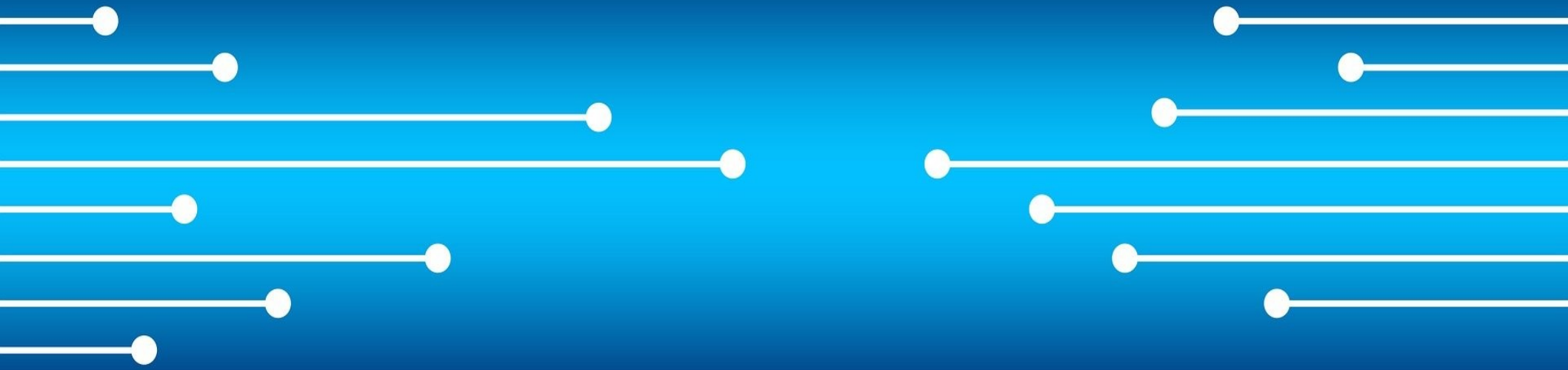
Datenverarbeitung

Teil des Moduls 5CS-DPDL-20



Prof. Dr. Deweß

Thema 4

„Collections“

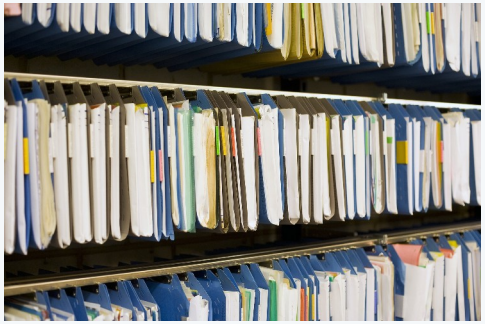
- Schnittstellen
- Implementierungen (Klassen)
- Beispiele

Java Collections Framework

- Objekt, um **mehrere Objekte strukturiert** als Einheit (Liste, Menge, Abbildung, Warteschlange …) zu repräsentieren
- benutzt, um Datensammlungen abzurufen und zu speichern, zu manipulieren und zu kommunizieren
- im Paket „java.util" enthalten, also

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

nicht vergessen



Java Collections Framework

einheitliche Architektur zum Darstellen und Bearbeiten von Daten-Sammlungen; ermöglicht Konzentration auf die eigentliche Programmieraufgabe anstatt auf Hilfsprogramme

Bestandteile eines „Collection Frameworks"

- **Schnittstellen (interfaces)**: abstrakte Datentypen, um Sammlungen detailunabhängig darzustellen
- **Implementierungen**: konkrete (teilweise trotzdem abstrakte) Implementierungen dieser Schnittstellen, also im Wesentlichen wiederverwendbare Datenstrukturen
- **Algorithmen**: wiederverwendbare, polymorphe Methoden für nützliche Aktionen wie „Suchen", „Mischen", „Sortieren", ...

Java Collections Framework

wichtige Schnittstellen im „Java Collections Framework“

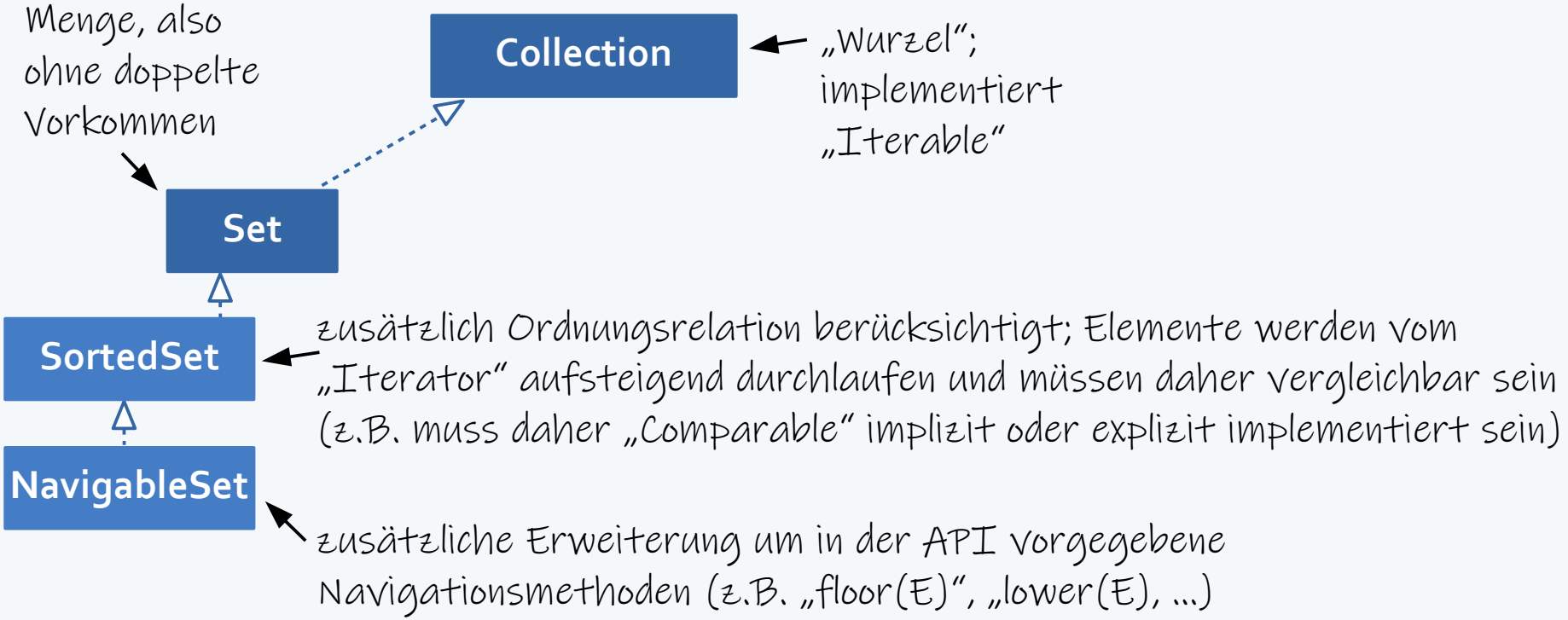
Collection ← „Wurzel“; implementiert „Iterable“

Anmerkungen

- „Iterable“ ermöglicht z.B. die Erstellung eines „Iterators“, eines „Splitarators’“ und die „default“-Methode „forEach“
- ein „Iterator“ erlaubt die Iteration (das Durchlaufen) durch die Elemente einer Quelle (z.B. durch Elemente einer „Collection“)
- ein „Spliteratur“ erlaubt, wie ein „Iterator“, die Iteration durch Elemente einer Quelle, unterstützt aber neben einer sequentiellen Iteration auch eine effiziente parallele Iteration, indem neben der Iteration auch die Zerlegung unterstützt wird

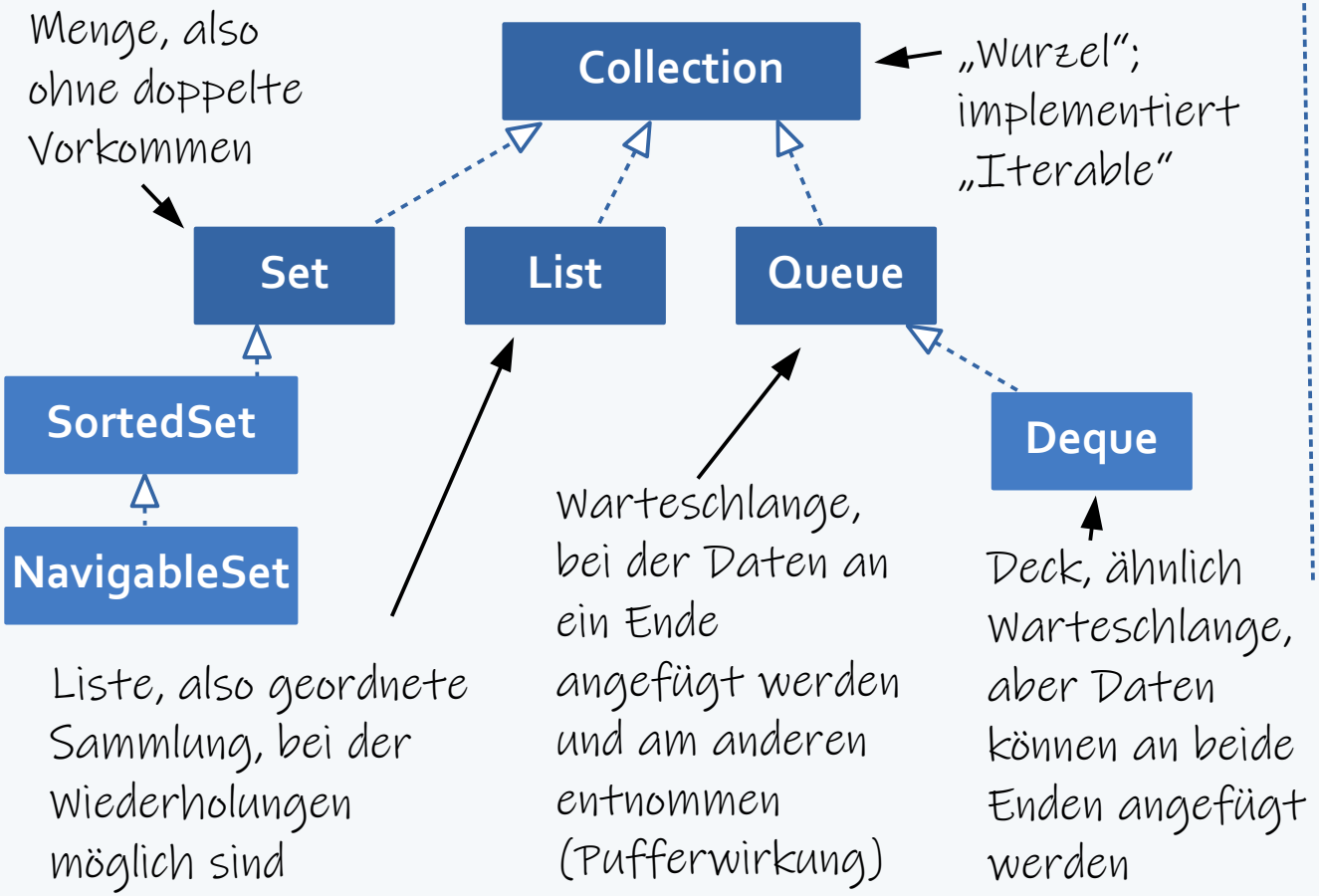
Java Collections Framework

wichtige Schnittstellen im „Java Collections Framework“

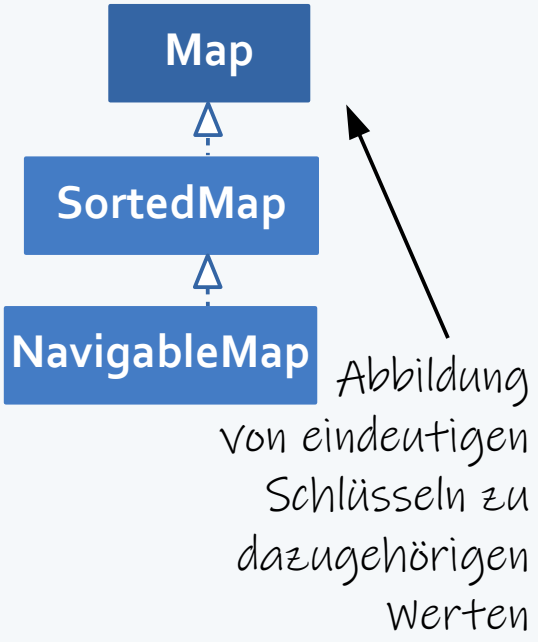


Java Collections Framework

wichtige Schnittstellen im „Java Collections Framework“

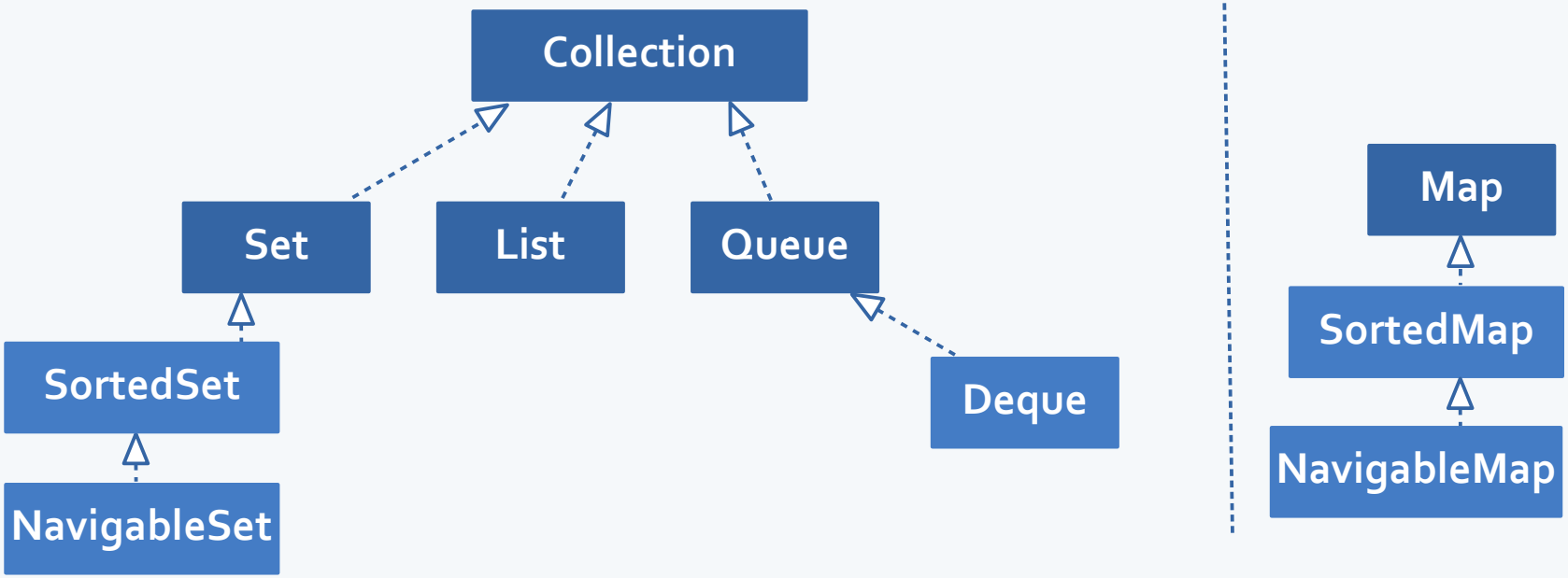


extra, da nur Schnittstelle im Collection Framework, aber keine Java-Collection



Schnittstellen sind generisch „<E>“ definiert, beim Erzeugen muss ein Referenztyp angegeben werden. (sehen wir später im Semester ...)

Java Collections Framework



Implementierungen der Schnittstellen

- Datenstrukturen, die Sie als Objekt erzeugen können (wobei Sie wissen, dass Sie viele praktische Methoden in den Schnittstellen dazu finden)
- finden Sie alles im Paket „java.util“:
  https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/package-summary.html
- **Beispiele:** ArrayList, LinkedList, HashSet, TreeSet, HashMap, ArrayDeque, …

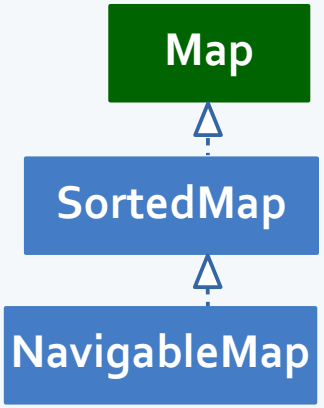
Wir schauen uns an, welche es gibt, und betrachten einige genauer …

Java Collections Framework

Map

Schnittstelle „Map“

- „Collection“, bei der jeder Eintrag aus einem „Schlüssel(-objekt)“ (key) und einem dazugehörigem „Wert(-objekt)“ (value) besteht, wobei jeder „Schlüssel“ nur einmal vorkommen darf
- wichtige Operationen sind „put(Schlüsselobjekt, Wertobjekt)“ zum Einfügen der Paare und „get(Schlüsselobjekt)“ zum Auslesen der Werte
- repräsentiert in der Informatik einen „Assoziativen Speicher“ und modelliert mathematisch damit eine eindeutige Abbildung (Funktion)



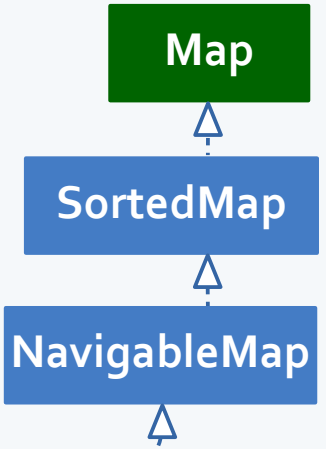
- **Implementierungen:**
  AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, Headers, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap

Java Collections Framework

Map

Schnittstelle „Map“ – Zwei allgemein besonders wichtige Klassen

- **„HashMap“:**
  - setzt nur korrekte Implementierung von „equals()“ und „hashCode()“ für die Schlüsselobjekte *(denn diese müssen „hashbar“ sein)* voraus *(was eigentlich nicht problematisch ist, weil selbst „Object“ diese Methoden schon implementiert, so dass sich eigentlich alle Klassen für Schlüsselobjekte eignen)*
  - schnelle Implementierung hinsichtlich Zugriff
- **„TreeMap“**
  - setzt Sortierbarkeit der Schlüsselobjekte voraus (wobei „compareTo()“ dann auch für die Definition von „gleich“ benutzt wird)
  - Schlüsselobjekte werden dementsprechend sortiert gehalten *(was damit anders als bei einer „HashMap“ null-Schlüssel ausschließt)*
  - beim Zugriff etwas langsamer als „HashMap“, dafür effiziente Iteration in Sortierreihenfolge möglich



Java Collections Framework

Map

Schnittstelle „Map“ – Implementierung „HashMap“ am Beispiel

```
package de.baleipzig.classes;
import java.util.*;
public class TestMap {
  public static void main(String[] args) {
    Map<String,String> emailAlias = new HashMap<>();
    emailAlias.put("Picard", "captain@sternenflotte.erde");
    emailAlias.put("Worf", "klingone@mail.kronos");
    emailAlias.put("Guinan", "info@zehn-vorne.schiff");
    emailAlias.put("Troi", "gedanken@ueberall.all");
    System.out.println("Guinan: " + emailAlias.get("Guinan"));
  }
}
```

Aufgabe 1

a) Fügen Sie noch ein weiteres Schlüssel-Wert-Paar hinzu.
b) Nutzen Sie die Methode „replace“, um die E-Mail-Adresse für Guinan auf „info@guinan.schiff“ zu ändern.
c) Lassen Sie sich mittels „values()“ alle E-Mail-Adressen anzeigen.

Java Collections Framework

Map

Lösung 1

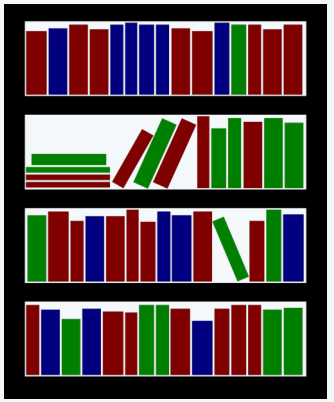
```
package de.baleipzig.classes;
import java.util.*;
public class TestMap {
  public static void main(String[] args) {
    Map<String,String> emailAlias = new HashMap<>();
    emailAlias.put("Picard", "captain@sternenflotte.erde");
    emailAlias.put("Worf", "klingone@mail.kronos");
    emailAlias.put("Guinan", "info@zehn-vorne.schiff");
    emailAlias.put("Troi", "gedanken@ueberall.all");
    System.out.println("Guinan: " + emailAlias.get("Guinan"));

    emailAlias.put("Data", "mensch@maschine.schiff");
    emailAlias.replace("Guinan","info@guinan.schiff");
    System.out.println(emailAlias.values());
  }
}
```

Java Collections Framework

Map

Schnittstelle „Map“ – Implementierung „TreeMap“ am Beispiel

```
package de.baleipzig.classes;
import java.util.*;
public class TestMap {
  public static void main(String[] args) {
    NavigableMap<Integer, String> shelfContent
      = new TreeMap<>();
    shelfContent.put(3, "Computer Programming Books");
    shelfContent.put(2, "Web Technology Books");
    shelfContent.put(1, "Data Processing Books");
    shelfContent.put(4, "Science Fiction");
  }
}
```



Aufgabe 2

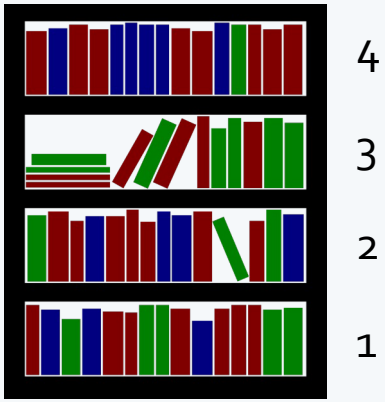
Bestimmen Sie mit Hilfe der Methode „size()“ die Anzahl der in „shelfContent“ enthaltenen Paare und weisen Sie diese Anzahl einer neuen Variablen zu.

Java Collections Framework

Map

Schnittstelle „Map“ – Lösung 2

```
package de.baleipzig.classes;
import java.util.*;
public class TestMap {
  public static void main(String[] args) {
    NavigableMap<Integer, String> shelfContent
      = new TreeMap<>();
    shelfContent.put(3, "Computer Programming Books");
    shelfContent.put(2, "Web Technology Books");
    shelfContent.put(1, "Data Processing Books");
    shelfContent.put(4, "Science Fiction");

    int shelfSize = shelfContent.size();
  }
}
```

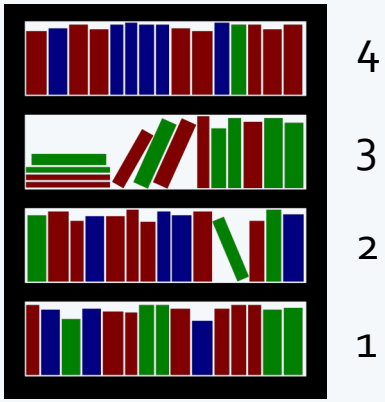


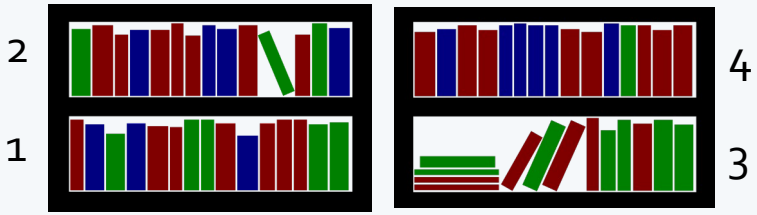
Java Collections Framework

Map

Schnittstelle „Map“ – Implementierung „TreeMap“ am Beispiel

```
package de.baleipzig.classes;
import java.util.*;
public class TestMap {
  public static void main(String[] args) {
    NavigableMap<Integer, String> shelfContent
      = new TreeMap<>();
    shelfContent.put(3, "Computer Programming Books");
    shelfContent.put(2, "Web Technology Books");
    shelfContent.put(1, "Data Processing Books");
    shelfContent.put(4, "Science Fiction");

    int shelfSize = shelfContent.size();
  }
}
```





Aufgabe 3

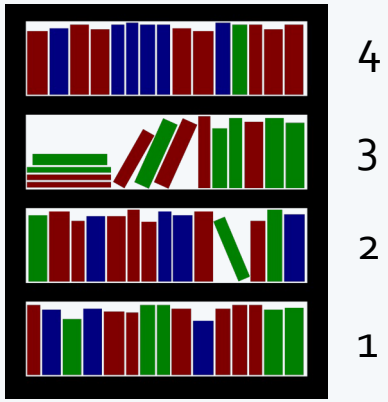
Nutzen Sie neben der üblichen Anzeige-methode die eben erstellte Variable und die Methode „subMap“, um sich die erste und die zweite Hälfte der Paare getrennt voneinander anzeigen zu lassen.

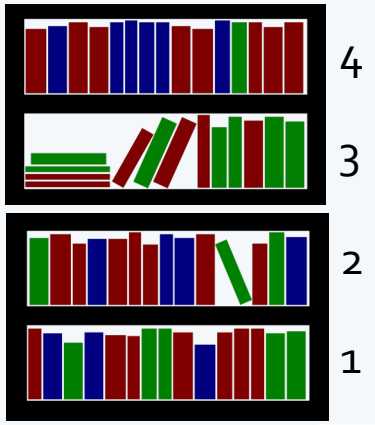
Java Collections Framework

Map

Schnittstelle „Map“ – Lösung 3

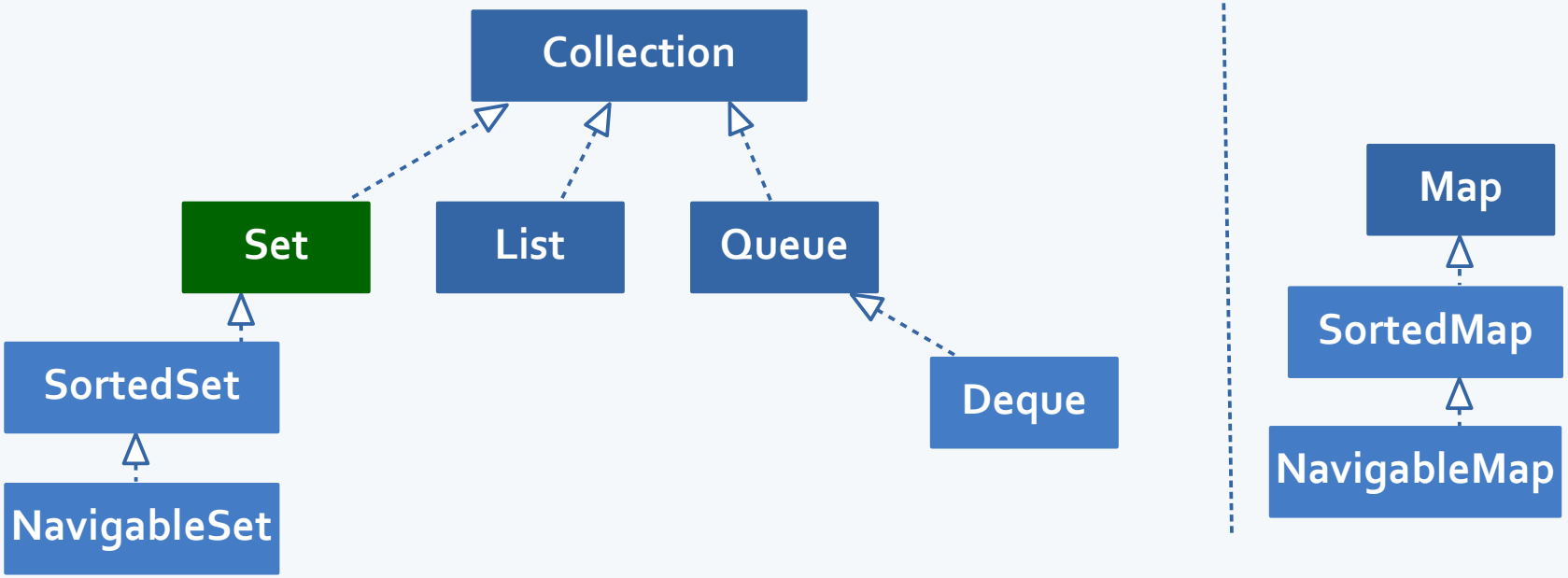
```
package de.baleipzig.classes;
import java.util.*;
public class TestMap {
  public static void main(String[] args) {
    NavigableMap<Integer, String> shelfContent
      = new TreeMap<>();
    shelfContent.put(3, "Computer Programming Books");
    shelfContent.put(2, "Web Technology Books");
    shelfContent.put(1, "Data Processing Books");
    shelfContent.put(4, "Science Fiction");

    int shelfSize = shelfContent.size();

    System.out.println(shelfContent
      .subMap(0, (shelfSize/2) + 1));

    System.out.println(shelfContent
      .subMap((shelfSize/2) + 1 , shelfSize + 1));
  }
}
```





Java Collections Framework

Set



Schnittstelle „Set“

- „Collection“, die keine doppelten Elemente enthält („doppelt“ im Sinne von „equals“, also keine Elemente e1 und e2, für die „e1.equals(e2))“ wahr ist)
- folgt damit dem Begriff der „Menge“ in der Mathematik
- **Implementierungen:**
  AbstractSet, ConcurrentHashMap.KeySetView, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet, JobStateReasons, LinkedHashSet, TreeSet

Java Collections Framework

Set

Schnittstelle „Set" - Beispielimplementierung „EnumSet"

- Set-Implementierung für die Verwendung mit Aufzählungstypen
- alle Elemente müssen von einem einzigen Enum-Typ stammen, der explizit oder implizit bei der Erstellung des Sets angegeben wird
- null-Elemente sind nicht erlaubt
- Elemente werden von einem Iterator in der Reihenfolge, in der die Enum-Konstanten deklariert sind, durchlaufen
- ist nicht synchronisiert; greifen mehrere teilweise auch schreibende Threads gleichzeitig auf ein EnumSet zu, muss es extern synchronisiert werden

Aufgabe 4
Erstellen Sie in Ihrem Paket, indem Ihr Planeten-„Enum" liegt, eine Klasse „TestSet" samt main-Methode und fügen Sie eine passende Import-Anweisung hinzu, sodass wir in der Klasse einfach „EnumSet" benutzen können.

Java Collections Framework

Set

Schnittstelle „Set" - Beispielimplementierung „EnumSet"

Lösung 4

Erstellen Sie in Ihrem Paket, indem Ihr Planeten-„Enum" liegt, eine Klasse „TestSet" samt main-Methode und fügen Sie eine passende Import-Anweisung hinzu, sodass wir in der Klasse einfach „EnumSet" benutzen können.

```
package de.baleipzig.classes;
import java.util.EnumSet;
public class TestSet {
  public static void main(String[] args) {

  }
}
```

Java Collections Framework

Set

Schnittstelle „Set" - Beispielimplementierung „EnumSet"

Dokumentation von „EnumSet"
https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/EnumSet.html

Methoden zur Erzeugung eines „EnumSet" – siehe Dokumentation

- allOf(...)
- clone()
- complementOf(...)
- copyOf(...)
- noneOf(...)
- of(...)
- range(...)

```
EnumSet<Planet> innerPlanets, outerPlanets, allPlanets;

innerPlanets = EnumSet.of(Planet.MERCURY, Planet.VENUS);
outerPlanets = EnumSet.complementOf(innerPlanets);
allPlanets = EnumSet.allOf(Planet.class);
```

Aufgabe 5: Ich habe schon alle Planeten von Merkur bis Venus besucht. Nutzen Sie in Ihrer Klasse „TestSet" in der main-Methode ein „EnumSet" und „range()", um dies zu erfassen. Lassen Sie sich zur Kontrolle das Set auf der Konsole ausgeben.

Java Collections Framework

Set

Schnittstelle „Set" - Beispielimplementierung „EnumSet"

Lösung 5

```
package de.baleipzig.classes;
import java.util.EnumSet;
public class TestSet {
  public static void main(String[] args) {
    EnumSet<Planet> visitedPlanets;
    visitedPlanets = EnumSet.range(Planet.MERCURY,Planet.VENUS);
    System.out.println("Visited planets: " + visitedPlanets);
  }
}
```

Aufgabe 6

Während Sie getippt haben, habe ich kurz den Eisgiganten „Neptun" besucht. Nutzen Sie eine für EnumSet/Set schon vorhandene Methode, um Neptun mit in die Menge meiner besuchten Planeten aufzunehmen, und machen Sie anschließend wieder eine Kontrollausgabe auf der Konsole.

Java Collections Framework

Set

Schnittstelle „Set“ - Beispielimplementierung „EnumSet“

Lösung 6

```
package de.baleipzig.classes;
import java.util.EnumSet;
public class TestSet {
  public static void main(String[] args) {
    EnumSet<Planet> visitedPlanets;
    visitedPlanets = EnumSet.range(Planet.MERCURY,Planet.VENUS);
    System.out.println("Visited planets: " + visitedPlanets);

    visitedPlanets.add(Planet.NEPTUNE);
    System.out.println("Visited planets: " + visitedPlanets);

  }
}
```

Java Collections Framework

Set

Schnittstelle „Set" – Beispielimplementierung „TreeSet"

- Set-Implementierung, die „NavigableSet" implementiert
- basiert auf einer „TreeMap"
- setzt Sortierbarkeit der Elemente voraus (wobei „compareTo()" dann auch für die Definition von „gleich" benutzt wird)
- Elemente werden dementsprechend sortiert gehalten (was damit null-Elemente ausschließt)
- ist nicht synchronisiert; greifen mehrere teilweise auch schreibende Threads gleichzeitig auf ein TreeSet zu, muss es extern synchronisiert werden

Java Collections Framework

Set

Schnittstelle „Set“ – Beispielimplementierung „TreeSet“

```
package de.baleipzig.classes;
import java.util.EnumSet;
import java.util.TreeSet;
public class TestSet {
  public static void main(String[] args) {
    //...
    TreeSet<Integer> primeNumbers = new TreeSet<>();
    primeNumbers.add(2);
    primeNumbers.add(3);
    primeNumbers.add(4);
    primeNumbers.add(5);
    System.out.println("Prime Numbers: " + primeNumbers);
  }
}
```

Aufgabe 7

Nutzen Sie die Methode „remove“, um die fälschlicher Weise enthaltene Nicht-Primzahl 4 aus der Menge zu entfernen.

Java Collections Framework

Set

Schnittstelle „Set“ – Lösung 7

```
package de.baleipzig.classes;
import java.util.EnumSet;
import java.util.TreeSet;
public class TestSet {
  public static void main(String[] args) {
    //...
    TreeSet<Integer> primeNumbers = new TreeSet<>();
    primeNumbers.add(2);
    primeNumbers.add(3);
    primeNumbers.add(4);
    primeNumbers.add(5);
    System.out.println("Prime Numbers: " + primeNumbers);
    primeNumbers.remove(4);
  }
}
```

Java Collections Framework

Set

Schnittstelle „Set“ – Aufgabe 8

„TreeSet<E>“ implementiert „Iterable<E>“.
Wir wollen daher versuchsweise einen Iterator nutzen, um uns unsere Primzahlen anzeigen zu lassen.

```
//...
primeNumbers.remove(4);

Iterator<Integer> primeNumberIterator = primeNumbers.iterator();
System.out.print("Prime Numbers using Iterator: ");

while(primeNumberIterator.hasNext()) {
    System.out.print(primeNumberIterator.next() + " ");
}
```

Java Collections Framework

Set

Schnittstelle „Set“ – Aufgabe 8

„TreeSet<E>“ implementiert „Iterable<E>“.
Wir wollen daher versuchsweise einen Iterator nutzen, um uns unsere Primzahlen anzeigen zu lassen.

```
//...
primeNumbers.remove(4);

Iterator<Integer> primeNumberIterator = primeNumbers.iterator();
System.out.print("Prime Numbers using Iterator: ");

while(primeNumberIterator.hasNext()) {
    System.out.print(                 + " ");
}
```

Sorgen Sie durch einen geeigneten „import“ dafür, dass Sie „Iterator“ überhaupt benutzen können. Suchen Sie anschließend in der Dokumentation der Schnittstelle „Iterator“ eine Methode, die das nächste Element in der Iteration liefert, und nutzen Sie diese in obiger Ausgabemethode zur Primzahlenanzeige.

Java Collections Framework

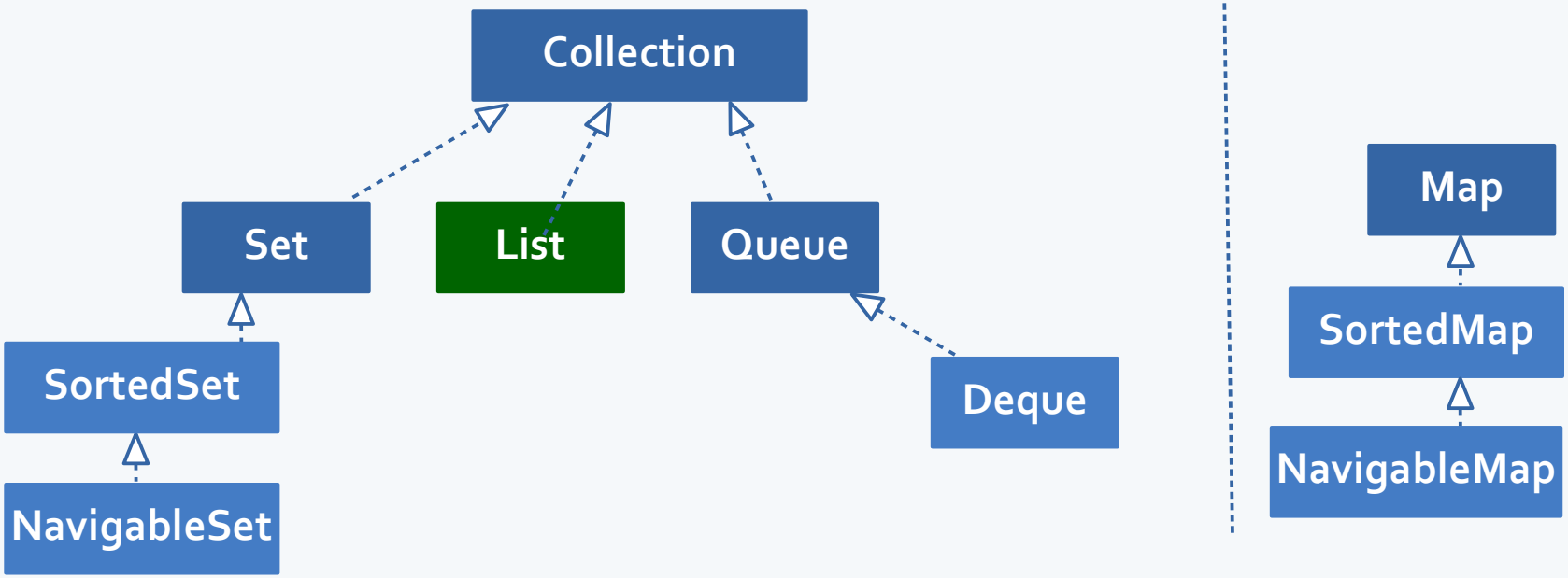
Set

Schnittstelle „Set“ – Lösung 8

```
package de.baleipzig.classes;

import java.util.EnumSet;
import java.util.TreeSet;
import java.util.Iterator;

public class TestSet {

  public static void main(String[] args) {
    // ...
    Iterator<Integer> primeNumberIterator = primeNumbers.iterator();
    System.out.print("Prime Numbers using Iterator: ");

    while(primeNumberIterator.hasNext()) {
      System.out.print(primeNumberIterator.next() + " ");
    }
  }

}
```

Java Collections Framework

List



Schnittstelle „List“

- „Collection“, bei der Elemente wunschgemäß geordnet enthalten sind
- Elemente, auch null-Elemente, können mehrfach enthalten sein
- repräsentiert damit mathematische „Tupel“
- **Implementierungen:**
  AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

Java Collections Framework

List

Schnittstelle „List" – Einige Klassen im Vergleich (nicht klausurrelavant)

- **ArrayList** (benötigte Größe sollte vorab bedacht werden)
  - ist als „größenveränderlicher" Array implementiert (was gegebenenfalls dazu führt, dass bei Vergrößerung der bisherige Array komplett in einen neuen kopiert werden muss; Array-Größe wird bei Bedarf jeweils um 50% erhöht)
  - „get" und „set" sind übliche Zugriffsmethoden
- **LinkedList**
  - ist als doppelt verkettete Liste implementiert
  - damit sind das Einfügen und Entfernen von Elementen im Vergleich zur ArrayList problemloser und meist schneller realisierbarer, wohingegen „get" und „set" im Vergleich meist aufwendiger sind
  - implementiert auch Deque und Queue
- **Vector** (benötigte Größe sollte vorab bedacht werden)
  - ist vergleichbar zu „ArrayList" (wobei Array-Größe aber bei Bedarf verdoppelt wird), aber synchronisiert (bei thread-sicheren Programmen ist also ArrayList aufgrund des dort weniger betriebenen Synchronisierungsaufwandes die bessere Wahl)

→ **bei Optimierungsbedarf ggf. eigenen Implementierungen vorteilhafter**

Java Collections Framework

List

Schnittstelle „List“ – Beispielimplementierung „ArrayList“

```
package de.baleipzig.classes;
import java.util.ArrayList;
import java.util.Iterator;
public class TestList {
  public static void main(String[] args) {
    ArrayList<String> destination = new ArrayList<>();
    destination.add("Leipzig");
    destination.add("Berlin");
    destination.add("Rostock");
    destination.add("Leipzig");
    destination.add("Frankfurt");
    destination.add("Hamburg");
    destination.add("Leipzig");

    for (int i = 0;i<destination.size();i++){
      System.out.println(destination.get(i));
    }
  }
}
```

Aufgabe 9

Nutzen Sie die Methode „remove“, um einmal „Leipzig“ zu entfernen.

Java Collections Framework

List

Schnittstelle „List“ – Lösung 9

```
package de.baleipzig.classes;
import java.util.ArrayList;
import java.util.Iterator;
public class TestList {
  public static void main(String[] args) {
    ArrayList<String> destination = new ArrayList<>();
    //...
    for (int i = 0;i<destination.size();i++){
      System.out.println(destination.get(i));
    }
    destination.remove("Leipzig");
  }
}
```

Aufgabe 10

Versuchen Sie analog zu Aufgabe 8 einen Iterator zur Ausgabe der Listen-Elemente zu nutzen.

Java Collections Framework

List

Schnittstelle „List“ – Lösung 10

```
package de.baleipzig.classes;
import java.util.ArrayList;
import java.util.Iterator;
public class TestList {
  public static void main(String[] args) {
    ArrayList<String> destination = new ArrayList<>();
    //...
    for (int i = 0;i<destination.size();i++){
      System.out.println(destination.get(i));
    }
    destination.remove("Leipzig");

    Iterator<String> destinationIterator = destination.iterator();
    System.out.print("Destinations: ");

    while(destinationIterator.hasNext()) {
      System.out.print(destinationIterator.next() + " ");
    }
  }
}
```

Java Collections Framework

List

Schnittstelle „List“ – Lösung 10

```
package de.baleipzig.classes;
import java.util.ArrayList;
import java.util.Iterator;
public class TestList {
  public static void main(String[] args) {
    ArrayList<String> destination = new ArrayList<>();
    //...

    Iterator<String> destinationIterator = destination.iterator();
    System.out.print("Destinations: ");

    while(destinationIterator.hasNext()) {
      System.out.print(destinationIterator.next() + " ");
    }
  }
}
```

Aufgabe 11

Wenn viele Städte aufgenommen werden sollen, ist es ungünstig, wenn dadurch viele „Array-werd-größer-Kopiervorgänge“ nötig sind. Versuchen Sie, die Listenlänge von Anfang an auf 1000 Städte festzulegen.

Java Collections Framework

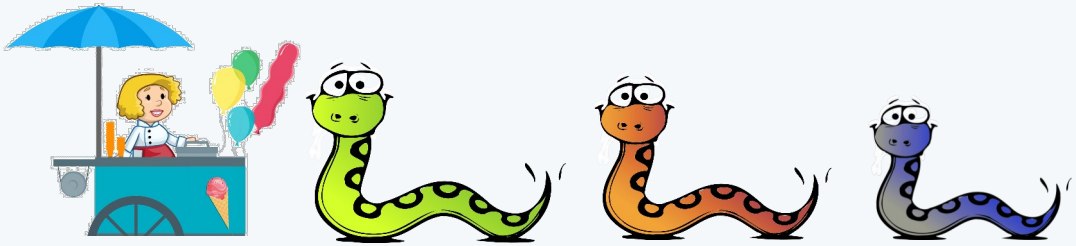
List

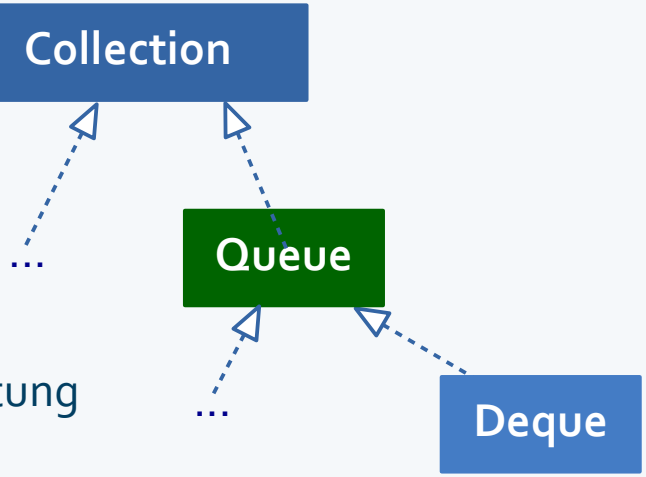
Schnittstelle „List“ – Lösung 11

```
package de.baleipzig.classes;
import java.util.ArrayList;
import java.util.Iterator;
public class TestList {
  public static void main(String[] args) {
    ArrayList<String> destination = new ArrayList<>(1000);
    //...

    Iterator<String> destinationIterator = destination.iterator();
    System.out.print("Destinations: ");

    while(destinationIterator.hasNext()) {
      System.out.print(destinationIterator.next() + " ");
    }
  }
}
```

Java Collections Framework

Queue



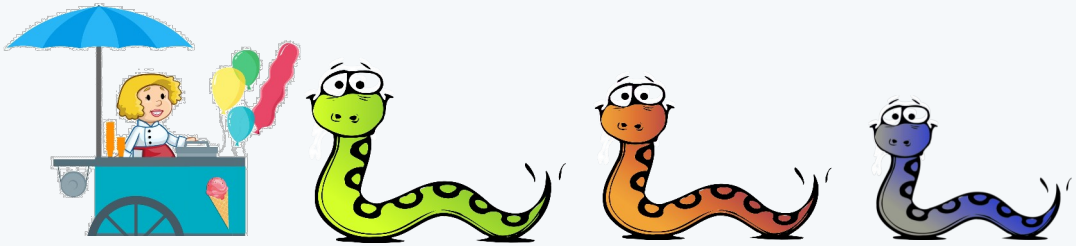


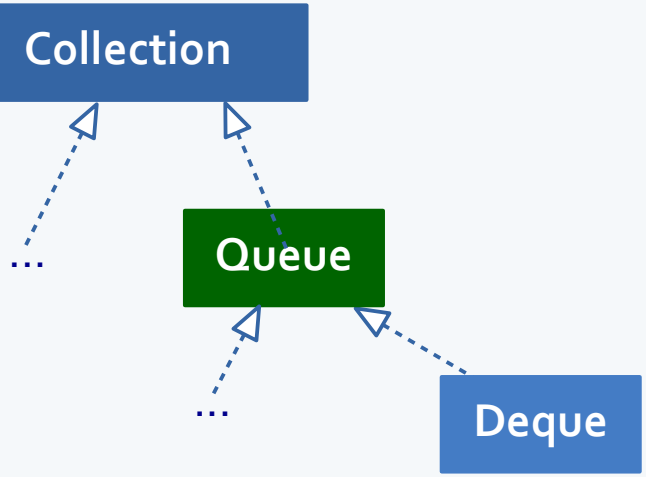
Schnittstelle „Queue“

- „Collection“, mit der Elemente vor der (Weiter-)Verarbeitung in einer Warteschlange gespeichert werden können
- meist sind Kopf und Ende der Warteschlange an entgegengesetzten Enden, wobei aber auch Implementierungen möglich sind, bei denen sich der Kopf und das Ende der Warteschlange am selben Ende befinden („Stack“ [Stapel])
- null-Elemente sind nicht erwünscht *(da es zu Verwechslungen mit dem Rückgabewert „null“ für eine leere Warteschlange kommen kann)*
- repräsentiert auch „Tupel“, jedoch mit anderer Intention als bei einer „List“
- **Implementierungen:**
  AbstractQueue, ArrayBlockingQueue, ArrayDeque, ConcurrentLinkedDeque, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, SynchronousQueue

Java Collections Framework

Queue





Schnittstelle „Queue" – Eigene Methoden

- zusätzliche Einfüge-, Entnahme- und Prüfoperationen, jeweils in zwei Varianten:
  - Variante, die beim Fehlschlagen der geforderten Operation eine Ausnahme wirft
  - Variante, die beim Fehlschlagen der geforderten Operation (sinnvoll z.B. beim Einfügeversuch bei einer kapazitätsbeschränkten Warteschlangen-Implementierung) einen speziellen Rückgabewert (null oder false, abhängig von der Operation) liefert

	wirft Ausnahme	spez. Rückgabewert
einfügen	add(e)	offer(e)
entnehmen	remove()	poll()
prüfen	element()	peek()

Java Collections Framework

List

Schnittstelle „Queue“ – Beispielimplementierung „LinkedList“

Aufgabe 12

Schreiben Sie eine Klasse „TestQueue“ und implementieren Sie in der main-Methode dieser Klasse folgenden Sachverhalt:

- Deklarieren Sie eine „Queue<String>“ namens „canteenQueue“ und initialisieren Sie diese mit einer neuen „LinkedList“.
- Fügen Sie dieser Warteschlange die hungrigen Gäste „Miss Marple“, „Sherlock Holmes“ und „Hercule Poirot“ hinzu.
- Geben Sie die Warteschlange auf der Konsole aus.
- Vervollständigen Sie die folgenden Konsolenausgaben durch sinnvolle Befehle:

```
System.out.println("first person served: " + [redacted]);
System.out.println("now head of the queue:" + [redacted]);
System.out.println("next person served: " + [redacted]);
System.out.println("now head of the queue: " + [redacted]);
System.out.println("Is waiting queue empty? : " + [redacted]);
System.out.println("size of waiting queue : " + [redacted]);
System.out.println("final queue:"+ [redacted]);
```

Nutzen Sie bei diesen letzten Konsolenausgaben keine Methode mehrfach.

Java Collections Framework

List

Schnittstelle „Queue“ – Lösung 12 „LinkedList“

```
package de.baleipzig.classes;
import java.util.LinkedList;
import java.util.Queue;
public class TestQueue {
  public static void main(String[] args) {
    Queue<String> canteenQueue = new LinkedList<>();
    canteenQueue.add("Miss Marple");
    canteenQueue.add("Sherlock Holmes");
    canteenQueue.add("Hercule Poirot");
    System.out.println("waiting queue : " + canteenQueue);
    System.out.println("first person served: " + canteenQueue.remove());
    System.out.println("now head of the queue: " + canteenQueue.element());
    System.out.println("next person served: " + canteenQueue.poll());
    System.out.println("now head of the queue: " + canteenQueue.peek());
    System.out.println("is waiting queue empty? : " + canteenQueue.isEmpty());
    System.out.println("size of waiting queue : " + canteenQueue.size());
    System.out.println("final queue:" + canteenQueue);
  }
}
```

Vielen Dank für Ihre Aufmerksamkeit.