hhu.

NKT
Tag 1: Idiomatische Schleifen, Klassenbibliothek, Funktionales Java und Generics

Sven Hoops, Paul Dötsch

15.09.2025

Gliederung

Was ist idiomatisch?

Heinrich Heine Universität Düsseldorf

Definition von Google Gemini:

"Der Begriff Idiomatisch bedeutet, dass ein Programmierer Quellcode schreibt, der den Konventionen, Best Practices und dem natürlichen Stil einer bestimmten Programmiersprache oder eines Frameworks entspricht."

- Konventionen
- Best Practices
  - von Programmiersprache vorgeschrieben
  - von Framework vorgeschrieben

Idiomatische Schleifen

Heinrich Heine Universität Düsseldorf

Folgende Schleifen werden wir berachten:

- For Loop
- Enhanced For Loop
- While Loop
- Do While Loop
- For Method?[1]

Aber wie entscheiden wir nun was Best Practice ist?

[1] Hä das gibts als Methode?

Idiomatische Schleifen auswählen

Heinrich Heine Universität Düsseldorf

- For Loop -> falls wir den index benutzen
- Enhanced For Loop -> falls eine Collection komplett durchgegangen werden soll
- While Loop -> falls wir nicht wissen wie häufig wir etwas wiederholen müssen bspw. lesen einer eingabe[2]
- For Method -> in zusammenhang mit Streams[3]

[2] fehlt hierfür vielleicht noch ein genau so idiomatischer weg?
[3] hierzu später mehr

Idiomatisch?

Ist folgender code Idiomatisch?

```
1    public static void printList(List<Integer> list) {
2                for (int i = 0; i < list.size(); i++) {
3                        System.out.print(list.get(i) + "␣");
4                }
5                System.out.println();
6        }
```

Sollten wir diesen ändern?

Idiomatisch?

Heinrich Heine Universität Düsseldorf

Ist folgender code Idiomatisch?

```
1    public static int sumArray(List<Integer> list) {
2                int count = 0;
3                int summe = 0;
4                while (count < list.size()) {
5                        summe += list.get(count);
6                        count++;
7                }
8                return summe;
9        }
```

Sollten wir diesen ändern?

Idiomatisch?

Ist folgender code Idiomatisch?

```
1   public static boolean contains(List<String> list, String toFind
        ) {
2               boolean gefunden = false;
3               for (String element : list) {
4                       if (element.equals(toFind)) {
5                               gefunden = true;
6                               break;
7                       }
8               }
9               return gefunden;
10      }
```

Sollten wir diesen ändern?

Klassenbibliothek

Heinrich Heine
Universität Düsseldorf

Die Java Klassenbibliothek bietet fertige Datenstrukturen. Wir sprechen heute über:

- Collection
- List
- Queue
- Set
- Map
- Optionals

Aber Achtung Dies sind alles nur Interfaces[4]

[4] sehr häufiger Fehlerin der Klausur

Klassenbibliothek Falsch genutzt

Folgendes wird also nicht funktionieren:

```
1    List<String> strings = new List("eins","zwei");
```

Interfaces können nicht erstellt werden
↪ Ein Objekt kann nur **Instanz** eines Interfaces sein

Klassenbibliothek korrekt genutzt

Die Klasse ArrayList implementiert das List interface. Daher funktioniert folgendes:

```
1    List<String> strings = new ArrayList(List.of("eins","zwei"));
2    strings.addAll(strings);
```

[5] Wir können auch eigene Klassen erstellen, die das List Interface Implementieren, oder die von Klassen erben die dies tun[6], und diese dann als List Objekt speichern

[5] List.of(...) erstellt *immutable* List

[6] sinnvollerer weg, wenn man nicht alles neu implementieren will

Klassenbibliothek Interface Aufgabe

Heinrich Heine
Universität Düsseldorf

Schreiben sie eine Rotating List Klasse:

- wenn bei get(int index) ein index eingegeben wird der zu hoch ist soll dieser korrigiert. Bsp.: liste [a][b][c][d], index=5 -> b, index=4 ->a
- fügen sie eine methode rotate(int x) hinzu. Bsp.: x=2 [a][b][c][d] -> [c][d][a][b]

Tipp: benutzen sie eine ArrayList

Welche Klasse welche methode?

- Hizufügen
  - List -> add, addAll(Collection<?>)
  - Set -> add, addAll(Collection<?>)
  - Map -> put, putAll(Map<?>), putIfAbsent(<K,V>)
- Löschen
  - List -> remove, removeAll(Collection<?>)
  - Set -> remove, removeAll(Collection<?>), removeIf(Predicate<?>)
  - Map -> remove(K)

7

[7] K=key,V=value

Optionals

Optionals sind Container, die Werte enthalten können, aber nicht müssen

```
1        Optional<String> optional1 = Optional.empty();
2        Optional<String> optional2 = Optional.of("value");
3        Map<Boolean,String> map = new HashMap<>();
4        map.put(optional1.isPresent(),optional1.orElseGet(() -> "
             namenslos"));
5        map.put(optional2.isPresent(),optional2.get());
6        map.entrySet().stream().forEach(System.out::println);
```

Was wird hier die Ausgabe sein?

Frage

Sollen wir noch Weitere Aufgaben zu der Klassenbibliothek machen? Bspw. zu Maps?

Beispiel

Stellen wir uns vor, wir möchten Bücher sortieren.

```
1  void sortBooksAuthor(Book[] books) {
2    boolean changed = false;
3    do {
4      for (int i = 1; i < books.length(); i++){
5          if (books[i - 1].author().compareTo(books[i].author()) < 0)
               {
6              var swap = book[i-1];
7              book[i-1] = book[i];
8              book[i] = swap;
9              changed = true;
10         }
11     }
12   } while (changed)}
```

Beispiel

Jetzt möchte der Bibliothekar nach Farben sortieren.

```
1  void sortBooksColor(Book[] books) {
2    boolean changed = false;
3    do {
4      for (int i = 1; i < books.length(); i++){
5          if (books[i - 1].color().compareTo(books[i].color()) < 0) {
6              var swap = book[i-1];
7              book[i-1] = book[i];
8              book[i] = swap;
9              changed = true;
10         }
11     }
12   } while (changed)}
```

Beispiel

Wir merken, dass der code fast identisch ist.
Idee: wir führen ein externes Interface Comparator<T> ein

```
1 public class BookAuthorComparator implements Comparator<Book> {
2     public int compare(Book firstBook, Book secondBook) {
3         return firstBook.author().compareTo(secondBook.author());
4     }
5 }
```

Nun müssen muss dies nur der sortBooks methode als parameter übergeben werden

```
1     void sortBooks(Book[] books, Comparator<Book> bookComparator)
          {...}
```

Funktionale Interfaces

Definition von Google Gemini:
Ein funktionale Interface ist eine Schnittstelle mit genau einer abstrakten Methode und kann beliebig viele Default-Methoden und statische Methoden enthalten.

```
1  @FunctionalInterface
2  interface Comparator<T> {
3  int compare(T first, T second);
4  boolean equals(Object obj);
5  }
```

Ist dies also ein funktionales Interface?

Wasn das für nen Pfeil? Das ist Super Java!

Heinrich Heine Universität Düsseldorf

Lambda Expression sind wiefolgt aufgebaut: (parameters) -> expression
Bsp.:

```
1     sort(books, (first, second) ->
2     first.color().compareTo(second.color()));
```

Wenn wir ein funktionales Interface verwenden können wir auch Methodenreferenz verwenden:

```
1 map.entrySet().stream().forEach(entry->System.out.println(entry));
2 map.entrySet().stream().forEach(System.out::println);
```

Consumer

Heinrich Heine
Universität Düsseldorf

Consumer<T> hat keine Rückgabe. Wird verwendet in Stream forEach methode

```
1 Consumer<Test> doesWork = (Test toConsume) -> toConsume.method();
```

- besitzt void accept(T t) Methode

Supplier

Supplier<T> erstellen Objekte des Datentypes T

```
1 Supplier<String> supplier = () -> "supplier";
```

Beispielsweise benutzt von Stream.iterate()

Function

Heinrich Heine Universität Düsseldorf

Function<T, R> kann auf Objekt des Datentypes T angewendet werden und es wird ein Objekt des Datentypes R zurück gegeben

```
1 Function<String, Integer> function = string -> string.length();
```

Wird aufgerufen mit Methode apply(T)

```
1 int f = function.apply("test");
```

BiFunction

Heinrich Heine
Universität Düsseldorf

BiFunction<T,V,R> wird auf den elementen T und V aufgerufen und R ist der Rückgabetyp
Aufgabe aus der Klausur:
Schreiben sie eine generische, statische Methode, die zwei Listen und eine BiFunction entgegen nimmt. Die bifunction soll auf die Elemente der beiden Listen angewendet werden.
(Also Zunächst auf das erste Objekt von Liste a und auf das erste Objekt der Liste b)

Predicate

Predicate<T> nimmt das objekt von Datentyp T entgegen und gibt einen boolean zurück

```
1 Predicate<String> predicate = string -> string.startsWith("a");
```

Wird beispielsweise in stream().filter() benutzt

Generics

```java
public static void printStack(int[] array) {
    for (int i : array) {
        System.out.print(i + "␣");
    }
    System.out.println();
}
```

Klappt folgendes?:

```java
public static void main(String[] args) {
        Integer[] array = { 1, 2, 3, 8, 9, 10 };
        Double[] doubleArray = { 1.1, 2.2, 3.3, 6.6, 7.7 };
        printStack(array);
        printStack(doubleArray);
}
```

Generics

Heinrich Heine
Universität Düsseldorf

Schreiben sie die Methode printStack zu einer Generischen Methode um.