Java

Weitere Programmierkonzepte in Java

Marcus Köhler

14. Dezember 2018

Java-Kurs

Überblick

1. Casting
   - Was ist Casting?
   - Warum casten?
   - Guidelines
2. Rekursion
   - Was ist Rekursion?
   - Beispiel
3. Lambdas(Java 8+)
   - Lambdas?
   - Anwendungen von Lambdas
4. Übung

Casting

Was ist Casting?

Bei einigen Programmabläufen in Java ist es nötig, eine Referenz oder einen primitiven Datentypen in einen anderen Typ umzuwandeln. Hierzu gibt es in Java(wie auch in den meisten anden C-ähnlichen Sprachen) das sogenannte *Casting* bzw. *Typecasting*. Eine Variable bzw. Referenz wird gecastet, indem man den „Zieltyp“ in Klammern vor das Statement setzt, das konvertiert werden soll:

```
1 int dividend = 42;
2 int divisor = 8;
3
4 float res = (float)dividend/divisor;
5 System.out.println(res); //prints 5.0 because int-division is
      whole number division
```

Warum casten?

Gerade in der Verwendung von Polymorphie muss oft casting angewendet werden, wenn man mehr als die Methoden des Typs der Referenz verwenden will:

```
1  Object objReference = new ArrayList<Integer>();
2  //objReference can only access methods defined in Object
3  objReference.add(9); //won't compile
4
5  Collection<Integer> colReference = (Collection)objReference;
6  //colReference can also access the methods defined in Collection
7  colReference.listIterator(); //won't compile
8
9  List<Integer> listReference = (List)objReference;
10 //listReference can access all methods that are defined in List
11 listReference.add(5, 9); //adds 9 at index 5; works just fine
```

Warum casten?

Es ist auch möglich, nur einzelne Teile eines Statements zu casten, indem man den „zu castenden“ Teil(inkl. des Castings an sich) mit Klammern zusammenfasst:

```
1 int dividend = 42;
2 int divisor = 8;
3
4 float res = ((float)dividend)/divisor; // float/float division
      because of casting
5 System.out.println(res) // prints 5.25, the correct result
```

Guidelines für Casting

Beim Casting sollte man einige grundlegende Regeln beachten:

- Bei primitiven Datentypen funktioniert Casting nur, wenn dabei keine Informationen verloren gehen:
  `int -> float` ist erlaubt, `float -> int` nicht

Guidelines für Casting

Beim Casting sollte man einige grundlegende Regeln beachten:

- Bei primitiven Datentypen funktioniert Casting nur, wenn dabei keine Informationen verloren gehen:
  `int -> float` ist erlaubt, `float -> int` nicht
- Bei Objekten muss das „Quellobjekt“:

Guidelines für Casting

Beim Casting sollte man einige grundlegende Regeln beachten:

- Bei primitiven Datentypen funktioniert Casting nur, wenn dabei keine Informationen verloren gehen:
  `int -> float` ist erlaubt, `float -> int` nicht
- Bei Objekten muss das „Quellobjekt“:
  - eine Instanz des Zieltyps oder einer seiner Subklassen sein, falls der Zieltyp eine Klasse ist

Guidelines für Casting

Beim Casting sollte man einige grundlegende Regeln beachten:

- Bei primitiven Datentypen funktioniert Casting nur, wenn dabei keine Informationen verloren gehen:
  `int -> float` ist erlaubt, `float -> int` nicht
- Bei Objekten muss das „Quellobjekt“:
  - eine Instanz des Zieltyps oder einer seiner Subklassen sein, falls der Zieltyp eine Klasse ist
  - eine Instanz einer Klasse K(oder einer Subklasse von K) sein, wobei K das Interface implementiert, falls der Zieltyp ein Interface ist

Rekursion

Was ist Rekursion?

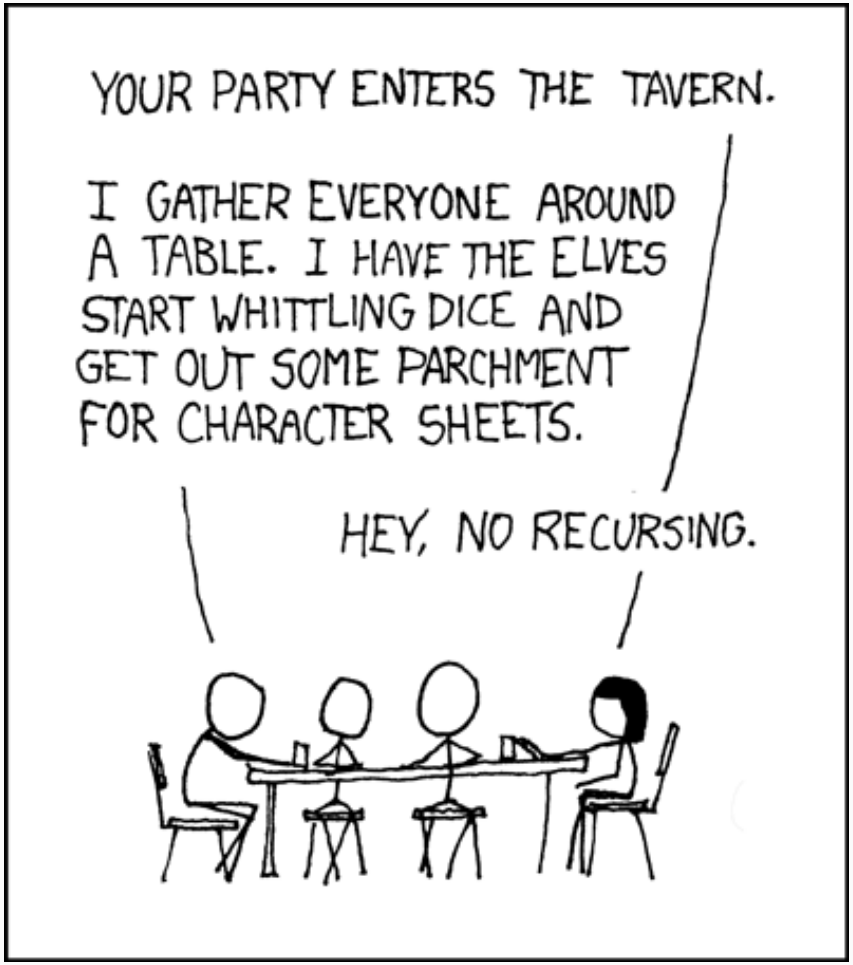
Rekursion bezeichnet eine Funktion bzw. eine Lösungsstrategie, die sich selbst wieder aufruft, meistens mit einem Subset der ursprünglichen Daten.

Dies passiert so oft, bis entweder ein sogenannter *Basecase* auftritt, für den eine Lösung bekannt ist, oder aber keine Lösung gefunden werden kann (oder eine `StackOverflowException` auftritt).

Was ist Rekursion?

Rekursion bezeichnet eine Funktion bzw. eine Lösungsstrategie, die sich selbst wieder aufruft, meistens mit einem Subset der ursprünglichen Daten.

Dies passiert so oft, bis entweder ein sogenannter *Basecase* auftritt, für den eine Lösung bekannt ist, oder aber keine Lösung gefunden werden kann (oder eine `StackOverflowException` auftritt).



https://xkcd.com/244

Rekursion:Beispiel

Ein klassisches Beispiel für Rekursion ist eine Funktion, die die n-te Zahl der Fibonacci-Folge berechnet:

```
1 public int fibonacci(int n) {
2     if(n < 0) throw new IllegalArgumentException();
3     if(n == 0 || n == 1) return 1; //define basecase
4     return fibonacci(n-1) + fibonacci(n-2); //recursive call
5 }
```

Rekursion:Beispiel

Ein klassisches Beispiel für Rekursion ist eine Funktion, die die n-te Zahl der Fibonacci-Folge berechnet:

```
1 public int fibonacci(int n) {
2     if(n < 0) throw new IllegalArgumentException();
3     if(n == 0 || n == 1) return 1; //define basecase
4     return fibonacci(n-1) + fibonacci(n-2); //recursive call
5 }
```

->DEMO<-

Lambdas(Java 8+)

Lambdas?

Lambdas sind Javas Implementierung von sogenannten *anonymen Funktionen*.
Im Klartext bedeutet das, dass man mittels Lambdas Funktionen als Parameter an Methoden übergeben kann.

Lambdas?

Lambdas sind Javas Implementierung von sogenannten *anonymen Funktionen*.
Im Klartext bedeutet das, dass man mittels Lambdas Funktionen als Parameter an Methoden übergeben kann.
Ein Lambda hat die Form `<Argument> -> <Funktion>`:

```
1 List<Integer> intList = new ArrayList<>();
2 intList.add(1);
3 intList.add(3);
4
5 Iterator<Integer> iter = intList.iterator();
6
7 iter.forEachRemaining(i -> System.out.print(i + " "));
8 //prints 1 3
```

Lambdas?

In einigen Fällen ist es notwendig, mehr als eine Funktion auf einmal auf dem Argument anzuwenden. Hierzu kann man auch mehrere Statements und das `return` Keyword verwenden. Dabei muss man die Funktion in packen:

```
1 //same setup as before
2 iter.forEachRemaining(i -> {int sq = i*i;
3                              System.out.print((sq+10));});
4 //prints 11 19
```

Dieser Code ist äquivalent zu folgendem:

```
1 private void printSquarePlusTen(int i) {
2     int sq = i*i;
3     System.out.println((sq+10));
4 }
5
6 iter.forEachRemaining(this::printSquarePlusTen)
7 //:: is the Method reference Operator
8 //this::printSquarePlusTen is equivalent to i -> printSquarePlusTen(i)
```

Lambdas!

Lambdas werden häufig in `Streams` eingesetzt. Streams sind eine neue Strategie(seit Java 8), über `Collections` zu iterieren:

```
1 List<String> stringList = new Linkedlist<>();
2 stringList.add(Arrays.asList("Peter", "Paul", "Petra")); //shorthand to
      add multiple elements at once
3 Stream<String> stringStream = stringList.stream(); //"streams" the
      contents of the list
4
5 stringStream = stringStream.filter(s -> s.length() == 5);
6 stringStream = stringStream.map(s -> s.replace('e', '@'));
7 stringStream.forEach(System.out::println); //prints P@t@r P@tra
```

Lambdas!

Lambdas werden häufig in `Streams` eingesetzt. Streams sind eine neue Strategie(seit Java 8), über `Collections` zu iterieren:

```
1 List<String> stringList = new Linkedlist<>();
2 stringList.add(Arrays.asList("Peter", "Paul", "Petra")); //shorthand to
      add multiple elements at once
3 Stream<String> stringStream = stringList.stream(); //"streams" the
      contents of the list
4
5 stringStream = stringStream.filter(s -> s.length() == 5);
6 stringStream = stringStream.map(s -> s.replace('e', '@'));
7 stringStream.forEach(System.out::println); //prints P@t@r P@tra
```

Durch Verkettung von Methodenaufrufen kann man diesen Code auf den folgenden reduzieren:

```
1 List<String> stringList = new Linkedlist<>();
2 stringList.add(Arrays.asList("Peter", "Paul", "Petra"));
3
4 stringList.stream().filter(s -> s.length() == 5)
5                     .map(s -> s.replace('e', '@'))
6                     .forEach(System.out::println);
```

Lambdas!

Lambdas werden häufig in `Streams` eingesetzt. Streams sind eine neue Strategie(seit Java 8), über `Collections` zu iterieren:

```
1 List<String> stringList = new Linkedlist<>();
2 stringList.add(Arrays.asList("Peter", "Paul", "Petra")); //shorthand to
      add multiple elements at once
3 Stream<String> stringStream = stringList.stream(); //"streams" the
      contents of the list
4
5 stringStream = stringStream.filter(s -> s.length() == 5);
6 stringStream = stringStream.map(s -> s.replace('e', '@'));
7 stringStream.forEach(System.out::println); //prints P@t@r P@tra
```

Durch Verkettung von Methodenaufrufen kann man diesen Code auf den folgenden reduzieren:

```
1 List<String> stringList = new Linkedlist<>();
2 stringList.add(Arrays.asList("Peter", "Paul", "Petra"));
3
4 stringList.stream().filter(s -> s.length() == 5)
5                     .map(s -> s.replace('e', '@'))
6                     .forEach(System.out::println);
```

->DEMO<-

Übung

Übung

1. (Rekursion) Implementiere eine Funktion, die die Quersumme einer Zahl berechnet, mittels Rekursion.

2. (Lambdas/Streams) Implementiere eine Funktion, die die Inhalte einer Liste ausgibt
   a) mittels Iterators und `while`
   b) mithilfe von Streams und Lambdas.