disk + " van " + t1 + " naar " + t3);
      doTower(disk-1, t2, t1, t3);
   }
}
```

- Recursieve oplossing !

Exponentiële complexiteit







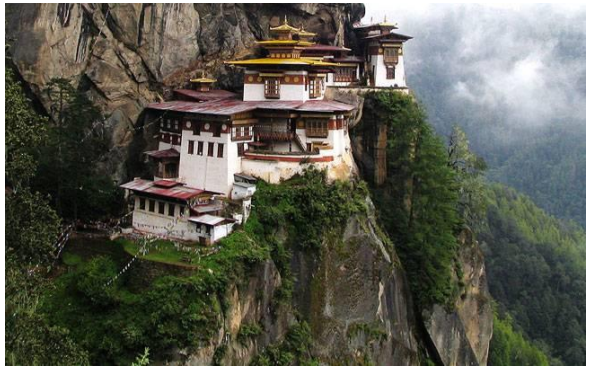
Legende…

Volgens de legende zou er in India een tempel zijn met drie diamanten naalden en 63 gouden schijven. Als alle schijven ritueel zijn overgezet, zal de wereld eindigen.

https://ianparberry.com/TowersOfHanoi/

Stel dat een verplaatsing 1 seconde zou duren, dan duurt het oplossen van de puzzel 585 miljard jaar.

Maar wanneer zijn ze begonnen…



Recursie voorbeeld : backtracking

Probleem :

- Vind de juiste weg door een doolhof.
- Los een sudoku op

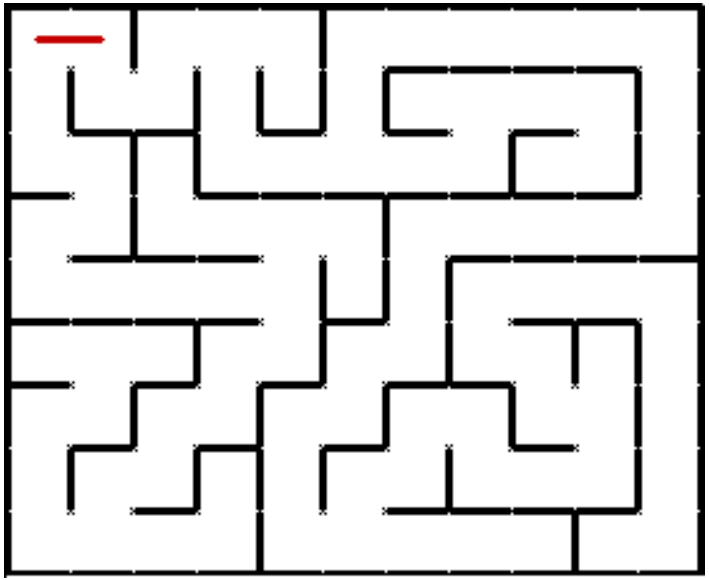
Oplossing : backtracking

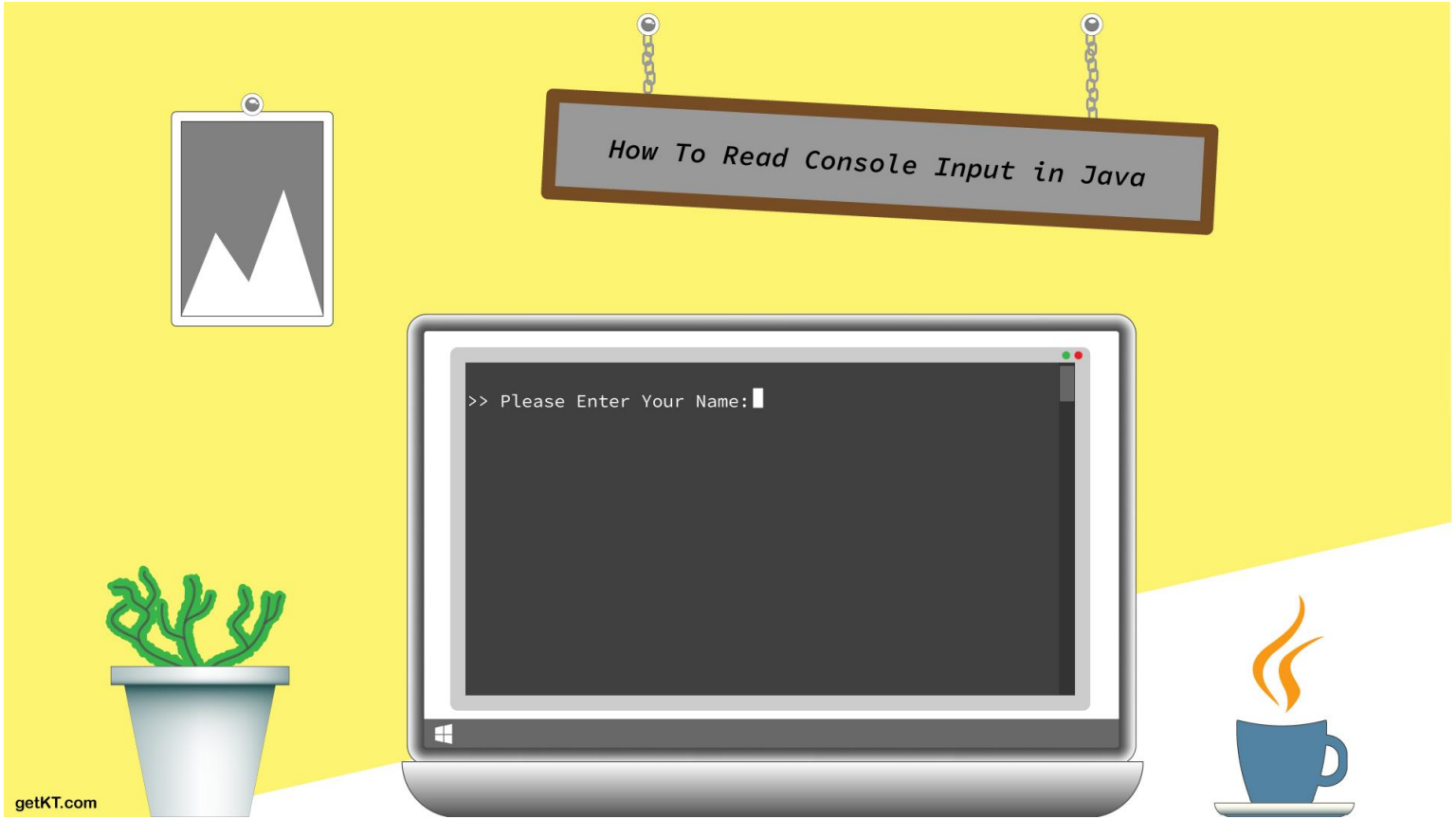
Je lost het probleem op door recursief oplossingen te proberen waarbij je foute oplossingen weg haalt.

Bv : volg een pad door het doolhof tot je niet verder kan en ga dan een stapje terug om een ander pad te volgen

Backttracking voorbeeld:

Vind de juiste weg door het doolhof




How To Read Console Input in Java
>> Please Enter Your Name:
getKT.com

Werken met de console

Via System.out.println(); kan je zaken in de console tonen.

Kan je ook input van de gebruiker vragen in de console?

JA : via (o.a.) het Scanner object

Er zijn ook andere manieren zoals BufferedReader of Console, maar Scanner is het meest flexibele.

Werken met de console

```
Scanner scanner = new Scanner(System.in);

String input = scanner.nextLine();
```

- Maak een nieuw scanner object aan en geef als constructor parameter System.in mee. System.in verwijst naar input op de console.
- scanner.nextLine() vraagt aan de gebruiker om een waarde in te geven, je moet eindigen met ‘enter’.
- Het resultaat van de scanner.nextLine() kan je in een variabele stockeren.

Werken met de console – meer input

```
Scanner scanner = new Scanner(System.in);

String firstName = scanner.nextLine();
String lastName = scanner.nextLine();
```

Blijf het scanner object gebruiken! Je maakt geen nieuw scanner object aan voor elke lijn die je wenst in te lezen.

Opgepast voor scanner.next() : deze kan niet overweg met spaties! Gebruik steeds scanner.nextLine()

Werken met de console – input getal

```
Scanner scanner = new Scanner(System.in);

int age = scanner.nextInt();
scanner.nextLine(); // vangt de newline op van de ‘enter’ knop
String placeOfBirth = scanner.nextLine();
```

Je kan rechtstreeks een getal inlezen via scanner.nextInt().

Opgepast : nextInt() leest énkel getallen in. De ‘enter’ is géén getal en zal automatisch de volgende nextLine() triggeren. Daarom moet je deze ‘enter’ opvangen met een extra scanner.nextLine() .


FOR TODAY
ANYWAY...